# Questions for Each Module

Module 1: Architecture

1. **What is an Enterprise?**

An enterprise is any organization like company or business with an objective and intention to gain profit.

1. **What is Architecture?**

An architecture is a framework and set of guidelines to build a new system. It is a series of principles, guidelines and rules that determine the logical and physical interrelationship between components.

1. **What is Enterprise Architecture?**

Enterprise Architecture is a conceptual blueprint that defines structure and operation of an organization to determine how an organization can achieve its current and future objectives. It involves the practice of analysing, planning, designing and eventually implementing the analysis on enterprise.

1. **Why do you need an EA?**

An enterprise architecture is necessary to build a strategy for the organization such that all the business components of the organization work together effectively to achieve the same strategic business goal.

1. **Who is an Enterprise Architect?**

An enterprise architect analyses the business structures and processes to ensure that they align with the business goals effectively. Understanding and keeping in view what the long term strategy of the organisation should be, the enterprise architect designs the enterprise architecture for the organization such that all the components(processes, products, data, people, technologies) of the organization work together effectively to achieve the same strategic business goal.

1. **What is integration architecture?**

Integration architecture is a software architecture that facilitates the integration of multiple IT components. Business objectives is hard to achieve if the systems won’t talk to each other. So APIs are built to achieve this. Integration architecture enables us to process the application and data objects within our IT architecture while providing and using interfaces.

1. **Difference between EA and Integration architecture?**

An integration architecture determines the best practices and integrations patterns for an organization. In contrast, an enterprise architecture oversees architecture domains like application, data, and technology and ensures they align with an organization's standards.

1. **What is a framework?**

An architecture framework is an encapsulation of a minimum set of practices and requirements for artifacts that describe a system's architecture. Models are representations of how objects in a system fit structurally in and behave as part of the system.

1. **How are frameworks useful for EA?**

An enterprise architecture framework (EA framework) defines how to create and use an enterprise architecture. An architecture framework provides principles and practices for creating and using the architecture description of a system.

1. **What is TOGAF?**

TOGAF stands for ‘The Open Group Architectural Framework’. It is an IT management framework developed by the U.S. Defence Department, that offers high-level framework for enterprise software development. It helps to organize the development process through a systematic approach aimed at reducing errors, maintain timelines, staying on budget and aligning IT with business units to produce quality results. It explains rules for developing good principles, rather than providing a set of architecture principles.

[Phases of Architectural Development Method (ADM):

* Preliminary Phase
* Phase A: Architecture Vision
* Phase B: Business Architecture
* Phase C: Information Systems Architectures
* Phase D: Technology Architecture
* Phase E: Opportunities and Solutions
* Phase F: Migrating Planning
* Phase G: Implementation Governance
* Phase H: Architecture Change Management
* Requirements Management ]

1. **What is a Zachman?**

Zachman Framework is a formal and structured way of viewing an organization. It has a two-dimensional classification schema with rows representing distinct view of the organisation and the columns representing the questions asked to the enterprise.

This framework does not provide guidance on sequence, process or implementation. It focuses on all views are well established. It ensures a complete system regardless of the order in which they were established.

Columns(dimensions) – Data(what), Function(how), Network(where), People(who), Time(When), Motivation(why)

Rows(perspectives) – Scope(planner), Business model(owner), system model(designer), technology model(builder), detailed representations(sub-contractor), Functioning enterprise(user)

1. **What is 4+1 View point?**

4+1 View Model separates the different aspects of a software system components into different views of a system. It describes software architecture using five concurrent views, each of which addresses a specific set of concerns – Logical view, Process view, Physical view, Development view and Scenario.

[

Logical View – Class diagram, Object Diagram

Process View – Sequence diagram, activity diagram, state diagram, communication diagram

Physical View – Deployment diagram, network topology

Development view – Component diagram, Package diagram

]

1. **When to use TOGAF, Zachman and 4+1 view point?**

Each framework has a different intent and purpose.

A TOGAF is used for mission critical applications which give utmost importance to standards, governance and meeting business objectives. Organizations that need structured and organized approach to minimise the errors, maintain timelines and stay on budget use TOGAF.

Zachman provides a formal and structural way of viewing and defining enterprise. Organizations that want to understand and define every artifact/asset/resource of the enterprise use this framework.

4+1 view point is used when an organization wants to separate different aspects of a software system components into different views of a system based on the stakeholders. This model is used for software-intensive systems.

1. **Real time use-case of using TOGAF framework?**

Department of Social Security (DSS-UK) has separate IT systems for each of its agencies. DSS uses a framework that is based on and is an addition to TOGAF framework to ensure that each IT system can use their own architecture independent of vendor while the core strategy and standards remain the same.

1. **Real time use-case of using Zachman framework?**

Sydney Medical School, University of Sydney wanted to convert their normal face-to-face classes to online e-learning system while continuing their current education program. They needed to develop 13 units that consisted of 52 modules in a span of one year. In order to make the modules consistent in terms of instructional design throughout the e-learning program , they used Zachman framework so that the academics prepare online content in the acceptable format. Each cell in the framework was a small unit of work supported by detailed planning documentation. This framework helped the university to come up with a detailed roadmap featuring the guidelines and course content submission.

1. **Real time use-case of using 4+1 View point?**

Different stakeholders always have different interest in a system. Some aspects are relevant to developers, some aspects are relevant to system administers. Following 4+1 view model makes architecture modelling easier as it makes it easier to organise. Also this allows the architects to understand which view should be prioritised when there is not enough time to model every single diagram possible in an architecture.

1. **When will you use API Led connectivity?**

API led connectivity can be used for application systems that connect with external sources or data through APIs. This approach allows businesses to easily accommodate new technologies, do more and understand their customers better, innovate faster.

1. **What is logical view, development view, process view, physical view?**

**Logical** **view** provides the end user perspective defining the services provided by the system to the user. Eg: class diagram, object diagram

**Development** **view** is the developer’s perspective of a system that focuses on actual software module. It helps to understand how code modules work together and also understand the dependencies between modules. Eg: Component diagram, package diagram

**Process** **View** is a system integrator’s perspective of a system, closer to the working of an actual software. The purpose of this view is to capture the sequence, flow and timing of inter-process information exchange. Eg: Activity diagram, sequence diagram, communication diagram

**Physical View** is of infrastructure perspective which defines where an application will be deployed and decide which framework would be followed. Physical architecture considers non-functional requirements like availability, reliability, performance and scalability. It illustrates which node will host which process from the process view. Eg: network topology, deployment diagram

1. **Explain these views with an example?**

**Logical view**

A class diagram defines every parameter and function of each class in an application system. These functions are in-turn the features provided to the end-user.

**Development View**

A component diagram breaks down the actual system under development into various high level functionality components. A component diagram shows how each component in the larger component interact with other elements and what data flows in & out of the component. A clear idea of how each module or component works enables the developer to develop the functionalities easily.

**Process View**

A sequence diagram depicts the order in which each instantiated object interact with each other. This helps the system integrator to understand the flow of the process and determine how successful the outcomes will be.

**Physical View**

A deployment diagram shows the execution architecture of a system, including nodes such as hardware or software environments, and the middleware connecting them. They are typically used to

1. **What tools do you use to create each of the views in 4 + 1 view points?**
2. **What do you achieve or what is the purpose of creating these views in 4+1 view model?**

4+1 View model makes modelling easier to do because it makes it easier to organise. This approach also provides a way for architects to be able to prioritise modelling concerns. It is rare that a project will have enough time to model every single diagram possible for an architecture. Architects can prioritise different views. For example, for a business domain intensive project it would make sense to prioritise the logical view. In a project with high concurrency and complex timing it would make sense to ensure the process view gets ample time. Similarly, the 4 + 1 approach makes it possible for stakeholders to get the parts of the model that are relevant to them.

1. **What is SOA?**

Service Oriented Architecture (SOA) is a layered architectural pattern for building software systems based on loosely coupling system. In this structure the services are provided to other application components through a communication protocol over a network. The most important principle of this architectural pattern is that the communication is independent of vendor and other technologies. Also, the loosely coupling refers to the client of service being independent of the service it requires.

