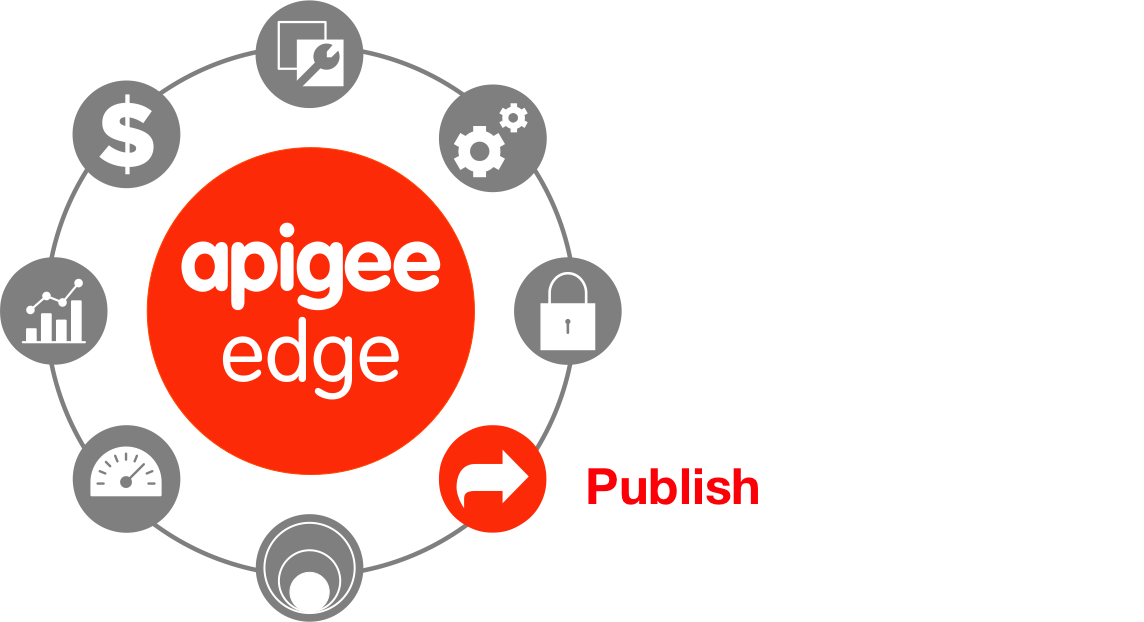


**Lab 3 – Publish APIs**



**Objectives**:

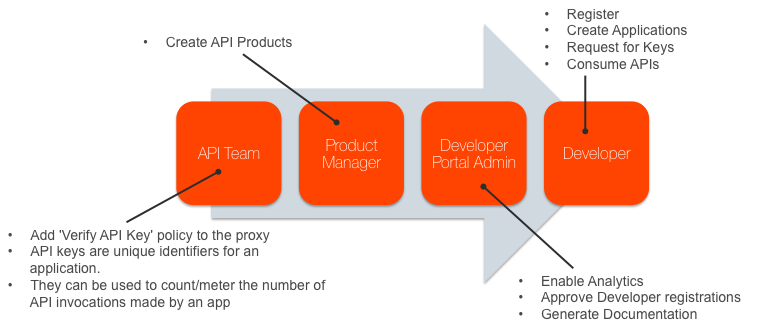
Publishing APIs can be broadly defined by the following tasks:

1. Create the API products on Edge that bundle your APIs.
2. Register app developers on Edge.
3. Register apps on Edge.
4. Provide documentation and community support for your APIs, including:
   * API reference documentation
   * Examples and tutorials
   * Forums, blogs, and other features to foster the developer community

**Prerequisites**

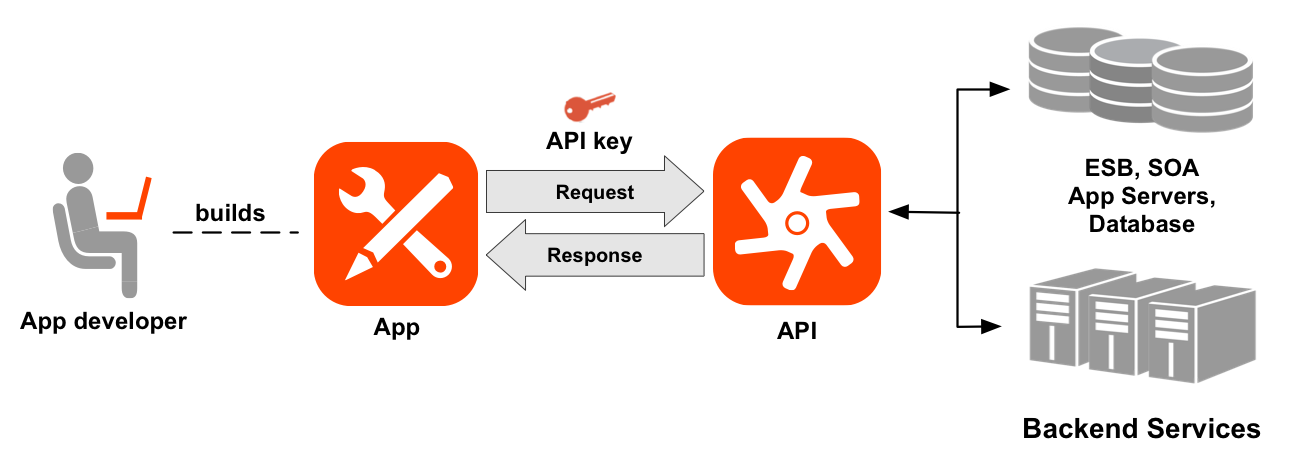
* At a minimum, Lab 1 is completed
* A developer portal is provisioned

