

Pilot API Versioning

Workflows

Pilot Flying J

v1.2, 2023-11-27

# Overview

Not all changes require a new version. API versioning strategies can be categorized into two categories: backward-compatible and non-backward-compatible (or “breaking”).

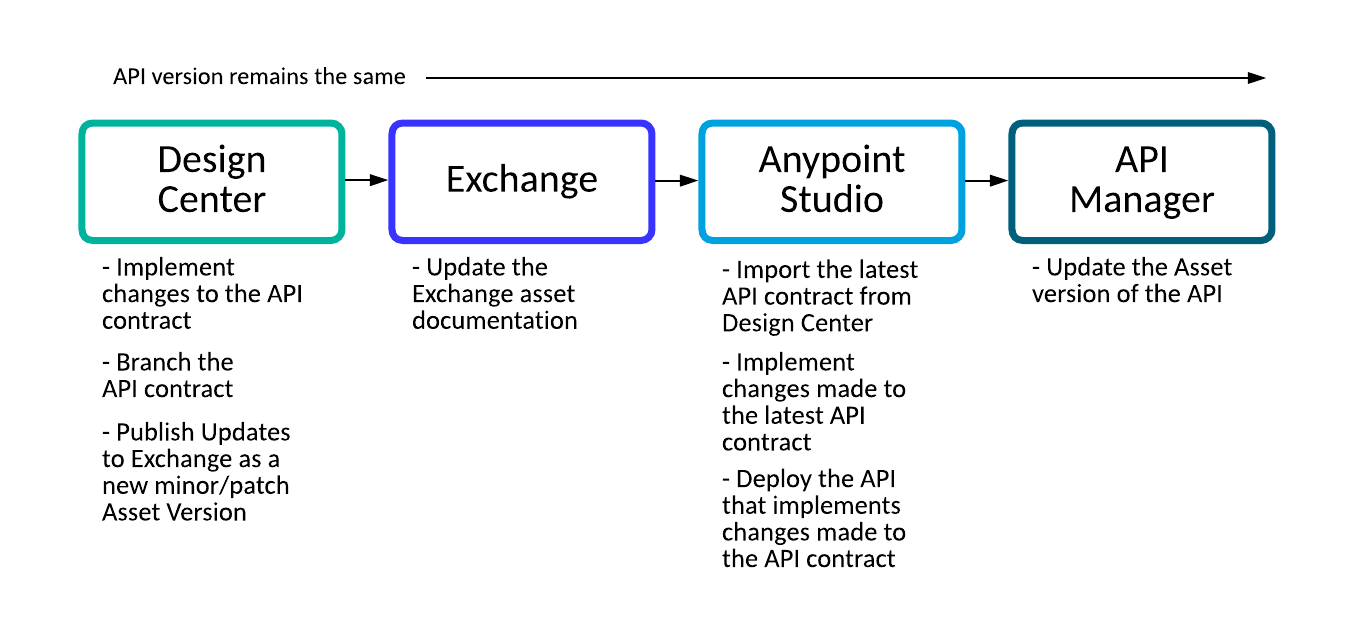
Small changes that are backwards-compatible typically do not require a new version. Since a backwards-compatible change does not change the behavior of the existing API (even though a new resource has been added or the data types supplied by the API request/response have been expanded), the API can utilize the same major version as described. Once the initial major version has been deployed, newer ‘backward-compatible’ deployments can use the same major version.

Changes that are not backwards-compatible usually require the consumer to alter the way their clients interact with the API prior to interaction. Changes that are *not* backward-compatible (or “breaking”) should be well-documented so that potential consumers fully understand the differences between the API major versions. Breaking changes can be captured as a major version change (e.g. v1 to v2).

# API Versioning Workflows

## Backwards-Compatible (“Non-Breaking”) Changes

Since all changes made to the API contract are backwards-compatible, the API version will be preserved. Altering the API contract in this way requires activities to be performed in the following systems:



### Design Center

#### Implement changes to the API contract

In Design Center, make the *non-breaking* changes to the API. These changes could be adding new resources to the API or extending the data types returned by the API.

