Effectively Managing Mule API Versions and Portal Site

Yuan Meng – Sr. Solutions Architect, Perficient Inc.
Email: yuan.meng@perficient.com

05/08/2018 San Jose

Mule API

- Powerful API Platform with reach features
- Leader in Gartner’s top magic quadrant
- API Design Center, RAML editor
- Anypoint Exchange
- Mule API runs *In Cloud, on-Premise and Hybrid*

API Versions and API Manager 1.x Challenges

- API Manager 1.x (prior to Crowd Release)
  - The multitude of API versions create challenges to manage the APIs with 1.x
  - A systematic way to manage the versions with CI / CD is strongly desired.

challenge

goal

- API Manager 2.x (Crowd Release and Mule 4 Release)
  - differentiates APIs per each environment
  - Starting API from Design Center and Exchange

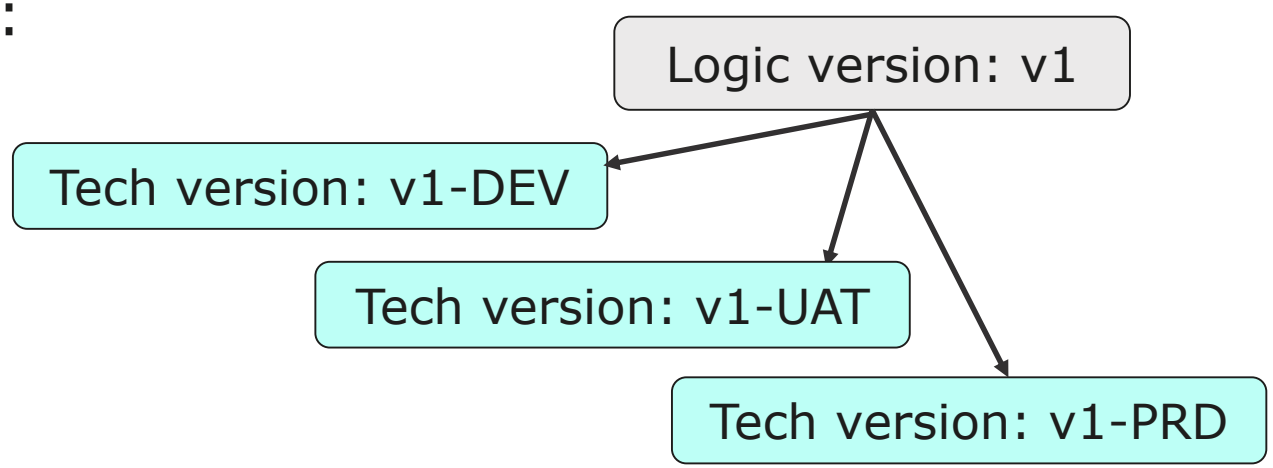
- Knowledge of API Manager 1.x helps understanding and appreciation of API Manager 2.x!

Mule API Logical Version vs. Technical Versions

Every good API starts with a version:

RAML

```
version: v1
/foo:
  get:
/bar:
  post:
```



This *logic version* in RAML declares the API interface (contract).

APIs can go through version changes. When the same API deploys to multiple environments, each environment has a different *technical version,* but they are just an incarnation of the same logic API version.

Technical versions should be hidden from API consumers & developers.

