1

Spec driven development

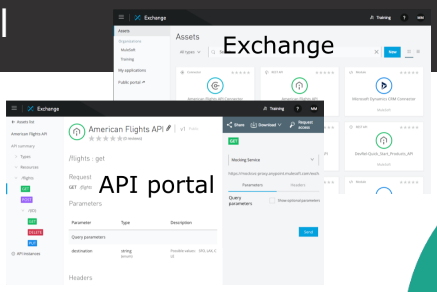
MuleSoft

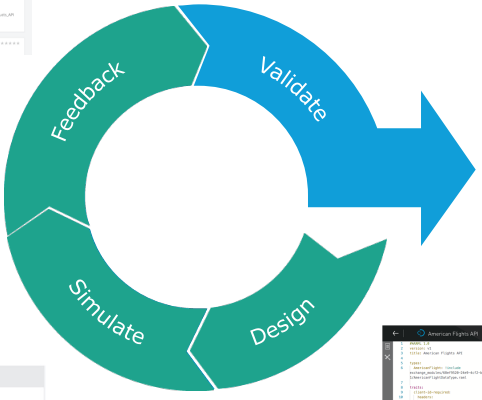
- We discussed in the last modules about the benefits of designing an API first before actually building it
- This is often referred to as **spec driven development**
  - A development process where your application is built in two distinct phases
    - The creation of a spec (the design phase)
    - Development of code to match the spec (the development phase)
- In this module, we'll
  - Create this API specification using a standardized API description language (RAML)
  - Then learn to test it with users without writing any code

2

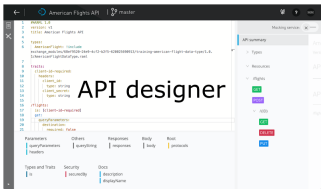
Goal

MuleSoft





API Spec (RAML)



3

At the end of this module, you should be able to

MuleSoft

- Define APIs with RAML, the Restful API Modeling Language
- Mock APIs to test their design before they are built
- Make APIs discoverable by adding them to the private Anypoint Exchange
- Create public API portals for external developers

4

Reviewing the options for defining APIs

5

Approaches to API design

MuleSoft

Hand coding

API Blueprint

apiary

OpenAPI Spec

OPEN {API} INITIATIVE

RAML

RAML

6

Introducing RAML

7

RAML: RESTful API Modeling Language

MuleSoft

- **A simple, structured, and succinct way of describing RESTful APIs**
- A non-proprietary, vendor-neutral open spec
- Developed to help out the current API ecosystem
  - Encourages reuse, enables 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, and more!

RAML

http://raml.org

8

8

RAML syntax

- RAML is based on broadly-used standards such as YAML and JSON
- Uses a human-readable data serialization format where **data structure hierarchy is specified by indentation**
  - Not additional markup characters

```
#%RAML 1.0
version: v1
title: American Flights API

/flights:
  get:
  post:

  /{ID}:
    get:
    delete:
    put:
      responses:
        200:
          body:
            application/json:
```

Notice the indentation used to specify to what each line applies

Defining resources and methods

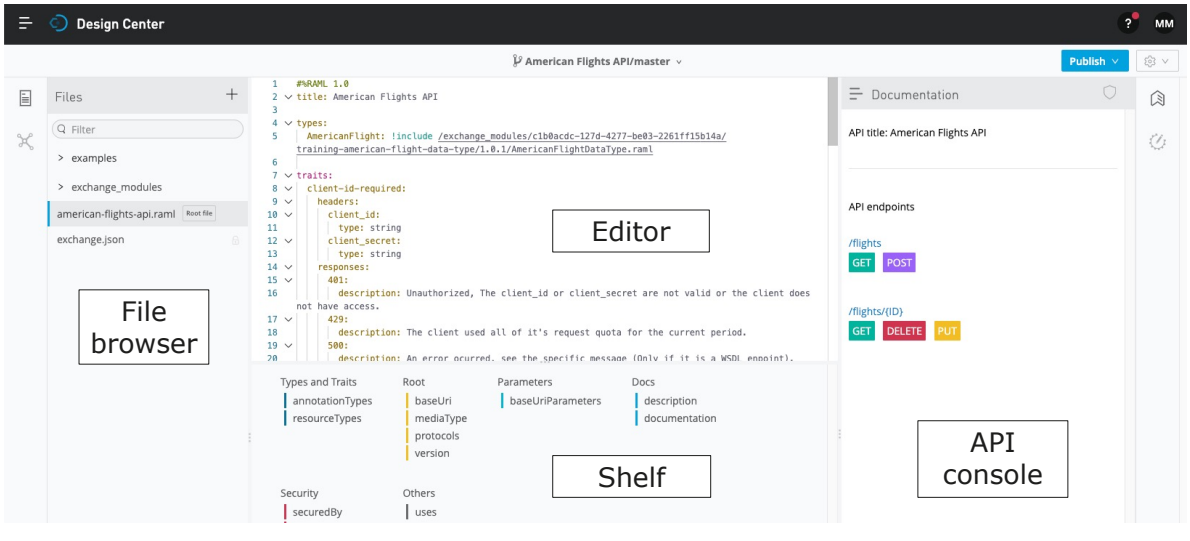
- Resources are the objects identified by the web service URL that you want to act upon using the HTTP method used for the request
- All resources begin with a slash
- Any methods and parameters nested under a resource belong to and act upon that resource
- Nested resources are used for a subset of a resource to narrow it
  - URI parameters are enclosed in {}

```
#%RAML 1.0
version: v1
title: American Flights API

/flights:
  get:
  post:

  /{ID}:
    get:
    delete:
    put:
      responses:
        200:
          body:
            application/json:
```