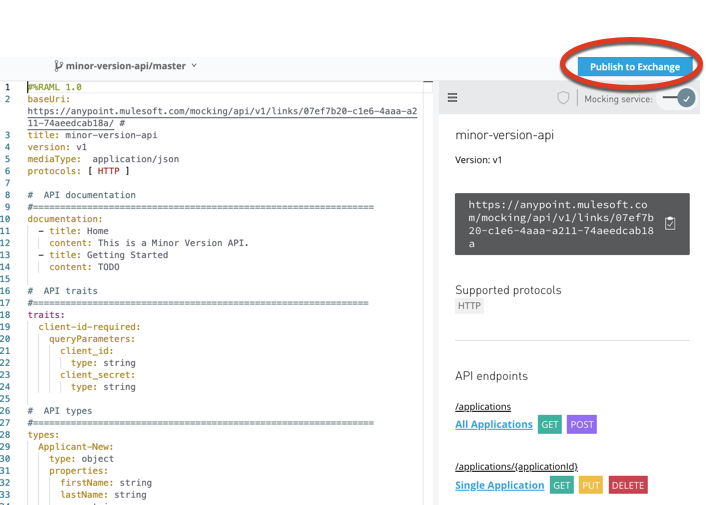
#### Branch the new API contract

Whatever new changes are made to API contract, ensure that a ‘branch’ has been created so that it can be easily identified in Design Center.

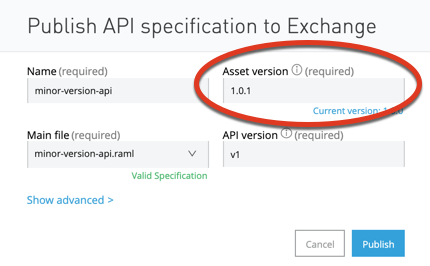


#### Publish API updates to Exchange as a new minor/patch Asset version

Once all changes have been made, publish the API updates to Exchange.



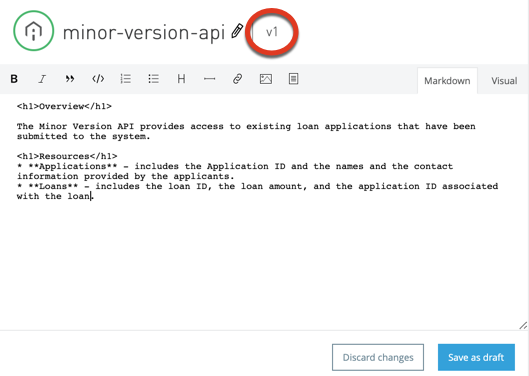
Specify a different Asset version (minor upgrade 1.1.0 or patch upgrade 1.0.1) in which this change will be published.