Two Ways to Deploy an API

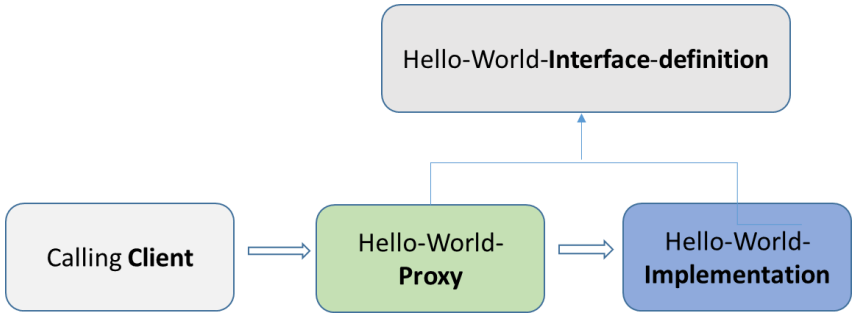


Fig. 1 – Proxy Routes to Implementation

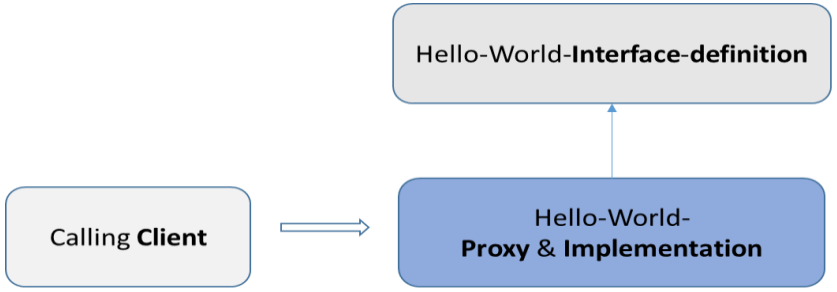


Fig. 2 – Merge Proxy & Implementation

- Modern Web API can still be viewed with the traditional Interface-Proxy-Implementation paradigm
- API RAML is the counter part of traditional interface definition (such as IDL, WSDL, or simply Java interface)
- API proxy and implementation still fit into the classic framework

Two Ways to Deploy an API:

- Endpoint with proxy – fig.1
- Basic endpoint – fig.2

Auto-Discovery – The Glue of Mule API

Auto-discovery is based on 3 key factors:

1. API RAML (technically not required)
2. Auto Discovery Tag: **<api-platform-gw:api apiName**="*myHelloAPI*" **version**="*v1-${env}*" apikitRef="*proxy-config*" flowRef="*proxy*" doc:name="*API Autodiscovery*"/>
3. Environment properties

two key attributes: apiName, version

```
anypoint.platform.analytics_base_uri=https://analytics-
ingest.anypoint.mulesoft.com ...
anypoint.platform.client_id=75xxxxxxxxxxf410
anypoint.platform.client_secret=10xxxxxxxxxx1BF5
```

Cloudhub: defined under properties tab

OnPrem: defined in wrapper.conf

Mule API Manager relies on Auto-discovery to manage the APIs

- Endpoint with a Proxy: auto-discovery is inserted by the API manager
- Basic endpoint: need manually adding auto-discovery

API Version Naming Convention – 1.x

MuleSoft CONNECT San Jose, CA 2018

	myHelloAPI	version	<api-platform -gw:api> tag	Deployment Endpoint URL On-Premise	CloudHub
1	Design Doc	v1			
2	RAML	v1			
3	DEV	v1-dev	v1-${env}	dev-hostname/myHelloAPI/v1	myHelloAPI-dev.cloudhub.io/v1
4	QA	v1-qa	v1-${env}	qa-hostname/myHelloAPI/v1	myHelloAPI-qa.cloudhub.io/v1
5	PROD	v1-prod	v1-${env}	prod-hostname/myHelloAPI/v1	myHelloAPI.cloudhub.io/v1

For the "version" column:
row #1 the version that appears in your design document. This is the **logic version**.
row #2 this is the RAML version, it is same logic version as in the API design doc.
row #3 - #5, these are the **technical versions**.