**Estimated Time: 60 mins**

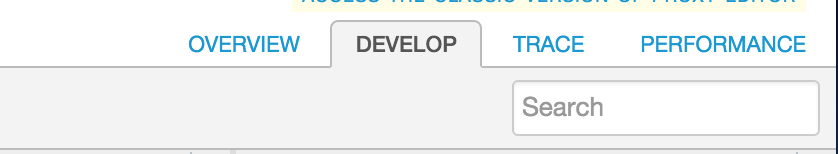


**Adding API Key Verification:**

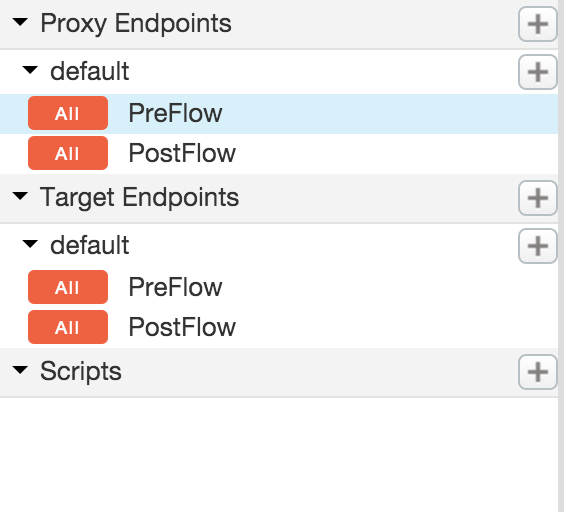
A *developer* builds an *app* that makes requests to your *APIs* to access your backend services. The following image shows the relationship of a developer to an app and to an API:



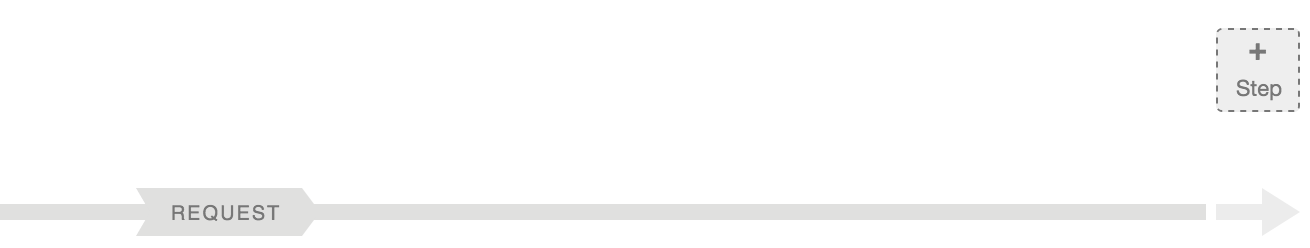
1. **Adding an API Key Verification Policy** 
   1. Go to the Apigee Edge Management UI browser tab
   2. Select your API proxy
   3. Click on the develop tab