Using API Designer to define APIs with RAML

11

API Designer



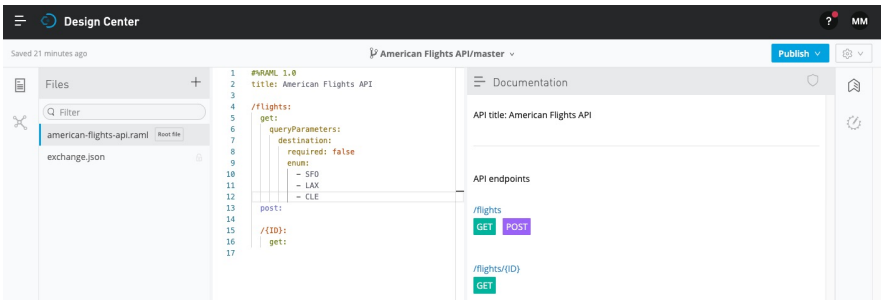
12

12

Walkthrough 3-1: Use API Designer to define an API with RAML

MuleSoft

- Define resources and nested resources
- Define get and post methods
- Specify query parameters
- Interact with an API using the API console

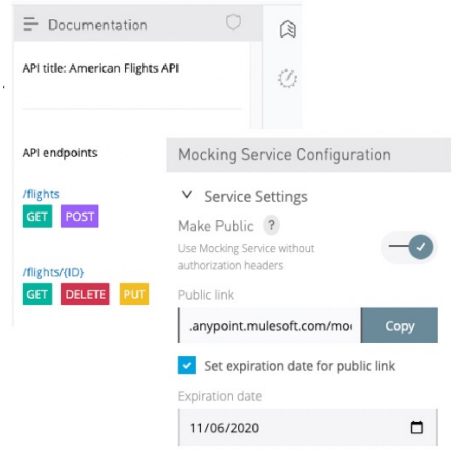


Testing API design without writing code

Simulating an API

MuleSoft

- You can mock an API to test it before it is implemented
  - Useful to get early feedback from developers
- Use the **API console** and the **mocking service** to run a live simulation
  - Returns sample API responses defined in the API definition
- Mocking is available to outside stakeholders through shareable links
  - An expiration date can be set
- The API console is available in
  - **API Designer** – so the API designer can test
  - **API portals in Exchange** – so users/developers can test it

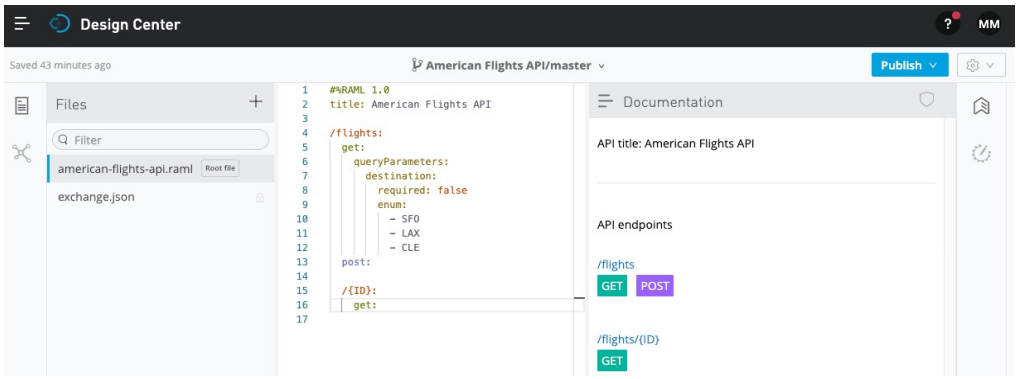


15

Walkthrough 3-2: Use the mocking service to test an API

MuleSoft

- Use the API console to make calls to a mocked API
- Use a shareable public link to make a call to a mocked API