Use-case:

A customer placing a new order may require the coordination of many systems. The business needs to validate the customer ID, verify the customer’s good standing, check inventory, fulfill the order, get a shipping quote, compute sales tax, send a bill, etc. This process can easily span across five or six different systems. From the customer’s perspective, it is a single business transaction.

Example:

There are dozens of big brands implementing SOA into software development — Harvard Medical School, Motorola, NASA, to name a few. The most prominent is the First Citizens Bank. They not only provide services to their own customer base but also cover more than 20 other institutions.

[Services should be discoverable, reusable and easily maintainable.]

1. **What is a micro service?**

Microservices is also known as microservice architecture which structures an application as a collection of loosely coupled services which can be developed, deployed and maintained independently. Each of these services are responsible for discrete tasks (only one business function) and communicate with other services through messaging system like simple APIs to solve a larger complex business problem.

Examples of microservices usage:

Netflix – A team of engineers runs several business operations that are powered by asynchronous orchestration of tasks which rely on microservices. This allows the company to prepare titles for streaming across the globe seamlessly.

eBay – The company runs more than a 1000 microservices granting them the ability to remove all kinds of dependencies and add new functionality without taking the platform down.

PayPal – Everytime a person hits the “make a payment” button, a special microservice app is triggered. This allows the company to modify/update all modules without the need to remove it (i.e continuous support).

1. **Difference between SOA and Microservice?**

Scope: The key difference between SOA and Microservice is its scope. SOA has an enterprise scope whereas microservice has an application scope. SOA is an enterprise-wide approach to software development that takes advantage of reusable software components or services. Microservices are typically used to build individual applications in a way that makes them more agile, scalable, and resilient.

SOA maximise application service reusability whereas Microservices is focused on decoupling.

Governance: SOA follows common governance and standards across the enterprise whereas microservices has relaxed governance with focus on team collaboration and freedom of choice.

Containers: SOA does not commonly use containers like Dockers whereas microservices does.

Threading: SOA is multi-threaded, hence there is more overhead to manage I/O. Microservices is single threaded and uses event loop for non-blocking I/O handling.

Data storage: SOA services share data storage whereas each microservice can have their independent data storage.

Communication: SOA uses Enterprise Service Bus for communication whereas microservices use messaging system.

SOA supports multiple message protocols. Microservices uses light-weight protocols such as HTTP or REST.

1. **What is EDA? with a use-case or example?**

An event-driven architecture uses events to trigger and communicate between decoupled services and is common in modern applications built with microservices. An event is a change in state, or an update, like an item being placed in a shopping cart on an e-commerce website. Events can either carry the state (the item purchased, its price, and a delivery address) or events can be identifiers (a notification that an order was shipped).

It follows publisher/subscriber model of communication and passes messages asynchronously. So, the major components of the model would be the event publisher, event router and event consumer.

For example, consider an e-commerce store where customer have credit limits and a new order cannot exceed their credit limit.

* + 1. When the e-commerce application receives an order requesr, the Order service creates an order in a pending state and publishes an OrderCreated event into the event queue.
    2. The Customer Service receives this event and attempts to reserve credit for that order.
    3. It then publishes either a CreditLimitExceeded event or CreditReserved event.
    4. The Order Service receives the event and changes the status of the order to approved or cancelled.

1. **What is ETL? with a use-case or example?**

## Module 2: Design Centre

1. **What is RESTFul API?**

REST stands for Representational State Transfer. RESTful APIs are based on REST, which is an architectural style and approach to communications used in web service development. It is a set of rules defined for client-server communication that suggests the server-side data be made available through representations of data in simple formats like JSON or XML. REST uses HTTP requests to access and use data.

[The representation of resources or collection of resources are potentially modifiable, with actions or relationships being made discoverable via a method known as hypermedia.]

1. **What is a Non-Restful API?**

Any API that do not follow any one of the six guiding constraints of REST can be called Non-RESTful API. Alternatively, these APIs can be just HTTP APIs. The common indicator of an API being Non-Restful is that it has a single endpoint. REST is about representations of resources, each is represented by its own URI. All the requests made to the API is through POST method rather then using other HTTP methods. POST request is used to create resources, if that is all that is happening it should be okay. But if the POST request to the API retrieves, updates or deletes data, then it isn’t a RESTful Service.

1. **Characteristics of a RESTFul API?**

REST has six guiding constraints that restrict the way server can process and respond to the client requests. These form the characteristics of RESTful API:

* + 1. Client-Server Model: To implement REST, architecture should have a client and server. Loose coupling of client and server enables each to be developed and enhanced independent of the other.
    2. Stateless: It signifies that server must not store any session or application data that may be required to serve the subsequent requests. This ensures visibility, reliability and scalability.
    3. Cacheable: Caching refers to storing server response in the client itself, so that the client does not need to make a server request for the same resource again and again.
    4. Uniform interface: Uniform interface defines the interface between the client and server. It simplifies and decouples the architecture , which enables each part to evolve independently but still be able to communicate. Uniform interface is defined by four main elements –
       1. Resources
       2. Representations
       3. Self-descriptive messages
       4. Hypermedia as the engine of application state (HATEOAS)
    5. Layered System: In a layered system constraint, layers of service are injected between client and server. Each layer can only communicate with the adjacent layer. A typical three-tier system would have client, application server and database server. Layered system allows efficient load balancing and implementing server proxy. Layered system can be useful for caching at boundaries for performance benefits and also applying security policies at boundaries similar to firewall.
    6. Code on Demand: This is optional. Most of the times, servers send static representation of resources in JSON or XML format. Sometimes, servers can temporarily extend or customise the functionality of client by sending across logic to it that can be executed. Examples of this may include compiled components like Java applets or client-side scripts such as javascripts.

1. **What is the use of designing an API?**

Designing an API means providing an effective interface that helps the API’s consumers better understand, use and integrate with them while helping us maintain it effectively. Every product needs a usage manual, and designing an API is no different to that.

Designing an API:

Helps in better implementation

Facilitates incremental development

* + 1. Facilitates better documentation
    2. Improves developer experience

Good API design improves the usability of our API, resulting in higher adoption, fewer headaches, and an overall better chance of success for our API endeavors.

1. **Is it mandatory to design an API? Explain.**

It is not mandatory to design an API. However, it is the best practice to design an API to enable API consumers to better understand and use it effectively. It also helps the developers to implement and improve the API.

1. **What ways can we design a RESTFul API?**

API design should encompass:

* + 1. The structure of resources.
    2. The documentation of the resources.

In Mulesoft, a RESTful API can be designed by creating a design specification using RAML or OAS.

1. **What is RAML?**

RAML stands for RESTful API Modeling Language and is a YAML based language for describing RESTful APIs. It contains all the information necessary to describe RESTful API. It makes it manage the whole lifecycle of API from design to deployment. It describes the resources, responses, and description of each resource.

1. **What is OAS?**

OpenAPI Specification (formerly Swagger Specification) is an API description format for REST APIs. An OpenAPI file allows us to describe our entire API, including:

* Available endpoints (/users) and operations on each endpoint (GET /users, POST /users)
* Operation parameters Input and output for each operation
* Authentication methods
* Contact information, license, terms of use and other information.

API specifications can be written in YAML or JSON. The format is easy to learn and readable to both humans and machines.

1. **What is APIARY?**

APIARY is an API-first platform designed specifically to help companies accelerate and control the design, development and documentation of APIs. APIARY supports multiple description formats: API blueprint and Swagger.

1. **What are the differences between RAML vs OAS vs APIARY?**

Goal:

Open API keeps documentation, client libraries and source code all in sync.