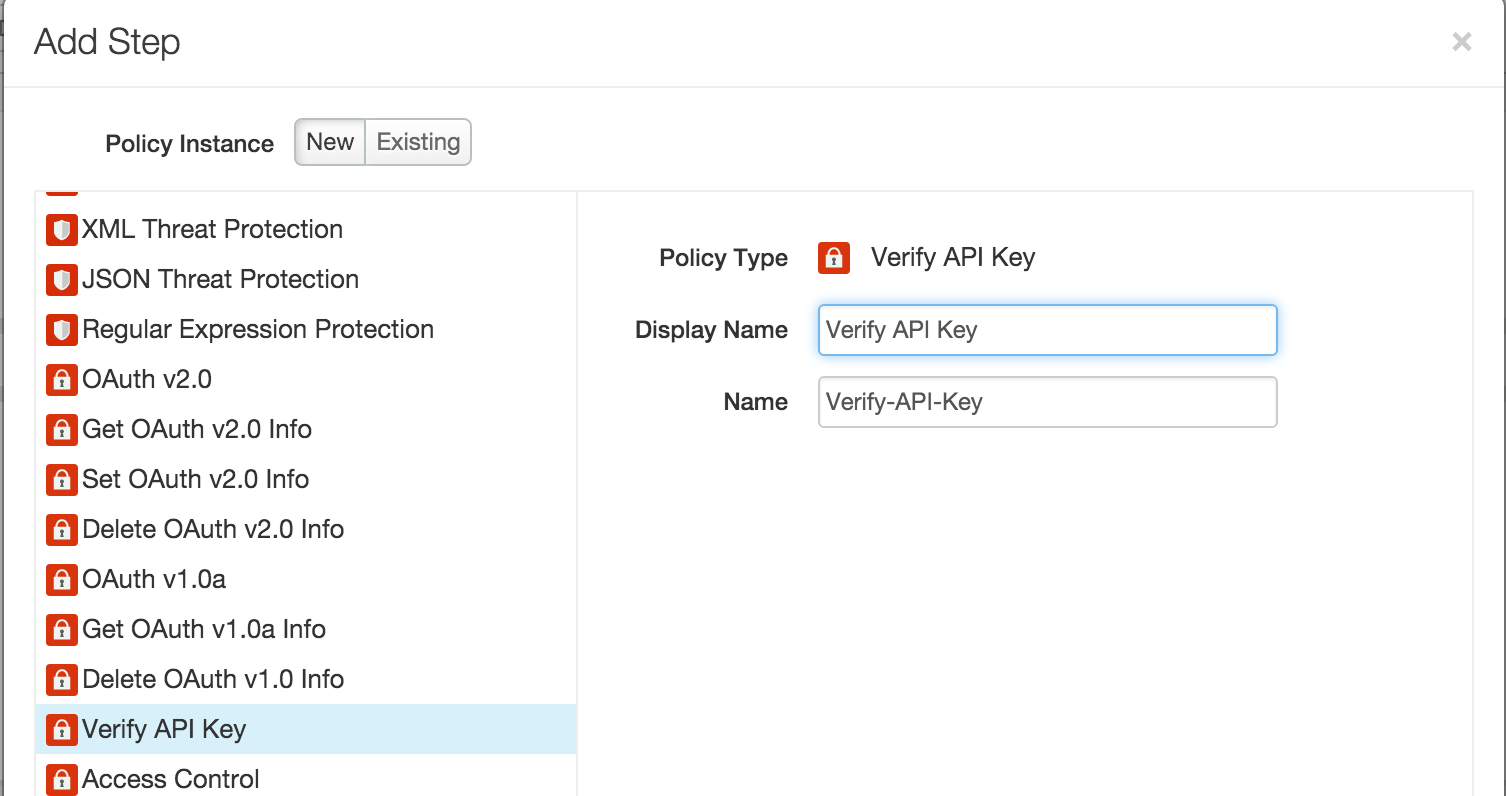
* 1. Click on Proxy Endpoints -> PreFlow



* 1. Click on “+ Step” on the Request Flow

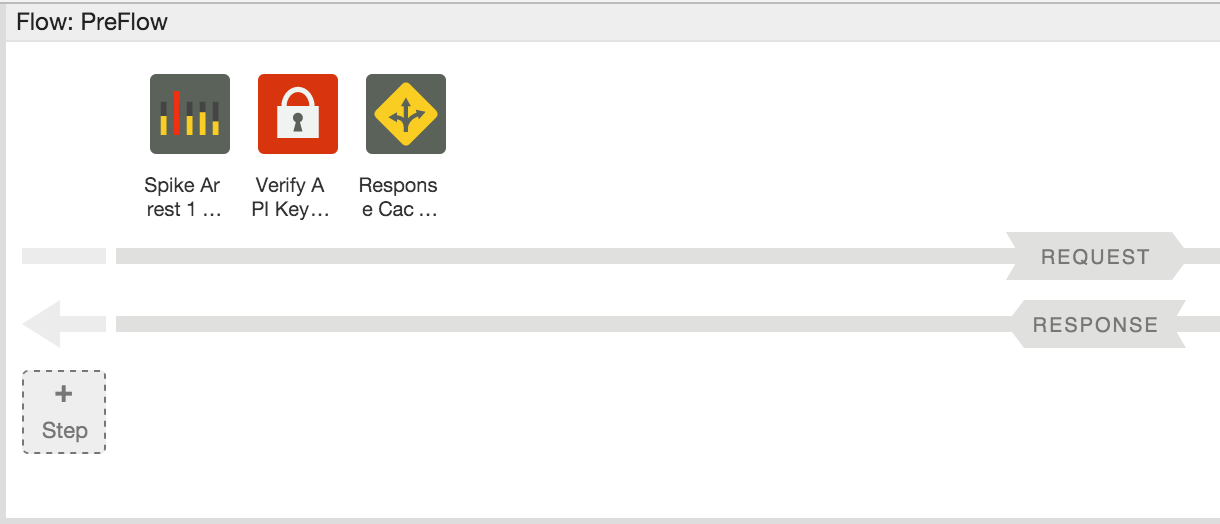


* 1. Select the ‘Verify API Key’ policy with the following properties:
     1. Policy Display Name: **Verify API Key**
     2. Policy Name: **Verify-API-Key**



**NOTE:** If you have not completed Lab 2, please skip this step.

The ‘Verify API Key’ policy will get added after the ‘Response Cache’ policy. **Drag and move** the ‘Verify API Key’ policy to be before the ‘Response Cache’ policy



**Note**: It depends on your use case, but typically API Key verification should be one of the first policies in the flow. In this scenario, we verify the API Key before the Response Cache policy to ensure that an API Consumer whose API Key may have been revoked is not able to get the data from the cache.