### Exchange

#### Update the Exchange asset documentation

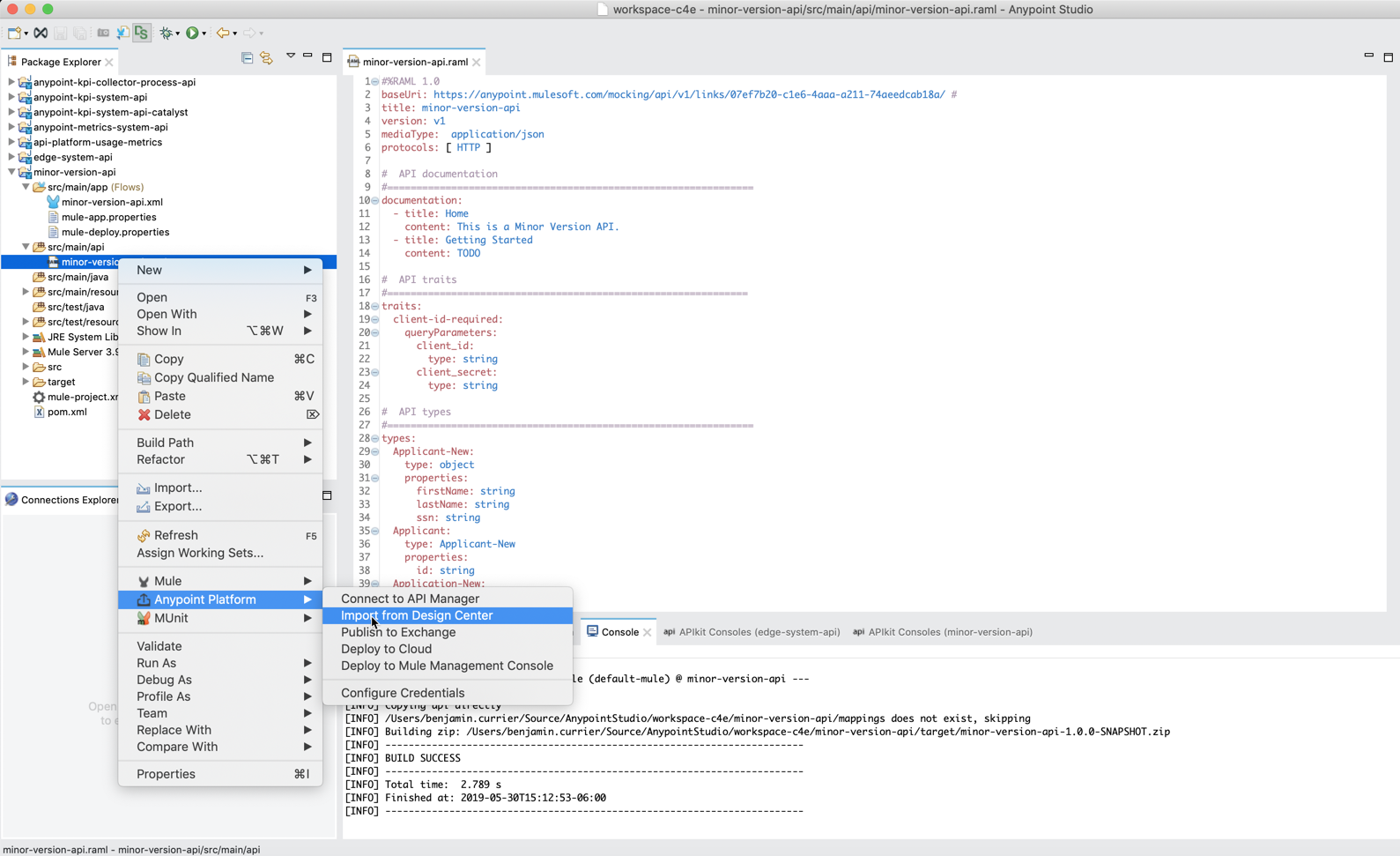
A good practice is to document the changes made to the API contract so that users have an up-to-date understanding of the API’s capabilities. For every change of the API contract, document the reason or business requirement that necessitated the change. This additional information will inform potential consumers of the API’s history and intended purpose.



### Anypoint Studio

#### Import the latest API contract from Design Center

In order to implement the changes made to the API contract, import the desired ‘branch’ version of the API contract from Design Center. This can be done by right-clicking on the project’s RAML file (e.g. minor-version-api.raml) and selecting Anypoint Platform -> Import from Design Center). This step will update the core RAML file as well as any other dependent RAML fragments. This step *should also* update the core API configuration (e.g. minor-version-api.xml) file for the project, but a good practice is to ensure that the changes made to the imported RAML are accurately reflected in the configuration files.



#### Implement changes made to the latest API contract

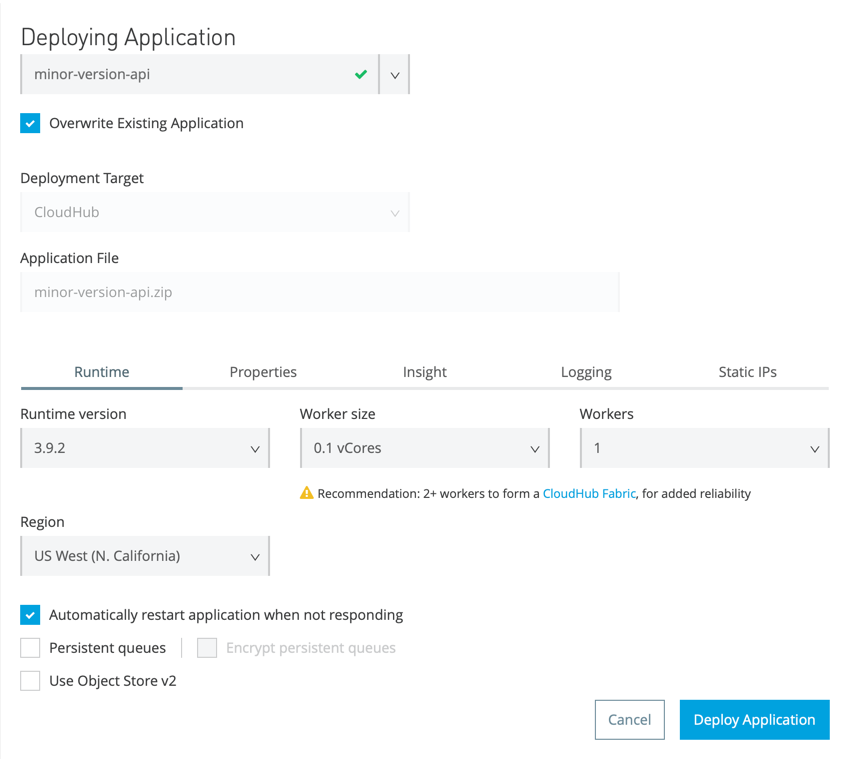
Ensure that the API implementation supports the changes made to the API contract. Please note that the autodiscovery configuration values or the API’s endpoint URL will *not* need to be updated.