Provide all the necessary information to describe RESTful APIs, thus providing a simpler way to design APIs

Document Format:

Open API- JSON, YAML 1.2

RAML- YAML 1.2

Editors:

Open API- Swagger tools (editor, codegen, UI)

RAML- API Workbench

Code Languages:

Open API - .NET, Ruby, Scala, PHP, Python, Perl Swift, Node.js, Java, JavaScript

RAML- Node.js,Java, .NET, Python, Mule, IOT

1. **When should you use each of these languages?**
2. **Example of RAML and OAS?**

OAS:

An OpenAPI object contains the following fixed fields:

openapi – required. Semantic version number.

Info – inof object required. Metadata about api

Servers – array of server objects, provide connectivity information to target server. Default: “/”

Paths – Required. Paths Object. Available paths and operations for API.

Components – Required. components object. Element to hold various schemas for specification

Info Object Example

{

"title": "Sample Pet Store App",

"description": "This is a sample server for a pet store.",

"termsOfService": "http://example.com/terms/",

"contact": {

"name": "API Support",

"url": "http://www.example.com/support",

"email": "support@example.com"

},

"license": {

"name": "Apache 2.0",

"url": "https://www.apache.org/licenses/LICENSE-2.0.html"

},

"version": "1.0.1"

}

Ref: <https://swagger.io/specification/>

1. **What ways can you design a Non-RESTFul API?**
2. **Explain GraphQL API with example?**

GraphQL is an open-source data query and manipulation language for APIs, and a runtime for fulfilling queries with existing data.

GraphQL is a syntax that describes how to ask for data, and is generally used to load data from a server to a client. GraphQL has three main characteristics:

* It lets the client specify exactly what data it needs.
* It makes it easier to aggregate data from multiple sources.
* It uses a type system to describe data.

With GraphQL, the user is able to make a single call to fetch the required information rather than to construct several REST requests to fetch the same.

A GraphQL query is a string that is sent to a server to be interpreted and fulfilled, which then returns JSON back to the client.

GraphQL Query:

{

subscribers(publication: "apollo-stack"){

name

email

}

}

This query would return the names and e-mails of all subscribers of our publication, Building Apollo, if we built an API for it. Here’s what the response would look like:

{

subscribers: [

{ name: "Jane Doe", email: "jane@doe.com" },

{ name: "John Doe", email: "john@doe.com" },

...

]

}

1. **Explain RPC with example?**

RPC – Remote Procedure Call enables “Call a function on another server.” An API is built by defining public methods; then, the methods are called with arguments. RPC is just a bunch of functions, but in the context of an HTTP API, that entails putting the method in the URL and the arguments in the query string or body.

It uses only GET and POST, with GET being used to fetch information and POST being used for everything else. Example to “send a message” the syntax is as below:



1. **Explain SOAP with an example?**
2. **Explain Async API with example?**

Synchronous/asynchronous APIs are application programming interfaces that return data for requests either immediately or at a later time, respectively. Synchronous/asynchronous APIs provide a way to make immediate or scheduled requests for resources, data or services when available.

The synchronous and asynchronous nature of an API is a function of the time frame from the request to the return of data. In the case of synchronous APIs, the expectation is that there will be an immediate return of data. The application requests data and waits for it until a value is returned.

In the case of asynchronous APIs, the availability of a resource, service or data store may not be immediate. These APIs may provide a callback to the requester when the requested resource is ready. Asynchronous requests are useful in maintaining functionality in an application rather than tie up application resources waiting on a request.

Example:

asyncapi: 2.0.0

info:

title: Hello world application

version: '0.1.0'

channels:

hello:

publish:

message:

payload:

type: string

pattern: '^hello .+$'

Ref: <https://www.asyncapi.com/docs/getting-started/hello-world>

1. **What are the constraints of REST?**

REST has six guiding constraints that restrict the way server can process and respond to the client requests. These form the characteristics of RESTful API:

* + 1. Client-Server Model: To implement REST, architecture should have a client and server. Loose coupling of client and server enables each to be developed and enhanced independent of the other.
    2. Stateless: It signifies that server must not store any session or application data that may be required to serve the subsequent requests. This ensures visibility, reliability and scalability.
    3. Cacheable: Caching refers to storing server response in the client itself, so that the client does not need to make a server request for the same resource again and again.
    4. Uniform interface: Uniform interface defines the interface between the client and server. It simplifies and decouples the architecture , which enables each part to evolve independently but still be able to communicate. Uniform interface is defined by four main elements –
       1. Resources
       2. Representations
       3. Self-descriptive messages
       4. Hypermedia as the engine of application state (HATEOAS)
    5. Layered System: In a layered system constraint, layers of service are injected between client and server. Each layer can only communicate with the adjacent layer. A typical three-tier system would have client, application server and database server. Layered system allows efficient load balancing and implementing server proxy. Layered system can be useful for caching at boundaries for performance benefits and applying security policies at boundaries similar to firewall.
    6. Code on Demand: This is optional. Most of the times, servers send static representation of resources in JSON or XML format. Sometimes, servers can temporarily extend or customise the functionality of client by sending across logic to it that can be executed. Examples of this may include compiled components like Java applets or client-side scripts such as javascripts.

1. **What is connectedness, stateless and unique resources in REST means?**

Connectedness: Sometimes representation of resources means not only stream of bits/data, but it may contain links to other resources. In REST, along with the responses from the server, hypermedia is sent as links to the next possible resource.

Stateless: In REST, the service never relies on information from previous requests. REST statelessness means being free on application state. : It signifies that server must not store any session or application data that may be required to serve the subsequent requests.

[

There are 2 types of states in REST.

1)Resource state - is the current state of a resource on a server at any point of time – and it has nothing to do with the interaction between client and server. It is what we get as a response from the server as API response. Referred to as resource representation.

2)Application state - is server-side data which servers store to identify incoming client requests, their previous interaction details, and current context information.

]

Unique Resources: One of the constraints of REST is having uniform interface across the API. Uniform interface is defined by having four elements one of which is resources. There must be resources with their unique resource identifier i.e. unique url in REST APIs.

1. **What is hypermedia?**

Hypermedia, an extension of the term hypertext, is a nonlinear medium of information that includes graphics, audio, video, plain text and hyperlinks. It gives the next logical resource the consumer might need to visit.

1. **Why do you want to use hypermedia?**

One of the key benefits of using hypermedia is that in addition to providing common parameters for all developers to code within, the server hosting a hypermedia API can also generate a complete list of options (or link relations) available to the client, which can then be easily, unambiguously called upon in future HTTP requests.

1. **What is Richardson Maturity Model? Explain?**

The Richardson Maturity Model is a way to grade your API according to the constraints of REST. The better your API adheres to these constraints, the higher its score is. The Richardson Maturity Model knows 4 levels (0-3), where level 3 designates a truly RESTful API.

Level 0: Swamp of Plain Old XML

* Single URI
* Single HTTP Method (POST)
* Eg: SOAP, XML, RPC

Level 1: Resources

* Many URIs
* Single http method

Level 2: HTTP verbs

* Many URIs
* Many http methods

Level 3: Hypermedia controls

* Combination of Level 2 and HATEOS
* Many URIs
* Many http methods
* Resource describes own capabilities
* Resource describes own interactions

1. **Explain how to use a visual editor in the design centre.**

Anypoint Design Centre’s Visual Editor is a visual interface for scaffolding API specifications in RAML or OAS. To be able to create an application or API specification in design centre, a user needs to have Design Centre Developer permission.

There are three panels in the visual editor in design centre:

Left Panel- which displays summary of the API being developed. It also provides controls that allow us to create resources, data types, and security schemes. These components are all included within the body of the specification and are not created as separate files.

Middle Panel- presents the forms in which we specify the properties of the components of the specification. When the editor first opens, it displays API summary, a form for specifying the main properties of the specification and for documenting it. We can click on the name of an item like resources, datatypes or security schemes in the left panel to open its form in the middle panel to create it. For example, we can click on resources to create a resource. In the middle panel, the form asks for uri, the http method, summary, responses, query parameters, headers.

Right Panel- displays the RAML Viewer and the OAS Viewer, both of which are read-only. We can click the tabs at the bottom of the panel to switch between them. These viewers display the current draft of the specification in either RAML 1.0 or OAS 3.0. As we enter values in the forms that appear in the middle panel, the draft is updated with the values.