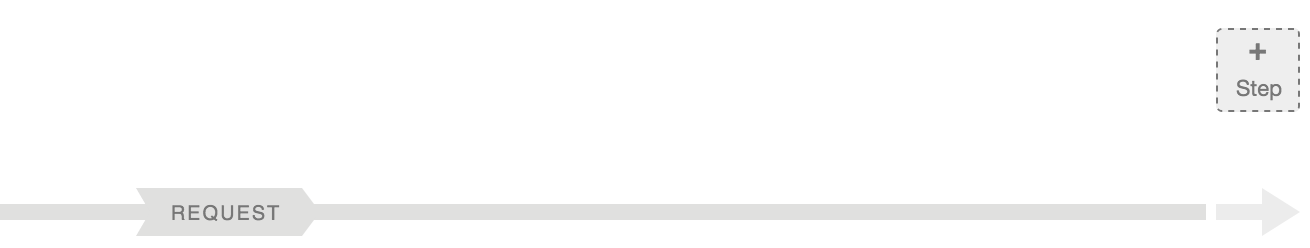
* 1. Examine the XML configuration in the ‘Code’ panel (or properties using the ‘Property Inspector’ panel) associated with the ‘Verify API Key’ policy. The XML for the policy should look something like this:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  <VerifyAPIKey async="false" continueOnError="false" enabled="true" name="Verify-API-Key">  <DisplayName>Verify API Key</DisplayName>  <FaultRules/>  <Properties/>  <APIKey ref="request.queryparam.apikey"/>  </VerifyAPIKey> |

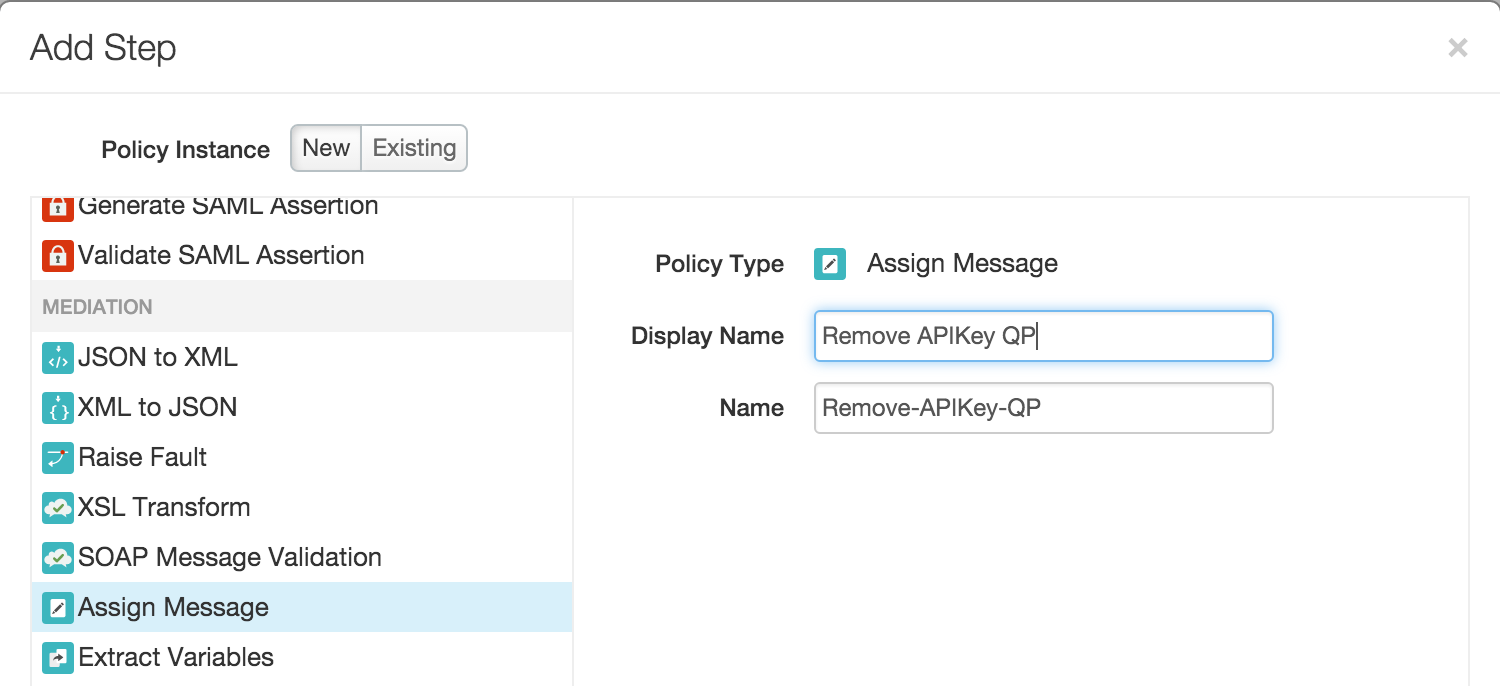
*(You can find the policy xml* [*here*](https://gist.github.com/prithpal/02ee175cc1e00a2de610)*. Click the “Raw” button and copy/paste into your policy editor).*

Note the <APIKey> element, which identifies where the policy should check for the API key. In this example, the policy looks for the API key in a query parameter named 'apikey'. API keys can be located in a query parameter, a form parameter, or an HTTP header. Apigee Edge provides a message variable for each type of location. For policy reference information, see [Verify API Key policy.](http://apigee.com/docs/api-services/reference/verify-api-key-policy)