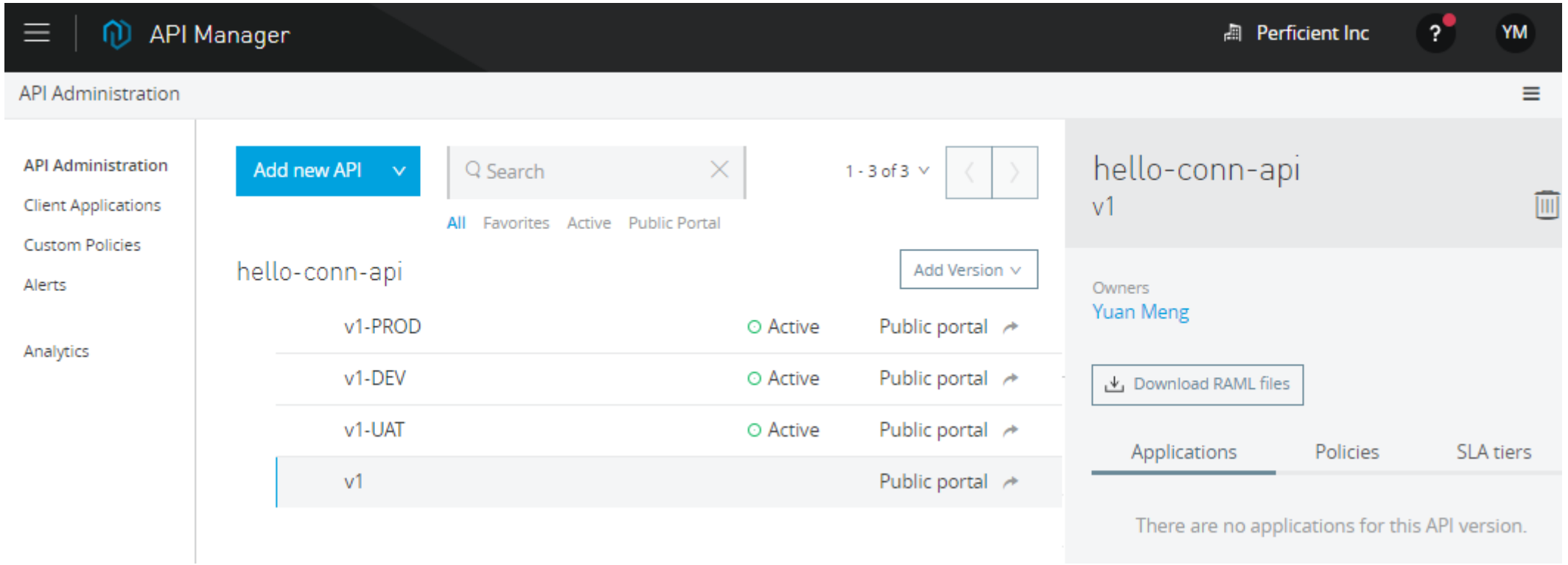
For column 2 and 3:
row #3 - #5: these are the technical versions in your API source code. They are supposed to be the same as the "version" column.

For “On-Premise" and “CloudHub" columns:
row #3 - #5: they should be named consistently as in the previous columns.

POM Snippet for Deployment


```
<plugin>
   <groupId>org.mule.tools.maven</groupId> <artifactId>mule-maven-plugin</artifactId>  …
    <configuration>  <deploymentType>cloudhub</deploymentType>
        <muleVersion>3.8.5</muleVersion> <username>${user}</username>  <password>${pwd}</password>
        <redeploy>true</redeploy><environment>${env}</environment><workerType>Micro</workerType>
        <applicationName>${artifactId}-${env}</applicationName>
        <properties>
                <env>${env}</env>
                <anypoint.platform.analytics_base_uri>…
                <anypoint.platform.platform_base_uri>…
                <anypoint.platform.coreservice_base_uri>…
                <anypoint.platform.client_id>7517bxxxx…
                <anypoint.platform.contracts_base_uri>…
                <anypoint.platform.client_secret>…
        </properties>
   </configuration>
   <executions> <execution>
        <id>deploy</id><phase>deploy</phase>
         <goals><goal>deploy</goal></goals>
   </execution></executions>
</plugin>
```