This panel also lets us run the mocking service to test endpoints that you have set up.

1. **What are API Fragments?**

API fragments are reusable component of RAML to make the design and build of a reusable API even quicker and easier.

Types of fragments:

* Trait
* Resource Type
* Library
* Data Type
* User Documentation
* Example
* Annotation Type
* Security Scheme

1. **Why do you need Fragments?**

Advantage of building an API spec out of reusable API fragments is that consistency of definitions reduces the effort of implementing APIs.

1. **Explain data type with example?**

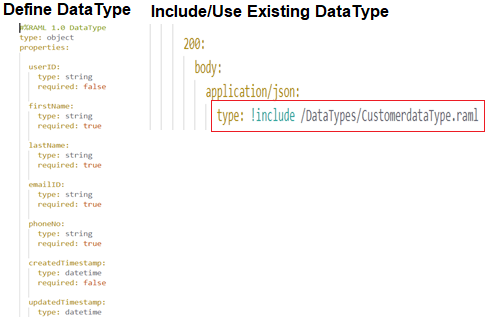
Data types in RAML provide more information for consumers and code generators to ensure they understand and meet all the API data type needs. Data Types can be used in place of schemas and examples, letting us define a data type that can be converted into XML or JSON media type on the fly. It allows us to define a Data type and RAML take care of the REST.

For example, CRUD operations are most common in APIs. If a system has more than one API which has CRUD operations or more than one APIs which require the same parameters, we can define them as Data type.

In a system which has users and user groups, we can define the types of parameters for user and group in the data type as well as the resources types that need to be defined in each API as a collection in the Data type.

Data type is added to an API’s RAML specification through ‘!include’ or ‘types’ keyword at the beginning of the specification.

Eg:



#%RAML 1.0

title: A CRUD API for Users and Groups

mediaType: application/json

types:

###############

# Common:

###############

Email:

pattern: "^\\w+(\\.\\w+)?@company.com"

StaticGroupNums:

description: Predefined user groups

enum: [ 12, 26, 30, 31, 32 ]

DynamicGroupNum:

description: Dynamically-defined user groups

pattern: "D\\-\\d+"

RecordId:

usage: An id of any record in the system

type: number

Record-base:

usage: Pattern for any record in the system

properties:

id:

type: RecordId

required: true

created:

type: date

required: true

###############

# User:

###############

User-base:

usage: The base type for user records

properties:

firstname:

required: true

lastname:

required: true

User-new:

usage: Data for creating a new user

type: User-base

properties:

HRAuth:

description: Authorization received from HR

required: true

User-update:

usage: Changeable properties of an existing user

properties:

firstname:

lastname:

User:

usage: A user in the system

type: [ User-base, Record-base ]

properties:

emails: Email[]

###############

# Group:

###############

Group-base:

usage: The base type for group records

properties:

name:

required: true

Group-new:

usage: Data for creating a new group

type: Group-base

properties:

ITAuth:

description: Authorization received from IT

required: true

Group-update:

usage: Changeable properties of an existing group

properties:

name:

Group:

usage: A group in the system

type: [ Group-base, Record-base ]

properties:

groupnum:

type: StaticGroupNums | DynamicGroupNum

required: true

resourceTypes:

###############

# Collection:

###############

collection:

get:

responses:

200:

body:

properties:

total:

type: number

required: true

members: <>[]

post:

body:

type: <>

responses:

201:

headers:

Location:

required: true

body:

type: <>

###############

# Member:

###############

member:

get:

responses:

200:

body:

type: <>

patch:

body:

type: <>

responses:

200:

body:

type: <>

delete:

responses:

204:

deleteOnlyMember:

delete:

responses:

204:

###############

# API:

###############

/users:

description: All the users

type:

collection:

typename: User

typename-new: User-new

/{userId}:

description: A specific user

uriParameters:

userId: RecordId

type:

member:

typename: User

typename-update: User-update

/groups:

description: The groups to which this user belongs

type:

collection:

typename: Group

typename-new: RecordId

/{groupId}:

type: deleteOnlyMember

delete:

description: Remove this user from this group

/groups:

description: All the groups

type:

collection:

typename: Group

typename-new: Group-new

/{groupId}:

description: A specific group

uriParameters:

groupId: RecordId

type:

member:

typename: Group

typename-update: Group-update

/users:

description: The users belonging to this group

type:

collection:

typename: User

typename-new: User-new

/{userId}:

type: deleteOnlyMember

delete:

description: Remove this user from this group

1. **Explain Traits with examples?**

Trait is like a function and is used to define common attributes/set of codes for HTTP method

‘is’ keyword is used for its implementation.



1. **Explain Library with example?**

Libraries provide the ability to bring in pre-defined sets of data-types, resourceTypes, traits, security schemas, and reusable assets - all in a namespaced environment.



1. **Explain Examples with examples?**

When defining API specification, we give examples for responses or request parameters. These examples can be long arrays or objects which clutters the specification. So to make the API specification more readable and to make the example re-usable, we can extract the examples as a fragment and use it in the specification through ‘!includes’ keyword.

Eg: We can externalise the example as RAML file or YAML file.

api.raml

#%RAML 1.0

title: test

types:

A:

properties:

givenName: string

familyName: string

examples: !include fragment.raml

fragment.raml

#%RAML 1.0 NamedExample

fullName:

givenName: ”Chiaki”

familyName: "Mukai"

1. **Difference between an example and examples?**

Example facet is used to include a .yaml file for a single example, and an examples facet to include a .raml NamedExample fragment file.

1. **What is data-modelling?**

Data modeling in software engineering is the process of creating a data model for an information system by applying certain formal techniques.

Data modeling is a process used to define and analyze data requirements needed to support the business processes within the scope of corresponding information systems in organizations. Therefore, the process of data modeling involves professional data modelers working closely with business stakeholders, as well as potential users of the information system.

It occurs at three levels:

* Physical Model: It is a schema how data is stored in database
* Conceptual Model: Identifies the high level, user view of data
* Logical Model: It is between Physical and Conceptual Model and allows for a logical representation of data to be separated from physical storage.

1. **What are the different approaches to create data-models?**

Different approaches to creating data models are:

* Canonical Data Model
* Enterprise Data Model
* Mirror as Backend System
* Bounded Context Model

1. **What is a canonical data model?**

Often, the data exchanged across various systems rely on different languages, syntax, and protocols. Canonical data models are a type of data model that aims to present data entities and relationships in the simplest possible form in order to integrate processes across various systems and databases. It can be defined as the solution to minimize transformations and dependencies when integrating applications that use different data format.

1. **What is Enterprise wide data-model?**

Enterprise data modeling (EDM) is the practice of creating a graphical model of the data used by an enterprise or company. In other words, It is an integrated view of the data and related processes that consumes and store data across the organizations. It is a single integrated definition of data unbiased of any single system or application. It is independent of “how” the data is physically sources, stored, processed or accessed.

Typical outputs of this activity include an Enterprise Data Model consisting of entity–relationship diagrams (ERDs), XML Schemas (XSD), and an enterprise wide data dictionary.

1. **What are the advantages and disadvantages of an enterprise wide data-model?**

Benefits of using EDM are

* It standardizes concepts and definitions across applications.
* Drives applications towards greater uniformity.
* Seeds data models for new applications.
* Reach beyond the current data to the anticipated future data.
* Provide a grand data model as a basis for application integration and data exchange.

Disadvantages:

* Time-taking

EDM has lot of information and resources, more than that can be delivered.

1. **What is a bounded context data model?**

Bounded Context is a logical boundary. It defines a tangible boundary of applicability of some sub-domain. It is an area where some sub-domains make sense and others don’t. It is a way of designing large domain with independent sub-domains.

Bounded Context originated from Domain-Design Driven pattern which is about dealing with large models and teams. It promotes an object-model-first approach to a service, defining a data model that a service is responsible for and is "bound to." In other words, the service owns this data and is responsible for its integrity and mutability.