#### Deploy the API that implements changes made to the API contract

Once the implementation is complete, upload the modified application to Runtime Manager for deployment. The upload can either be done through Anypoint Studio (see image below), the Mule Plugin for Maven, or through the Runtime Manager APIs.

**Note**: If multiple versions of the API are required in production, consider changing the name of the application during deployment. For example:

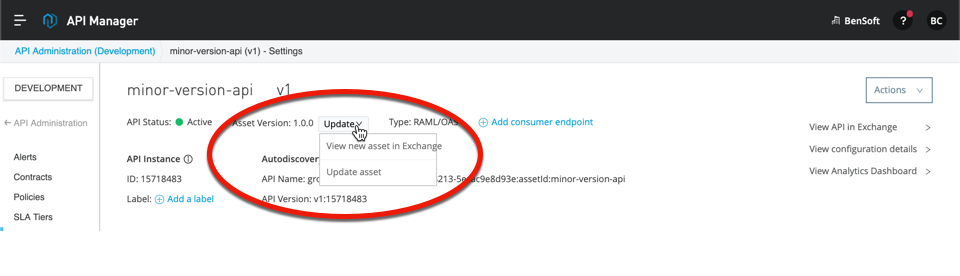
* major-version-api-v1-0
* major-version-api-v1-1



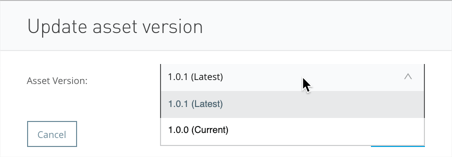
### API Manager

#### Update the Asset version of the API

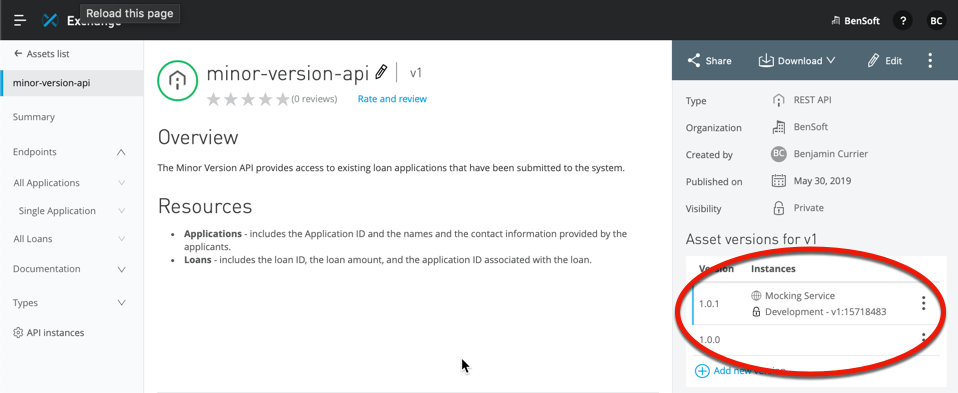
Once successfully deployed and tested, update the asset version of the API in API Manager. Click the dropdown next to the Asset Version of the API and select Update asset.



This selection will bring up the dialog (see below) which allows the user to select the latest asset version (e.g. 1.0.1).