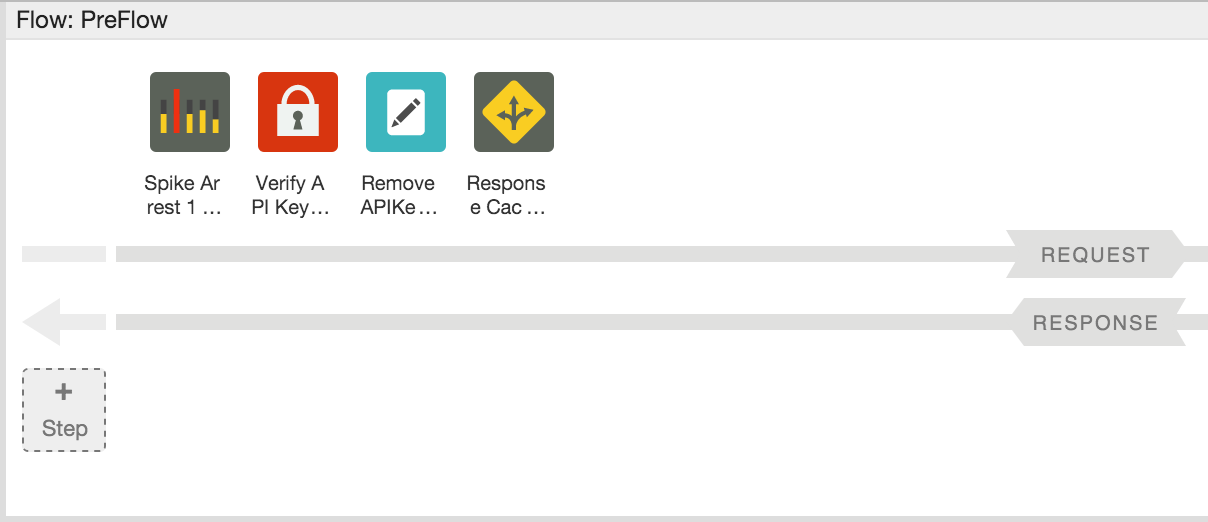
1. **Removing the API Key from the query parameters**
   1. Click on “+ Step” on the Request Flow



* 1. Select ‘Assign Message’ policy with the following properties:
     1. Policy Display Name: **Remove APIKey QP**
     2. Policy Name: **Remove-APIKey-QP**



* 1. The ‘Remove APIKey QP’ policy will get added after the ‘Response Cache’ policy. **Drag and move** the ‘Remove APIKey QP’ policy to be before the ‘Response Cache’ policy



* 1. **Save** the configuration
  2. For the ‘Remove APIKey QP’ policy, change the XML configuration of the policy using the ‘Code: Remove APIKey QP’ panel as follows:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  <AssignMessage async="false" continueOnError="false" enabled="true" name="Remove-APIKey-QP">  <DisplayName>Remove APIKey QP</DisplayName>  <Remove>  <QueryParams>  <QueryParam name="apikey"></QueryParam>  </QueryParams>  </Remove>  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>  <AssignTo createNew="false" transport="http" type="request"/>  </AssignMessage> |

(You can find the policy xml [here](https://gist.github.com/prithpal/cbf66ee17b9afe75fdf2). Click the “Raw” button and copy/paste into your policy editor).

As a security measure, the ‘Remove APIKey QP’ policy simply removes the ‘apikey’ query parameter from the HTTP request message attached to the flow so it is not sent to the backend service. In this scenario we are removing the ‘apikey’ immediately after verify API Key policy, but depending on your use case, removing the ‘apikey’ may need to be done at a later stage in the flow.

1. **Testing the API Key Verification Policy**

Until now anyone with the URL to the ‘{your\_initials}\_hotel’ API Proxy has been able to make a request with appropriate parameters and get a response back. Now that you have added the API Key Verification policy, that will no longer be the case.

* 1. Start the Trace session for the ‘{your\_initials}\_hotels’ proxy
  2. Now that the API Key Verification policy has been added to the proxy, try and send a test ‘/GET hotels’ request from Postman with the following query parameters: zipcode=98101&radius=200.

Note : Replace the URL of hotels API with **{your\_initials}**\_hotels

* 1. You will notice that the following fault is returned since an API Key has not been provided as a request query parameter:

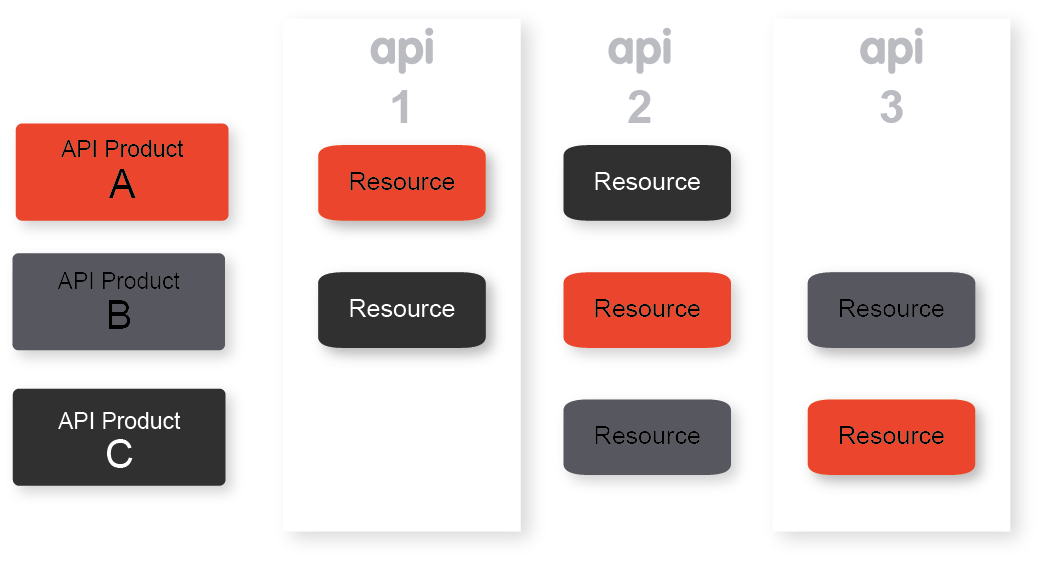
|  |
| --- |
| {  fault: {  faultstring: "Failed to resolve API Key variable request.queryparam.apikey",  detail: {  errorcode: "steps.oauth.v2.FailedToResolveAPIKey"  }  }  } |