For example, in an organization, we have sales team and support team. An organization as a whole would be processing a lot of data. Each team in the organization handle different data and to different levels. A sales team is more interested in the prices, inventory details, discounts, and number of sales of a product, whereas the support team would be interested about the specifications and features of the product. So, a product means different in each context. The organization would have atleast two bounded context known as Sales contect and Support Context.

1. **What is mirroring a data-model in a backend system?**

Data mirroring is the creation and maintenance of redundant copies of a data. It creates an exact copy of the selected folders and files from the source. The unique feature in this is when we delete a file from the source, that file will eventually be deleted on the mirror backup. This increased data protection.

There are two types of mirroring: Disk Mirroring and Server Mirroring.

Disk Mirroring can be realized by RAID(Redundant Array of Inexpensive Disks) which ensures data is always available.

Server Mirroring ensures that same information is quickly accessed at various sites.

## Module 3: Exchange

1. **What are connectors?**
2. **What are the different types of connectors?**
3. **What is a REST connector?**
4. **What is a SELECT Connector?**
5. **What are premium connectors?**
6. **What are mulesoft certified connectors?**
7. **What are community connectors?**
8. **What is a custom connector?**
9. **How to create a custom connector?**

## Module 4: Developer Portal

1. **What is API Notebook?**

An API Notebook is a web-based tool for building interactive tutorials and examples in a JavaScript scripting workspace. Users do not need to install anything to explore the API from your Notebook. You present the Notebook on your API Portal to users of your API who can explore the API.

1. **What is the use of API Notebook?**

API Notebook is used for API documentation and interactive testing of API using JavaScript.

API Notebook can generate an API client based on a RAML API definition, and through it perform authenticated live calls on a real server or on just an empty API interface. From the RAML definition, API Notebook obtains information about the API’s resources, methods, required arguments, and other things.

From our API Notebook, users can also create their own API Notebook for personalized testing of our API in their Github gist. Users can execute sequences of calls to an API for testing or other purposes.

[

API Notebook is used to interactively test an API. We do not need to actually request access for the API to test it. However, we need a client to test the API. That can be created by using the code:

API.createClient(alias, url, options? cb?)

The Notebook creates a client for an API when you specify the URL for the API's RAML spec. With that client you can send authenticated requests to the API and receive live data in return. The API Notebook currently supports OAuth1 and OAuth2 as well as unsigned requests.

Create an API client in a Notebook using the global method API.createClient.

API.createClient accepts two parameters: client name and RAML URL. The client name will become a global variable with the client object, and the URL can be any web-accessible RAML specification.

]

1. **How do you request access in API Notebook?**

We can request access to an API from its Asset Portal in Exchange. On clicking on Request Access in the Asset Portal, a pop-window asks for the name of the application for which we are requesting the API. By giving the name of application, on approval we can secure access to the API.

1. **Why do you share resources on public portal?**

Exchange public portals are accessible to anyone using the internet, enabling developers outside of an organization to access that organization’s REST, SOAP, and HTTP APIs.

A public portal is visible to anyone who has the URL of the portal. The URL for a public portal starts with https://anypoint.mulesoft.com/exchange/portals/Domain\_Name/

1. **How do you deprecate a version of API?**

When we open an asset in Exchange, we can click on the options beside asset version, and select ‘Deprecate version’ to deprecate a version of API.

1. **What is an API Community Manager?**

API Community Manager is powered by Salesforce Community Cloud and provides customization, branding, marketing, and engagement capabilities to serve the different needs of your developer audiences. APIs cataloged in Anypoint Exchange can be surfaced in API Community Manager for developers to discover them and use them to manage client applications, API access credentials, and consumption metrics.

1. **What is the use of API Community Manager?**

With Anypoint API Community Manager, you can build and operate communities around your APIs for developers and partners, both inside and outside of your organization, who write applications that use your APIs.

1. **Differences between API Portal vs API Community Manager?**

For API consumers, API portals are the place to register applications, reset credentials, share and interact with API documentation, provide feedback about the quality of the API and report bugs.

API Community Manager allows an organization to collaborate and co-create value of API ecosystem by understanding the needs of developers, partners and employees. API Community Manager combines a full-featured API portal and industry-leading digital experience capabilities, transforming how teams collaborate across the entire lifecycle of an API program.

## Module 6: Runtime Manager

1. **What is a Control plane?**

The Anypoint Platform control plane provides a set of cloud services that simplify the design, reuse, and management of integrations and APIs. It is includes the features of Anypoint Design Centre, Exchange and Anypoint Management Centre. It stores the following data received from the Runtime engine:

* Metadata
* API analytics information
* CPU utilization
* JVM utilization
* Application jar
* Logs of cloudhub

1. **What is a runtime plane?**

The Anypoint Platform runtime plane is where applications are deployed, and also where the Mule runtime engine and other application-related services, such as Anypoint Connectors run. The runtime engine includes Anypoint Security edge policies and tokenization, MQ, Object Store, and Connectors.

1. **Features of the control plane?**

Control Plane provides programmatic access to network administration. In Anypoint Platform, the control plane consists of Anypoint Design Center, Anypoint Management Center, and Anypoint Exchange. There are several features of Control Plane like:

* Provides necessary tools to design an API
* publish and make an API discoverable to others
* manage the permissions of an API
* control the resources assigned to an API
* analyze, monitor, alerts and logging

1. **Features of runtime plane?**

Runtime plane is where the mule application is deployed. It provides the necessary infrastructure and technology along with Mule runtime on which the application is run. It stores logs and payload and passes the runtime logs to control plan to analyze the data.

1. **What is network latency?**

Latency is a measure of delay. In a network, latency measures the time it takes for some data to get to its destination across the network. It is usually measured as a round trip delay - the time taken for information to get to its destination and back again.

1. **What is a throughput?**

**Throughput** is the rate of production or the rate at which something is processed.

### CloudHub Deployment

1. **What is CloudHub?**

CloudHub is an integration platform as a service (iPaaS) where we can deploy sophisticated cross-cloud integration applications in the cloud, create new APIs on top of existing data sources, integrate on-premises applications with cloud services, and much more.

1. **How to deploy an application to CloudHub?**

An application can be deployed to CloudHub in three ways:

* Deploy an application from Runtime Manager
* Deploy an application from Anypoint Studio
* Deploy an application from command line

1. **What happens behind the scenes when you deploy an application to CloudHub?**

When we deploy an application to Cloudhub, an EC2 instance is created in the Cloudhub for the Mule application with its own Mule runtime engine instance. This EC2 instance is known as a worker in Cloudhub. Each worker is isolated from other applications in the cloud. All the logs generated on running the application are stored within the worker itself. Each worker has certain memory assigned to it to process data.

1. **Advantages of deploying applications to CloudHub?**

* Provide users a fully managed, globally available, secure and highly available cloud platform to deploy an application on.
* Easy deployment
* Allows easy horizontal and vertical scaling.
* Ensures high availability with clustering feature
* If the application is working on multiple workers, cloudhub provides a shared load balancer which shares load automatically balancing incoming traffic.
* CloudHub provides a logging service for customizing log levels, searching logs, and downloading logs.

1. **Disadvantages of CloudHub deployment model?**

When deploying to CloudHub, keep in mind the following limitations:

* CloudHub blocks outbound SMTP traffic when more than 20 emails are sent in one hour.
* CloudHub deployment from Flow Designer fails when the external identity is set up.

1. **What is a Shared Load Balancer (SLB)?**

Load balancer balance, improve and distribute workload across multiple computing resources.

SLB is a shared resource, shared between customers and common for a specific CloudHub region.

1. **What are the characteristics of a SLB?**

SLB is used for load balancing external APIs. SLB supports HTTPS protocol but the user can not add it's custom certificates, sub-domains and can't expose custom vanity domain names.

1. **What is an EC2 instance?**

An EC2 instance is a virtual server in Amazon's Elastic Compute Cloud (EC2) for running applications on the Amazon Web Services (AWS) infrastructure. AWS is a comprehensive, evolving cloud computing platform; EC2 is a service that allows business subscribers to run application programs in the computing environment.