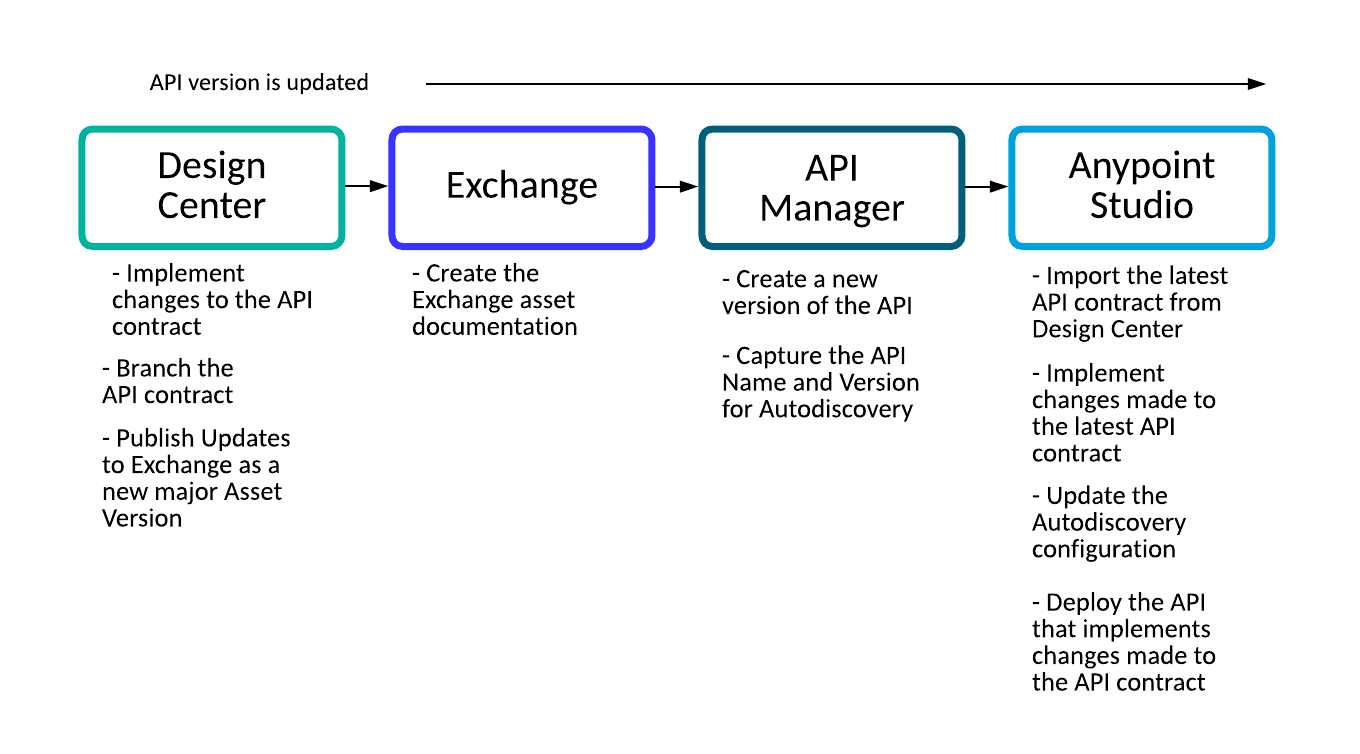


Choosing the latest version will indicate that the implementation is aligned with the most recent version of the API contract. This selection will also update Exchange to show that the API is now supporting v1.0.1 of the API.



## Backwards-Incompatible (“Breaking”) Changes

If the changes made to the API contract will not be backwards-compatible, the API version must be updated. Altering the API contract in this way requires activities to be performed in the following systems:



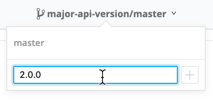
## Design Center

#### Implement changes to the API contract

In Design Center, make the *breaking* changes to the API. These changes could be removing/renaming an existing resource or removing/renaming any the data types returned by the API.