16

Using RAML to define specifications for requests and responses

17

Defining method response details with RAML

MuleSoft

- Responses must be a map of one or more HTTP status codes
- For each response, specify possible return data types along with descriptions and examples

```
/{ID}:
  get:
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight
            examples:
              output:
                ID: 1
                code: ER38sd
                price: 400
                departureDate: 2017/07/26
                origin: CLE
                destination: SFO
                emptySeats: 0
                plane:
                  type: Boeing 737
                  totalSeats: 150
```


18

Defining method request details with RAML

MuleSoft

- For a request, similarly specify the possible request data types along with descriptions and examples

```
/flights:
  get: …
  post:
    body:
      application/json:
        type: AmericanFlight
        examples:
          input:
            code: ER38sd
            price: 400
            departureDate: 2017/07/26
            origin: CLE
            destination: SFO
            emptySeats: 0
            plane:
              type: Boeing 737
              totalSeats: 150
```


Specifying examples

MuleSoft

- There are two optional facets you can use to specify example data: **example** and **examples**

- Use **example** to represent a single instance of the data

```
type: User
example:
  name: Bob
  lastname: Marley
```

- Use **examples** to represent multiple instances of the data as a map of key-value pairs

```
type: User
examples:
  person1:
    name: Paul
    lastname: McCartney
  person2:
    name: Lady
    lastname: Gaga
```

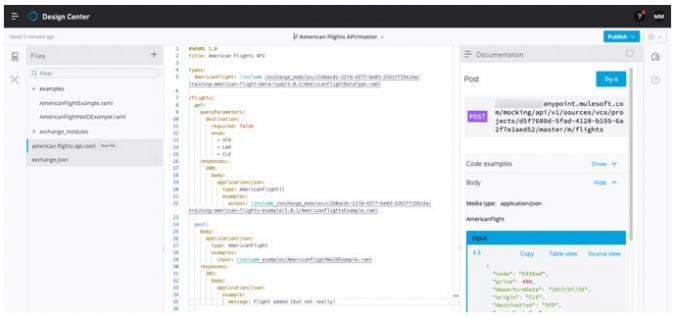
Modularizing APIs

- Instead of including all code in one RAML file, you can modularize it and compose it of reusable fragments
  - **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
  - In a separate API fragment project in Design Center
  - In a separate RAML fragment in Exchange

21

Walkthrough 3-3: Add request and response details

- Use API fragments from Exchange
- Add a data type and use it to define method requests and responses
- Add example JSON requests and responses
- Test an API and get example responses



22

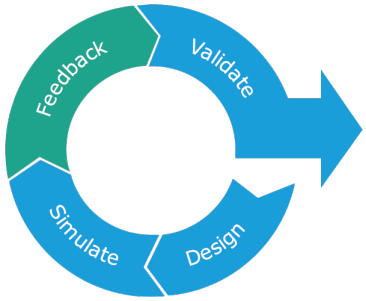
Engaging with users

23

Engaging users during the API design phase

MuleSoft

- To build a successful API, you should define it iteratively
  - Get feedback from developers on usability and functionality along the way
- To do this, you need to provide ways for developers to discover and play with the API
- Anypoint Platform makes this easy with **API portals in Exchange**
  - In **private Exchange** for internal developers
  - In a **public portal** for external developers



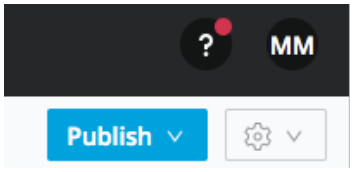
24

24

Publishing RAML APIs to Anypoint Exchange

MuleSoft

- You publish RAML API Specifications and RAML fragments to the Exchange **from API Designer**
  - Not from Exchange itself

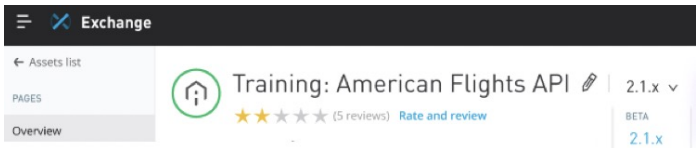


- **API portals** are automatically created for REST APIs added to Exchange
  - An **API console** for consuming and testing APIs
  - An **automatically generated API endpoint** that uses a **mocking service** to allow the API to be tested without having to implement it
- API portals can be shared with both internal and external users