1. **What is AWS and how CloudHub is related to AWS?**

AWS is an cloud platform that enables us to select the operating system, programming language, web application platform, database, and other services we need. With AWS, we receive a virtual environment that lets us load the software and services our application requires.

CloudHub uses Amazon AWS for its cloud infrastructure. The MuleSoft and AWS partnership enables customers to accelerate cloud adoption and seamlessly connect with data and workflows from any system, whether it's in AWS or on-premises or in a SaaS application.

1. **What is zero downtime deployment?**

Zero downtime deployment in Cloudhub means re-deploying an application without interrupting its services. This is achieved in case of multiple workers, where the requests are routed to an actively running worker until the new worker is up to take requests.

1. **What is a Worker?**

An application instance in cloudhub where the application is deployed is known as a worker. A worker has the capabilities of running an API and processing data with its own Mule runtime instance. Each worker is isolated from other workers in Cloudhub and managed independently.

One worker is equal to 0.1 vCores + 500 MB Heap memory and 8GB storage.

1. **What is an availability zone?**

An Availability Zone (AZ) is one or more discrete data centers with redundant power, networking, and connectivity in an AWS Region. Each group of logical data centers is called an Availability Zone.

There are several CloudHub Servers hosted in different regions. Each region has many servers hosted in different data centers known as availability zones. CloudHub control plane is hosted in US and EU region.

1. **What is a region w.r.t AWS data centers?**

AWS has the concept of a Region, which is a physical location around the world where it cluster data centers. Each group of logical data centers is called an Availability Zone. Each AWS Region consists of multiple, isolated, and physically separate AZ's within a geographic area. CloudHub offers worker clouds in 12 different regions.

1. **What is a \*\*vCore\*\* while deploying applications to CloudHub?**

A unit of compute capacity for processing on CloudHub, which is equal to one virtual core. Up to ten Mule Applications can be deployed for every VCore purchased.

1 vCore gives 1.5 GB of heap memory.

1. **What is application isolation?**

Application isolation is the separation of one program or application stack from the rest of the running processes. This is realised in CloudHub using workers which run in isolation with other workers in the same worker cloud.

1. **What is vertical scalability?**

Vertical scalability is the ability to increase the capacity of existing hardware or software by adding resources - for example, adding processing power to a server to make it faster. Vertical Scaling means adding more power or compute resources like memory, CPU in existing servers.

1. **How do you achieve vertical scalability?**

In MuleSoft, Vertical scaling can be achieved by increasing worker size in CloudHub.

1. **What is horizontal scalability?**

Horizontal scalability is the ability to add more number of machine into our pool of resources. It is is the ability to connect multiple entities so that they work as a single logical unit.

1. **How do you achieve horizontal scalability?**

In MuleSoft, Horizontal scaling can be achieved by increasing number of workers in CloudHub.

1. **What is high availability?**

High Availability (HA) describes systems that are dependable enough to operate continuously without failing. They are well-tested and sometimes equipped with redundant components.

It is the measure of a system’s ability to remain accessible in the event of a system component failure. Generally, HA is implemented by building in multiple levels of fault tolerance and/or load balancing capabilities into a system.

1. **How can we achieve high availability in CloudHub deployment?**

High availability can be achieved in CloudHub deployment through Clustering of worker nodes. Cluster is set of mule servers that provide a logical deployment unit, high availability and failover capabilities through distributed shared memory.

1. **How can we achieve disaster recovery in CloudHub deployment?**

We can use a load balancer (cloud or on-premises) for applications deployed to different regions to provide a better disaster recovery strategy.

1. **When should customers choose CloudHub deployment model?**

Cloudhub Deployment is a very good solution, if each application is designed to be a separate, encapsulated service, providing one business functionality. Applications are not related to one another, so any change made to one app does not impact other ones. Such services can have separate auto-scaling procedures, running separate mule connector versions or even separate Mule Runtime Versions (both 3.x and 4.x in the same Anypoint Platform environment).

Cloudhub is also a very good start point for migration between Mule versions, or running a proof of concept or seasonal services, without a need to provision the whole environment.

### CloudHub VPC

1. **What is a VPC?**

A Virtual Private Cloud (VPC) allows us to virtually create a private and isolated network in the cloud.

VPC stands for Virtual Private Cloud and it allows us to create logical or isolated networks in the cloud where we can deploy or run the resources securely.

1. **What is CloudHub VPC?**

MuleSoft CloudHub is a multi-tenant integration platform as a service. An Anypoint VPC is a logically private and isolated network hosted inside CloudHub. AnyPoint VPC allows us to create an isolated network where we can host the workers or mule applications.

1. **Why do you need a VPC?**

VPC is needed to create an isolated network for an organization where they can host the workers or mule applications securely. CloudHub VPC is part of CloudHub managed services which allows us to deploy, run and manage our applications in a dedicated and secure environment.

1. **Advantages of having a VPC?**

* Create a secure virtual network within CloudHub.
* Connect CloudHub to assets behind the firewall.
* Deploy mule runtime securely.
* Connect Cloudhub to any public cloud or on-premise data center securely.

1. **Disadvantages of having a VPC?**

* Pricing – Owning and running a VPC can be very expensive considering the cost of the VPC, cost of data flow and cost of private connection per hour to the VPC.
* Latency – Latency can be an issue when using VPCs over open internet through private connection depending on the physical point of presence, chattiness of the application and type of VPN encryption.

1. **How do you configure a VPC?**

A VPC can be created from Anypoint Runtime Manager by providing the Name of VPC, CIDR block and the environment that needs to be part of VPC. Also, select the region which is closer to the datacenter or AWS region to minimise latency.

1. **What is Dedication Load Balancer?**

A dedicated load balancer is not shared resource in CloudHub and resides inside the client's VPC, user can configure more than one DLB in VPC.

1. **What are the features or characteristics of DLB?**

As DLB is not a shared resource in CloudHub, So users can write custom URL mapping rules and run multiple API endpoints under the same domain name. It enable vanity URL, that is unique URL to market the brand.

1. **Differences between SLB and DLB?**

Shared Load Balancer

* Shared Load Balancer is available in all environments by default.
* Shared Load Balancer provided basic functionality like TCP load balancing.
* Shared Load Balancer doesn’t allow you to configure custom SSL certificates and proxy rules.
* Shared Load Balancers have lower rate limits and it is different for each region.
* Application deployed to Cloudhub exceeds the rate limit for shared load balancers, it will return 503 - Service Unavailable.

Dedicated Load Balancer

* One of the limitations of SLB is the lower rate limit. To avoid that issue, you can use a dedicated load balancer.
* All applications can be hosted under a single domain.
* Custom SSL certificates can be configured on DLB and optionally two-way authentication can be enforced.
* Handle load balancing among the different CloudHub workers that run your application.

1. **Advantages of having a DLB?**

* Allows the route of external HTTP/HTTPs traffic to multiple applications deployed to CloudHub within VPC.
* Supports Vanity URL
* All applications can be hosted under a single domain
* Allows one-way and two-way SSL certificate

1. **What are the ways that we can create a secure connection between CloudHub worker and customers' networks?**

* IPsec tunnel with network-to-network configuration: Recommended and most used solution for VPC to on-premise (I,e customer corporate data centres ) connectivity.
* VPC Peering: Pair an Amazon VPC directly to a CloudHub VPC.
* CloudHub Direct Connect: If the customer network connectivity has Amazon VPC using Amazon Direct Connect, they can create a hosted virtual interface to their CloudHub VPC.

1. **What is an IPsec tunnel?**

Internet Protocol Security (IPSec) is a protocol suite for securing communications between two networks. Typically used to connect Cloudhub to customers on premise network.

We can use an IPsec tunnel with network-to-network configuration to connect your on-premises data centers to your Anypoint VPC. An IPsec VPN tunnel is generally the recommended solution for VPC to on-premises connectivity, as it provides a standardized, secure way to connect. This method also integrates well with existing IT infrastructure such as routers and appliances.