The above response shows that the API Key Verification policy is being enforced as expected.

* 1. Review the Trace for the proxy and the returned response to ensure that the flow is working as expected.
  2. Stop the Trace session for the proxy

**API Products**

API products enable you to bundle and distribute your APIs to multiple developer groups simultaneously, without having to modify code. An API product consists of a list of API resources (URIs) combined with a Service Plan (rate-limiting policy settings) plus any custom metadata required by the API provider. API products provide the basis for access control in Apigee, since they provide control over the set of API resources that apps are allowed to consume.



As part of the app provisioning workflow, developer selects from a list of API products. This selection of an API product is usually made in the context of a developer portal. The developer app is provisioned with a key and secret (generated by and stored on Apigee Edge) that enable the app to access the URIs bundled in the selected API product. To access API resources bundled in an API product, the app must present the API key issued by Apigee Edge. Apigee Edge will resolve the key that is presented against an API product, and then check associated API resources and quota settings.

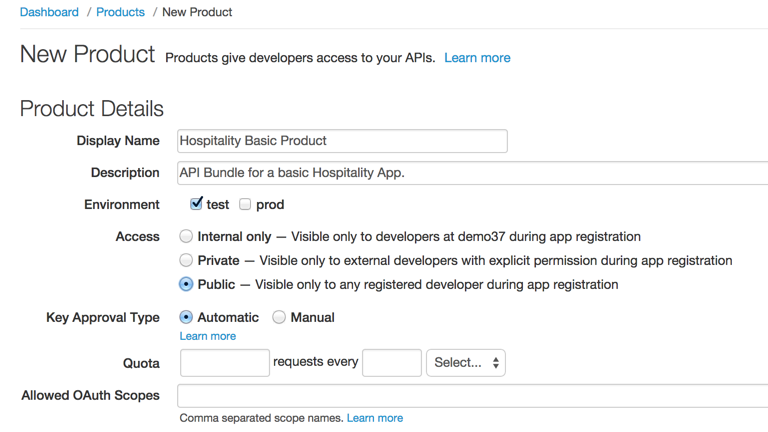
The API supports multiple API products per app key--your developers can consume multiple API products without requiring multiple keys. Also, a key can be 'promoted' from one API product to another. This enables you to promote developers from 'free' to 'premium' API products seamlessly and without user interruption.

The following table defines some of the terms used to register apps and generate keys:

|  |  |
| --- | --- |
| Term | Definition |
| **API product** | A bundle of API proxies combined with a service plan that sets limits on access to those APIs. API products are the central mechanism that Apigee Edge uses for authorization and access control to your APIs. For more, see [API Products](http://apigee.com/docs/developer-services/content/what-api-product) |
| **Developer** | The API consumer. Developers write apps the make requests to your APIs. For more, see Developer |
| **App** | A client-side app that a developer registers to access an API product. Registering the app with the API product generates the API key for accessing the APIs in that product. |
| **API key** | A string with authorization information that a client-side app uses to access the resources exposed by the API product. The API key is generated when a registered app is associated with an API product. |

**Publishing an API Product**

1. From the Apigee Edge Management UI, go to Publish → API Products
2. Click on ‘+ Product’ button to add a new product
3. In the ‘Product Details’ section of the new product screen, enter or select the following values for the various fields:
   1. Display Name: **{Your\_Initials}\_Hospitality Basic Product**
   2. Description: **API Bundle for a basic Hospitality App.**
   3. Environment: **Test**
   4. Access: **Public**
   5. Key Approval Type: **Automatic**



1. In the ‘Resources’ section select the following values for the various fields:
   1. API Proxy: **{your\_initial}\_hotels**
   2. Revision: **1**
   3. Resource Path: **/**