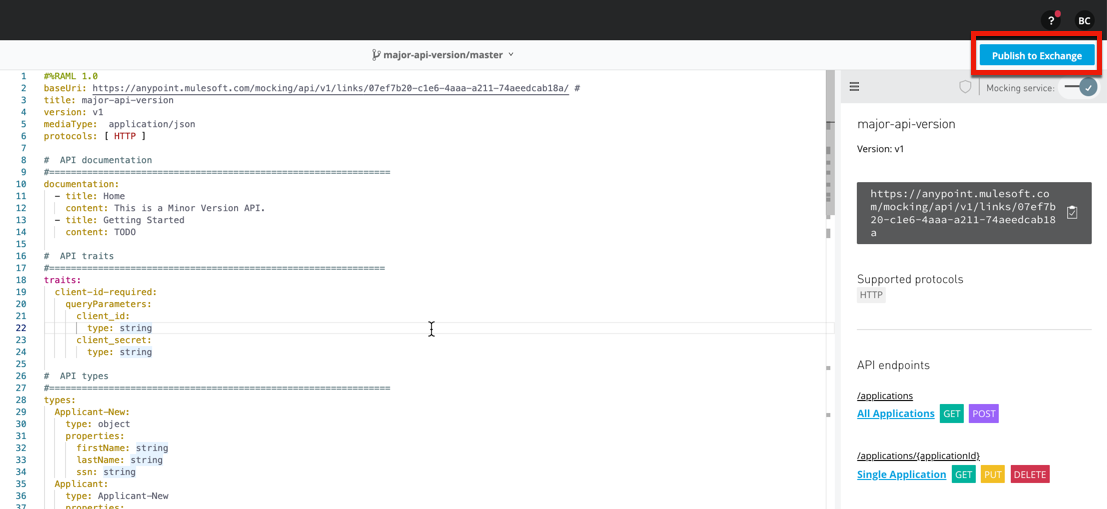
#### Branch the new API contract

Whatever new changes are made to API contract, ensure that a ‘branch’ has been created so that it can be easily identified in Design Center.

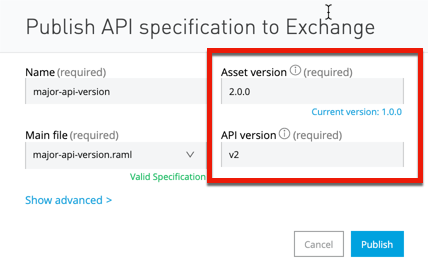


#### Publish API updates to Exchange as a new major Asset version

Once all changes have been made, publish the API updates to Exchange.



Specify a different major Asset version in which this change will be published (e.g. 2.0.0).



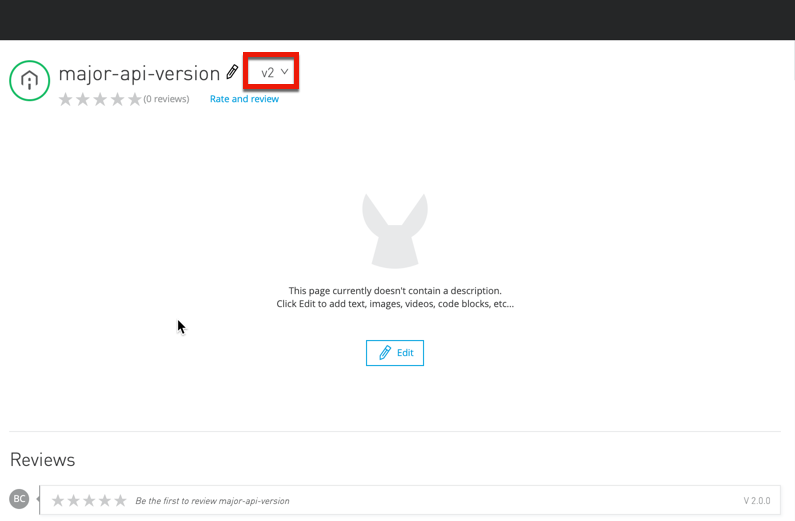
### 

### Exchange

#### Create the Exchange asset documentation

When a new major version of the API is published to Exchange, Exchange interprets the major version change as an opportunity to re-document the asset. The API designer can use this opportunity to document the API as a completely new Exchange asset or as an extension of the previous major version.

A potential consumer can switch between major versions using the dropdown to the right of the API name.