Viewing the Versions in API Manager 1.x Console



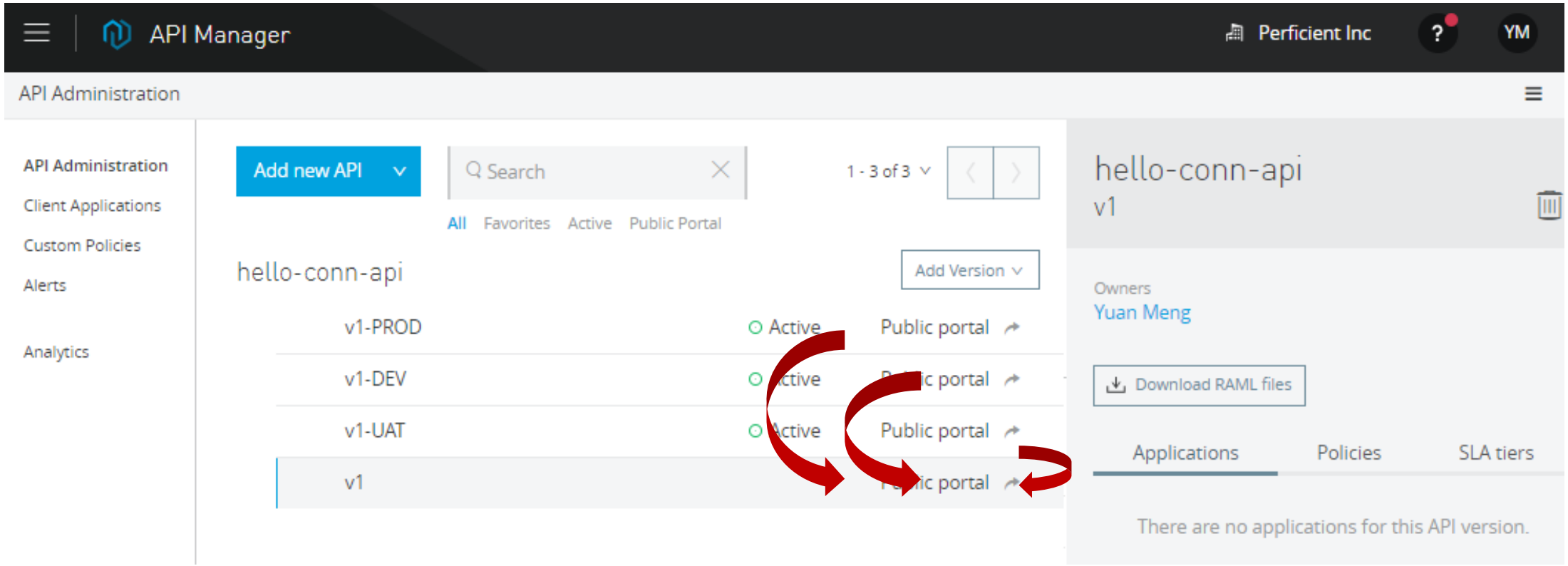
With Maven, CI/CD and API auto-discovery, this version scheme is automated without manually tweaking the API versions from the API Manager console.

The Canonical Portal Site

Each logic API version should have a unique portal site that defines the common interface