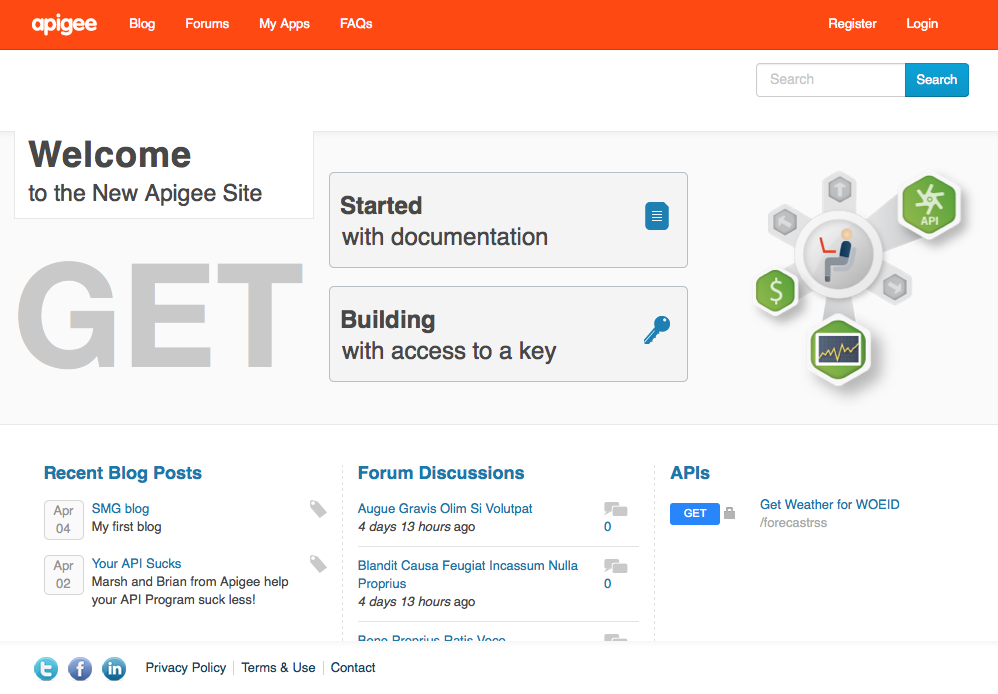


1. Click on **‘Import Resources’** to add the **‘/’** resource of your proxy to the API product
2. Repeat the above two steps for the **‘/\*\*’** resource
3. Click **‘Save’** to save the API Product**.** The new product should now be listed on the ‘Products’ page.

**Developer Portal:**

Developer portals with social publishing features are increasingly being used for communication with the developer community. This includes communicating static content, such as API documentation and terms-of-use, as well as dynamic community-contributed content such as blogs and forums.

As an API provider, you need a way to expose your APIs, educate developers about your APIs, sign up developers, and let developers register apps. Exposing your APIs to developers is only part of creating a truly dynamic community. You also need a way for your developer community to provide feedback, make support and feature requests, and submit their own content that can be accessed by other developers.



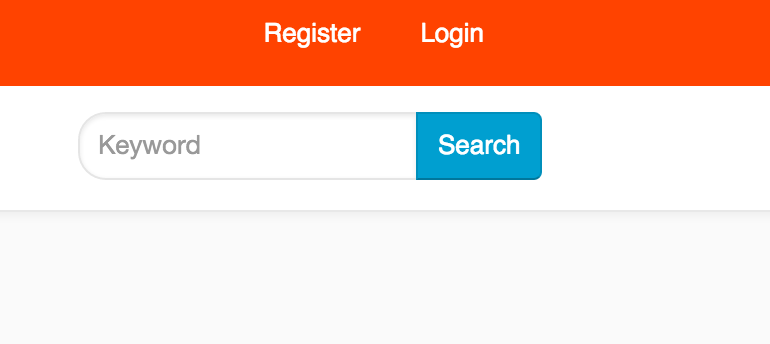
**Developers**

Developers access your APIs through apps. When the developer registers an app, they receive a single API key that allows them to access all of the API products associated with the app. However, developers must be registered before they can register an app.