1. **What is direct connect?**

This method establishes a dedicated network connection from your Amazon account to your Anypoint VPC. This enables you to create a hosted virtual interface to attach to your Anypoint VPC. To use Direct Connect, our AWS Direct Connect Partner and the Anypoint VPCs must be located in the same region.

[

AWS Direct Connect (DX) lets you create a dedicated network connection between your network and one of the DX locations. Once you have established DX connectivity in your own (or your partner's) AWS account, you can establish connectivity with a CloudHub VPC.

]

1. **What is VPC peering?**

VPC peering provides a connection between two VPCs. In this case, it pairs your private Amazon VPC directly to your Anypoint VPC. This enables you to route traffic between the two VPCs so they can communicate as though they are in the same network.

1. **What is a vanity URL?**

A vanity URL is a descriptive, memorable and pronounceable URL usually used to redirect URLs from one location to another. In the simplest terms, a vanity URL is a long URL that has been converted into a customized short link. A vanity URL can also be known as a branded Link or a custom short URL.

1. **How do you implement a two-way SSL using CloudHub VPC?**

SLB does not support two-way SSL. We need to use a DLB to implement a two-way SSL using CloudHub VPC. We need to configure SSL certificate to enable HTTPS (Public Key and Private Key). For two way authentication, we have to configure Client Certificate and that is optional. By default, the dedicated load balancer must be associated with at least a pair of one certificate.

1. **When should a customer choose a CloudHub VPC deployment model?**

When a customer wants an dedicated and secure environment to deploy, run and manage workers and mule applications, they can opt for Cloudhub VPC deployment model. Also, this allows the customer to have their own vanity URL to market their brand.

[Security, flexibility, branding]

1. **What are the firewall rules of VPC?**

Four VPC Firewall rules created by default are:

8081: Public port and can accept request through SLB over HTTP

8082: Public port and can accept request through SLB over HTTPS

8091: Private port and restricted within VPC (HTTP)

8092: Private port and restricted within VPC (HTTPS)

### Hybrid deployment

1. **What is the hybrid type of deployment model?**

“Hybrid” implementation means self-provisioned Mule Runtime Servers as a deployment target. Servers are provisioned on Cloud or on premises by a solution owner and, by means of using Anypoint Runtime Manager Agent, they can be connected to Anypoint Platform.

The control plane is Anypoint-hosted and the Runtime Plane is customer hosted.

1. **What are the types of hybrid deployment model?**

There are two types of hybrid deployment model where the runtime plane changes:

* Deployed On-premises: Standalone, Runtime Manager REST API, Mule Runtime Agent
* Deployed on Cloud

1. **Advantages of hybrid deployment model?**

* No core limitations per application
* Can deploy many applications as long as the Mule runtime can handle
* Minimises licensing cost compared to Cloudhub
* Provides flexibility and control over on-premise security
* Supports shared configuration among mule applications (Domain project)
* Ability to use Anypoint Control Plane features.

1. **Disadvantages of hybrid deployment model?**

* Load balancing not provided. Need to use external tools for load balancing.
* Runtime Manager logging feature not available. Need to configure on-premises apps to send logs to external logging tools like Splunk and ELK
* Object Store infrastructure not available. We must configure database to store data.
* The scheduling feature is not available for hybrid deployments. To schedule tasks, use the Scheduler endpoint element in your flows.
* After an application is deployed and running, we must apply any security updates for the selected runtime version manually.

1. **What is a standalone deployment?**

In a standalone deployment method we perform a manual deployment of our Mule application to your on-premises Mule instance.

[

Inside the plugin element, we add a configuration for our standalone deployment, replacing the placeholder values with our local Mule runtime engine information.

From the command line in your project’s folder, package the application and execute the deploy goal:

mvn clean package deploy -DmuleDeploy

]

1. **What is a cluster?**

A cluster is a set of up to eight servers that act as a single deployment target and high-availability processing unit. In a cluster configuration, the nodes are aware of one another and share common information and synchronize statuses.

1. **What is a server-group?**

A server group is a set of servers that act as a single deployment target for applications so that you don’t have to deploy applications to each server individually.

Deploying applications to servers in server groups provides redundancy so you can restore applications more seamlessly and quickly, with less downtime.

Unlike clusters, application instances in a server group run in isolation from the application instances running on the other servers in the group.

1. **What is a hazelcast?**

Mulesoft uses HazelCast in High availability cluster as a cluster-cache solution. HAzelcast is responsible for all the nodes in a cluster to communicate with each other. Hazelcast also offers an easy way to pass data or share state from one service to another.

1. **How does node replication happen in a hybrid-based deployment model?**
2. **What is the use of node replication?**
3. **What is distributed memory in a cluster-based deployment and how it is useful?**

The nodes in a cluster communicate and share information through a distributed shared memory grid. This means that the data is replicated across memory in different machines.

1. **What is unicast in a cluster and when should customers choose unicast?**

A unicast cluster requires that you configure the IP addresses of the nodes in the cluster. If a server has multiple interfaces, use the internal IP address that allows the node to communicate directly with other nodes. When all the server nodes are in the same network subnet, customer can use unicast.

1. **What is multicast in a cluster and when should customers choose multicast?**

A multicast cluster comprises servers that automatically detect each other. Servers that are part of a multicast cluster must be on the same network segment. So, when the servers are in the same network but not in the same subnet, the customer can use multicast.

1. **How do you achieve High availability using a Cluster based deployment model?**

High Availability is a method of designing a system to prevent any downtime for the applications that run on it. Cluster has multiple nodes which communicate and share information through a distributed shared memory grid. Whenever, a node is down, the requests are routed to the active nodes to achieve high availability.

1. **When should a customer choose a standalone based hybrid deployment model?**

When the customer wants the application to reside on their own infrastructure and security is the major concern, they can opt for standalone based hybrid deployment model.

1. **When should a customer choose a cluster-based hybrid deployment model?**

When the customer wants to achieve high availability, high scalability and reliability, they can opt for cluster-based hybrid deployment.

1. **When should a customer choose a server-group based hybrid deployment model?**
2. **How do you balance the load in hybrid?**

In hybrid deployment model, we use soft load balancing to balance the load between server nodes. We can use external load balancing tools like nginx or HAProxy to balance the load.

1. **How can we achieve high availability in hybrid?**
2. **How can we achieve vertical and horizontal scalability in hybrid?**

We can achieve horizontal scalability by adding more number of server nodes in hybrid model. Vertical scalability can be achieved by increasing the capacity of each server.

### **Runtime Fabric (RTF)**

1. **What is runtime fabric type of deployment?**

Runtime Fabric (RTF) is a container service to deploy mule runtime on cloud (AWS / Azure) or on a data center (on-premise). It provides all cloud (PaaS) benefits such as high availability, automatic failover, rolling upgrade, etc. RTF is based on Docker containers and Kubernetes for orchestration and has been packaged/maintained in such a way that very little or no knowledge of docker is required to run RTF.

1. **When should a customer choose RTF deployment model?**

There are many reasons customers opt for RTF as their deployment strategy:

* Do not want to expose their data
* Want to maintain their own services to which they can redirect their RTF is it goes down.
* Want to configure their own security.

1. **Advantages of using RTF?**

• Multicloud support - RTF can be installed to any cloud or customer datacenter, and managed via the AnyPoint Platform Runtime Manager

• Application Isolation

• Support for Multiple Runtimes

• Zero downtime upgrades, rollbacks, and scaling

• High availability

• Minimal infrastructure skills required

1. **Disadvantages of using RTF?**

* If a cluster service is destroyed, it can the recovered.
* Object Store persistence not supported. Object Store V2 not supported.
* Few connectors can not be used. For example, File connector can not be used. Instead have to use FTP.
* Can not access any asset outside the container.

1. **Differences between RTF and CloudHub deployment model?**

Host: Cludhub is hosted on cloud by Anypoint platform whereas RTF is hosted on-premises or cloud by customer.