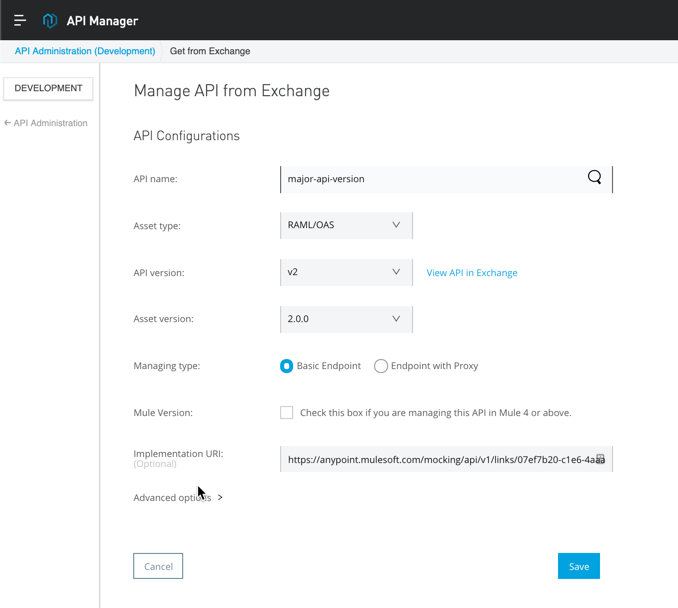


### 

### API Manager

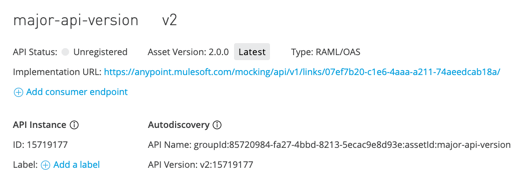
#### Create a new version of the API

Since the API has a new major version, a new version of the API should also be created in API Manager. Ensure that the correct API and Asset version fields are configured for the new major version.



#### Capture the API Name and Version for Autodiscovery

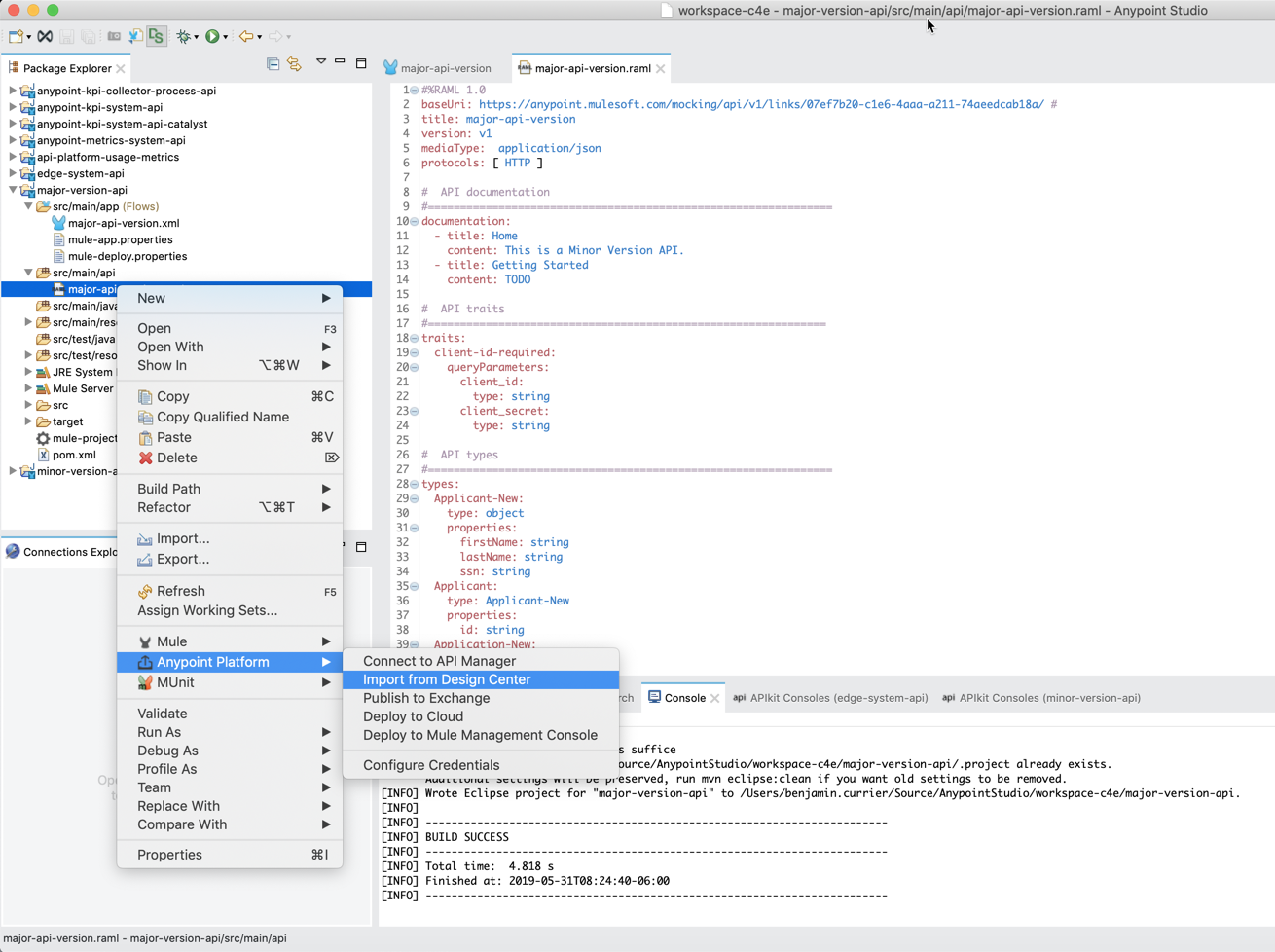
After creation, capture the API Name and API Version that will be used for autodiscovery configuration in Anypoint Studio.