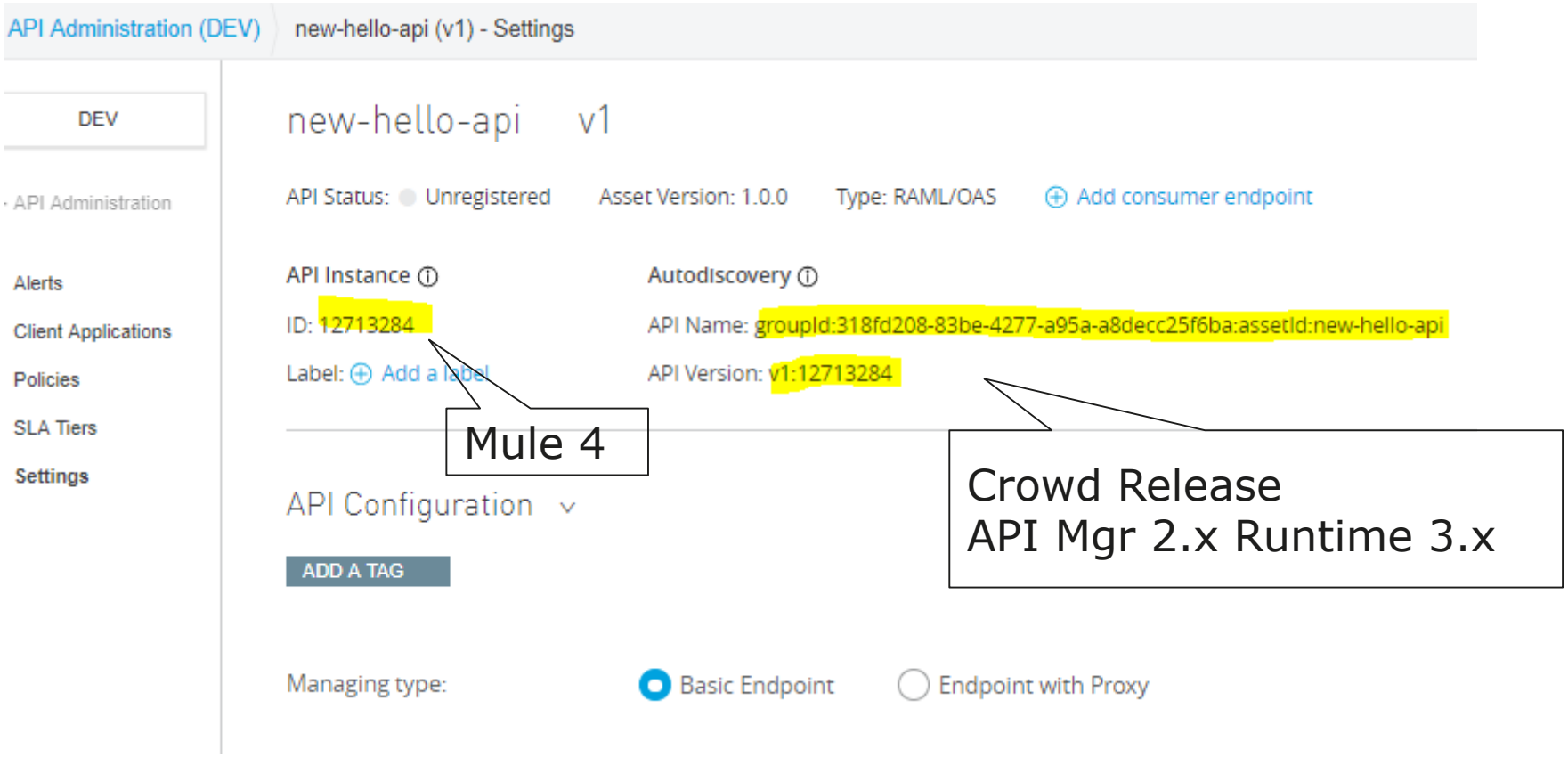
- API for each environment can potentially have its own portal site.
- It is best practice all technical versions (dev-qa-prod) of the same API version (“v1”) share the same portal site.
- It is possible that this canonical API can have portal site, and without any deployment behind it (as shown in the next slide).
- Developers may choose to use one of the “technical” versions (such as “v1-prod”) to establish the canonical portal site, other versions can share the same portal.

Viewing the Portals in API Manager 1.x Console



Example: dev, uat and prod share the same canonical portal site

API Manager 2.x – Crowd Release / Mule 4 Release



References

- https://docs.mulesoft.com/api-manager/v/1.x/api-auto-discovery
- https://blogs.perficient.com/2017/01/27/technically-what-makes-a-mule-application-an-api/
- https://blogs.perficient.com/2017/10/04/what-is-in-a-mule-api-version/

Q&A

Thank you!