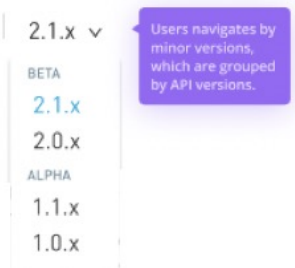
25

Consistent navigation in Anypoint Exchange

MuleSoft



- Provides a uniform and simplified way to navigate the asset portals for all types of assets
  - APIs
  - Templates
  - Connectors
  - Examples
- Navigation is based on **minor version**
- Asset properties are associated with each minor version of the asset
  - Tags
  - Markdown page
  - Categories
  - Custom fields

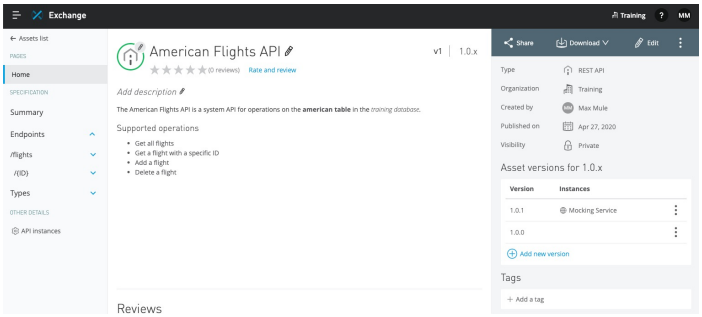


26

Walkthrough 3-4: Add an API to Anypoint Exchange

MuleSoft

- Publish an API to Exchange from API Designer
- Review an auto-generated API portal in Exchange and test the API
- Add information about an API to its API portal
- Create and publish a new API version to Exchange

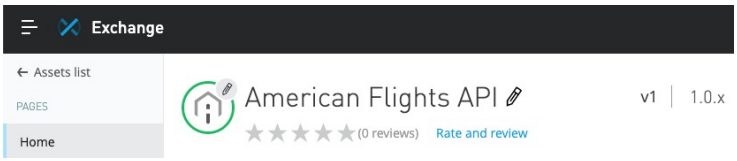


Sharing APIs

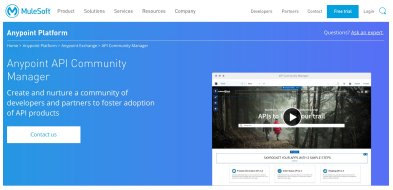
Sharing APIs

MuleSoft

- You can share an API in Exchange with other internal or external users



- Share an API within an org through the **private Exchange**
- Share an API with external users in a **public portal** that you create from Exchange
- Anypoint API Community Manager
  - www.mulesoft.com/platform/api/community-manager
  - Community of developers and partners

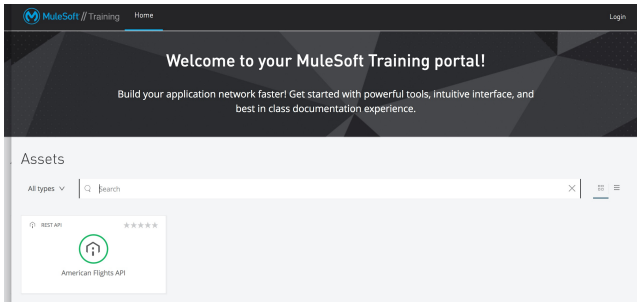


29

Walkthrough 3-5: Share an API

MuleSoft

- Share an API within an organization using the private Exchange
- Create a public API portal
- Customize a public portal
- Explore a public portal as an external developer



30

Summary

31

Summary

MuleSoft

- **RAML** is a non-proprietary, standards-based API description language spec that is simple, succinct, and intuitive to use
  - Data structure hierarchy is specified by indentation, not markup characters
- Use **API Designer** to write API specifications with RAML
- Documentation is auto-generated from a RAML file and displayed in an **API console**
- A **mocking service** can be used in API console to test an API and return the example data specified in RAML

32

32

Summary

MuleSoft

- Make an **API discoverable** by adding it to your **private Exchange**
- **API portals** are automatically created for the APIs with
  - Auto-generated **API documentation**
  - An **API console** that provides a way to consume and test an API
  - 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
  - Selectively share APIs in your org's **private Exchange** with other internal developers
  - Share APIs with external developers by creating and customizing a **public portal** from Exchange and specifying what APIs you would like to include in it

33

RAML resources

MuleSoft

- RAML definitions can be a lot more complex and sophisticated then what we built here
- Training: training.mulesoft.com
  - *Anypoint Platform: API Design*

- Website: raml.org
  - Documentation
  - Tutorials
  - Full spec
  - Resources

34