### Anypoint Studio

#### Import the latest API contract from Design Center

In order to implement the changes made to the API contract, import the desired ‘branch’ version of the API contract from Design Center. This can be done by right-clicking on the project’s RAML file (e.g. major-version-api.raml) and selecting Anypoint Platform -> Import from Design Center). This step will update the core RAML file as well as any other dependent RAML fragments. This step *should also* update the core API configuration (e.g. major-version-api.xml) file for the project but a good practice is to ensure that the changes made to the imported RAML are accurately reflected in the configuration files.



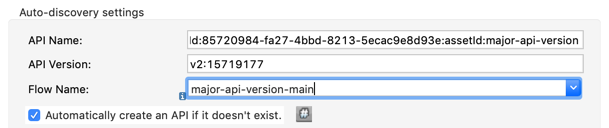
#### Implement changes made to the latest API contract

Ensure that the API implementation supports the changes made to the API contract. Since this is a major version change, the API’s endpoint URL must be updated. This can be done directly in the HTTP Connector properties or in a separate properties file.



#### Update the Autodiscovery configuration

Due to the creation of a new API version in API Manager, the autodiscovery configuration properties will need to be updated. Use the API Name and API Version properties captured in the step *Capture the API Name and Version for Autodiscovery.*

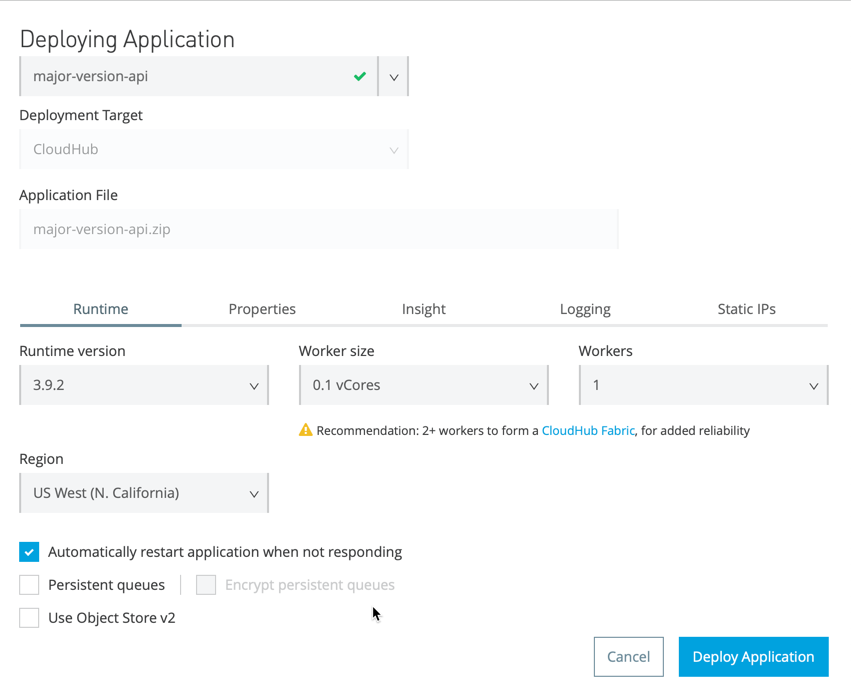
**

#### Deploy the API that implements changes made to the API contract

Once the implementation is complete, upload the modified application to Runtime Manager for deployment. The upload can be done either through Anypoint Studio (see image below), the Mule Plugin for Maven, or through the Runtime Manager APIs.

**Note**: If multiple versions of the API are required in production, consider changing the name of the application during deployment. For example:

* major-version-api-v1
* major-version-api-v2



Once deployment is complete, check API Manager to ensure that the new version of the API has successfully registered with API Manager using the updated autodiscovery properties.