Infrastructure: Infrastructure is owned and managed by Mulesoft in Cloudhub deployment. RTF is owned and managed by customer.

Mule Runtime: Coudhub is managed by MuleSoft and RTF is containerized.

Hardware: Hardware is to be provided by customer in RTF unlike Cloudhub.

1. **Differences between RTF and Hybrid deployment model?**

Mule Runtime: Mule Runtime is hosed on-premises in standalone deployment whereas in RTF it can be hosted on-premises or cloud and it is containerized.

Setup: In standalone, mule server configuration is required , in RTF no setup is required.

Elasticity/Isolation: Apps share resources on the same JVM in case of standalone unlike RTF where apps are in isolation.

1. **What is a Docker container?**

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. In mulesoft a docker image includes OS, Mule runtime, Mule APP/API, Registration script.

1. **Differences between a Docker and a VM?**

A virtual machine (VM) is an emulation of a computer system. Put simply, it makes it possible to run what appear to be many separate computers on hardware that is actually one computer. They are made up of user space plus kernel space of an operating system. Under VMs, server hardware is virtualized. Each VM has Operating system (OS) & apps. It shares hardware resource from the host.

With containers, instead of virtualizing the underlying computer like a virtual machine (VM), just the OS is virtualized. At the low level, a container is just a set of processes that are isolated from the rest of the system, running from a distinct image that provides all files necessary to support the processes. It is built for running applications. In Docker, the containers running share the host OS kernel.

1. **What is Kubernetes?**

Kubernetes is an open-source container-orchestration system for automating computer application deployment, scaling, and management. In RTF, docker container is managed by Kubernetes.

1. **Features of Kubernetes?**

* Automated Scheduling.
* Self-Healing Capabilities.
* Automated rollouts & rollback.
* Horizontal Scaling & Load Balancing.
* Offers environment consistency for development, testing, and production.
* Infrastructure is loosely coupled to each component can act as a separate unit.

1. **What is a controller?**

In Kubernetes, controllers are control loops that watch the state of your cluster, then make or request changes where needed. Each controller tries to move the current cluster state closer to the desired state

1. **What are the characteristics of a controller?**

The RTF controller is in fact a Kubernetes controller, so the orchestration, distributed database, and internal load-balancing capabilities are still being leveraged here. In addition, most control plane communication with AnyPoint Platform also happens here.

1. **What is a worker node?**

In Kubernetes, every cluster has at least one worker node. The worker node(s) host the Pods that are the components of the application workload. The control plane manages the worker nodes and the Pods in the cluster.

1. **What is a POD?**

Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. A Pod (as in a pod of whales or pea pod) is a group of one or more containers, with shared storage/network resources, and a specification for how to run the containers.

1. **How do you achieve application isolation in RTF?**

In RTF, application isolation is achieved by deploying each application in an isolated docker container.

1. **How do you achieve vertical and horizontal scalability in RTF?**

In RTF, vertical scalability can be achieved by assigning more vCores to an application while deploying the application. Horizontal scalability can be achieved by assigning more worker nodes to the application.

1. **How do you achieve high availability in RTF?**

RTF cluster installations are highly available by design, and deploying Mule applications in a highly available replica set or cluster is simple as moving a slider and clicking a checkbox.

1. **How do you achieve zero-downtime in RTF?**

By using the zero-downtime feature provided by Mulesoft as part of the installation in RTF. RTF can also achieve zero downtime. An application container is not destroyed until the new container is deployed and ready to accept requests.

1. **What are the minimum requirements of controller and worker nodes?**

To create an RTF, a minimum of one controller node and two worker node is required.

1. **How do you balance the load in RTF?**

Load balancing is handled by the controller node in RTF.

1. **What control plane capabilities are available for RTF?**

All the control plane capabilities are available to RTF.

### **Private Cloud Edition (PCE)**

1. **What is PCE?**

With Anypoint Platform Private Cloud Edition (Anypoint Platform PCE), you run and manage Mule applications on your local servers using your required security policies.

1. **When should a customer choose a PCE deployment model?**

If an organization has a strict compliance law on data processing within their own data centers then PCE is suitable. In such a case the organizations won’t even allow the flow of meta data outside the premise.

For example industries such as banks, insurance corps, healthe care and government organization have scenarios where they want to more tightly manage their data.

1. **Advantages of PCE?**
2. **Disadvantages of PCE?**
3. **What are the differences between RTF and PCE?**
4. **What control plane capabilities are available for PCE?**

* Anypoint Design Centre excluding flow designer
* Anypoint Runtime Manager
* Anypoint API Manager
* Anypoint Exchange
* Access Management

1. **What are the basic configurations of a PCE?**

There are two types of configuration available in PCE:

* 3-node configuration
* 6-node configuration

1. **How do you install the runtime plane in PCE?**

We need install all the necessary configurations of a runtime plane into the cloud. We need to setup docker and Kubernetes to manage, install Mule runtime to the docker containers.

1. **How do you install the control plane in PCE?**

To use the Anypoint Platform PCE option, you first register your Mule servers with the Runtime Manager agent. Then, from your on-premises Runtime Manager, you can optionally add those servers to server groups or clusters to provide high availability. Finally, you deploy your applications from your on-premises Runtime Manager to either a server, server group, or cluster.

1. **Does PCE support application isolation and why?**

PCE supports application isolation as each application runs in a docker container.

1. **How do you achieve vertical and horizontal scalability in PCE?**
2. **How do you achieve zero-downtime in PCE?**
3. **How do you balance the load in PCE?**

Anypoint Platform Private Cloud Edition must be run in a production environment with multiple servers. To distribute traffic among servers and to restrict access to specific ports, you must install and configure a load balance

### **Pivotal Cloud Foundry (PCF)**

1. **What is PCF?**
2. **When should a customer choose a PCF deployment model?**
3. **Advantages of PCF?**
4. **Disadvantages of PCF?**
5. **Explain the basic architecture of PCF.**
6. **What are the differences between RTF and PCF?**
7. **What are the differences between PCE and PCF?**
8. **What control plane capabilities are available for PCF?**
9. **What are droplets in PCF and how is it used?**
10. **How do you install the runtime plane in PCF?**
11. **How do you install the control plane in PCF?**
12. **Does PCF support application isolation and why?**
13. **How do you achieve vertical and horizontal scalability in PCF?**
14. **How do you achieve zero-downtime in PCF?**
15. **How do you balance the load in PCF?**

### **Runtime Behaviour / Thread pool**

1. **What is a thread?**
2. **What is a thread pool?**
3. **What is staged event-driven architecture (SEDA)?**
4. **What is reactive programming?**
5. **How reactive programming is used in mule?**
6. **What are the features of reactive programming?**
7. **What is backpressure?**
8. **How backpressure is implemented in mule?**
9. **What is automatic back-pressure?**
10. **What is manual backpressure?**
11. **How to configure a manual backpressure?**
12. **What is auto tuning in mule and how does it work?**
13. **What are the different processing types in Mule?**
14. **What is CPU\_LITE processing type?**
15. **Examples of connectors/processors which use CPU\_LITE processing type?**
16. **What is CPU\_INTENSIVE processing type?**
17. **Examples of connectors/processors which use CPU\_INTENSIVE processing type?**
18. **What is the BLOCKING\_IO processing type?**
19. **Examples of connectors/processors which use BLOCKING\_IO processing type?**
20. **What is the Proactor pattern?**
21. **What is Grizzly thread pool?**
22. **What is Grizzly Shared thread pool?**
23. **What is Grizzly Dedicated thread pool?**
24. **What is Java NIO and how is it used in mule?**
25. **What is the selector thread and how is it used in mule?**
26. **What is thread switching?**
27. **Why is thread switching expensive?**
28. **How does mule try to avoid thread switching?**
29. **What is the UBER thread pool?**
30. **What is a pooling strategy?**
31. **How does mule increase t**
32. **What is JVM tuning?**
33. **What is Heap memory?**
34. **What is metaspace memory?**