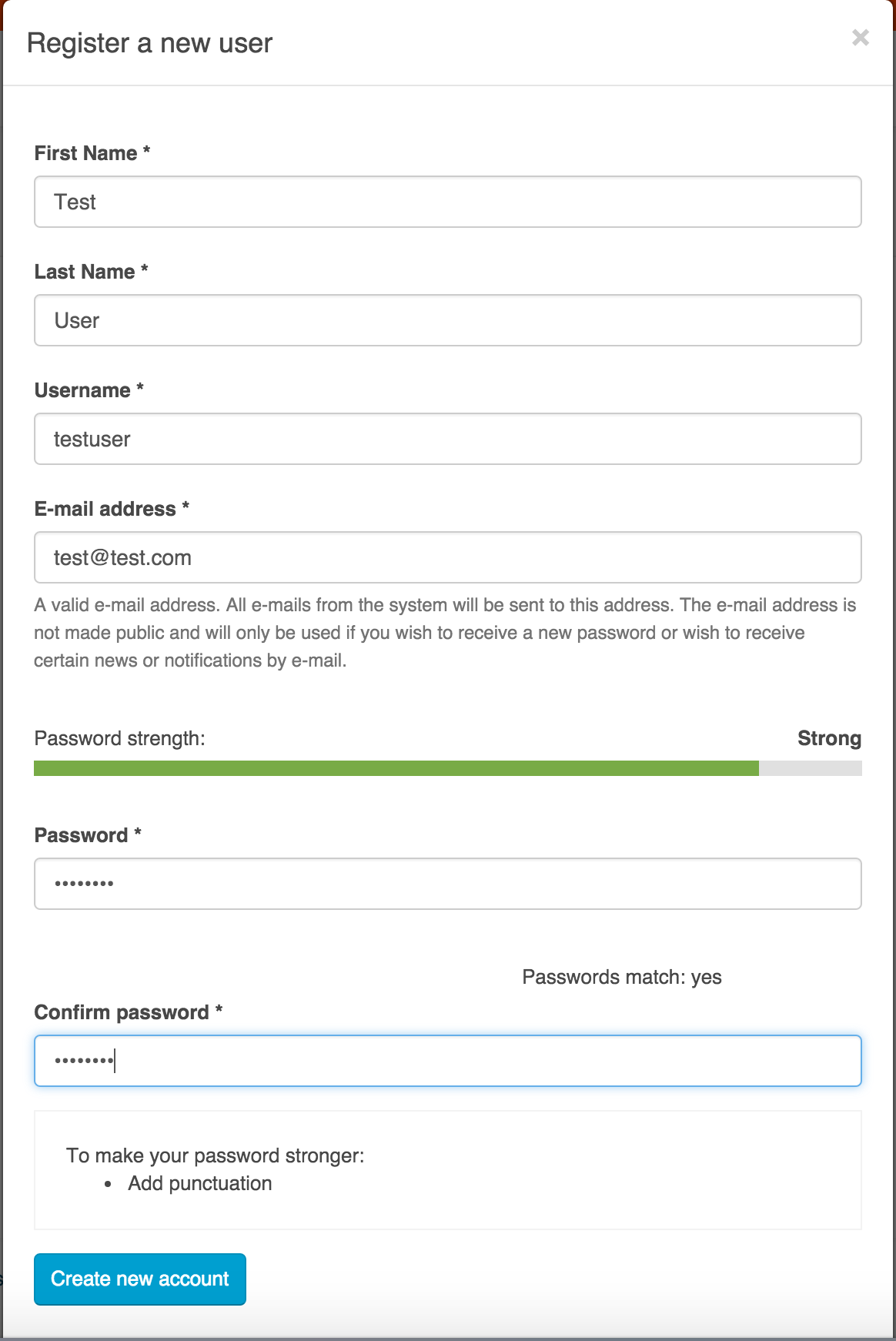
.

**Register a developer from the developer portal:**

1. The user selects **Register** on the developer portal home page

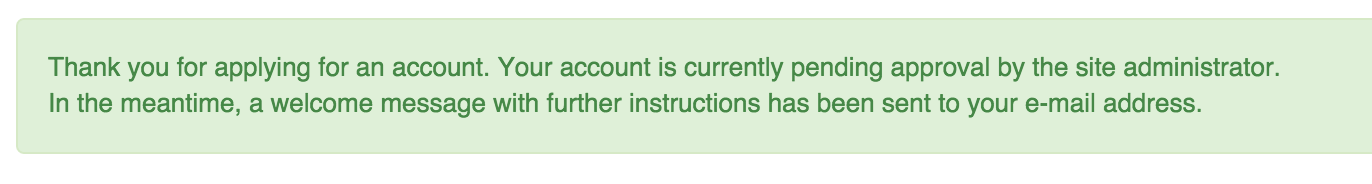


1. The registration page appears



1. The user enters the required information and selects **Create new account**.  
   Depending on the new account registration settings, when the new account is created, the user is sent an automated welcome email.

NOTE: If you see this message:

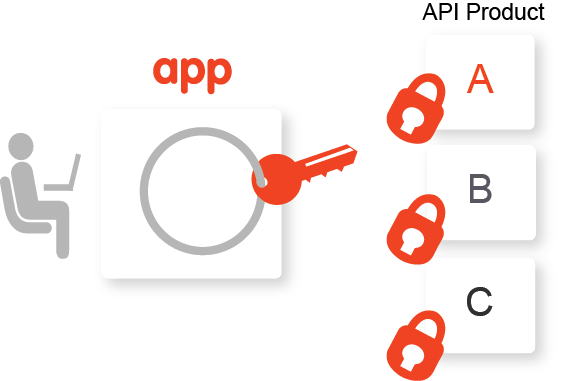


Then the developer portal administrator MUST approve the developer before the developer can sign in.

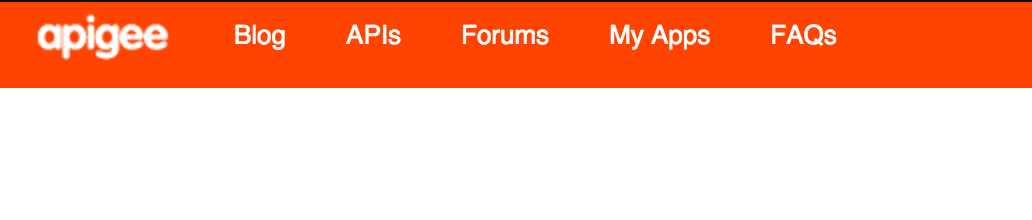
**Register an app from the developer portal:**

Developers register apps to access your API products. When a developer registers an app, the developer selects the API products to associate with the app and Edge generates an API key. Each app has a single API key that provides access to all API products associated with the app.

Apps allow you to control who can access your APIs. You can revoke an app's key, preventing it from accessing all API products. Or you can revoke access to a single API product associated with the app.

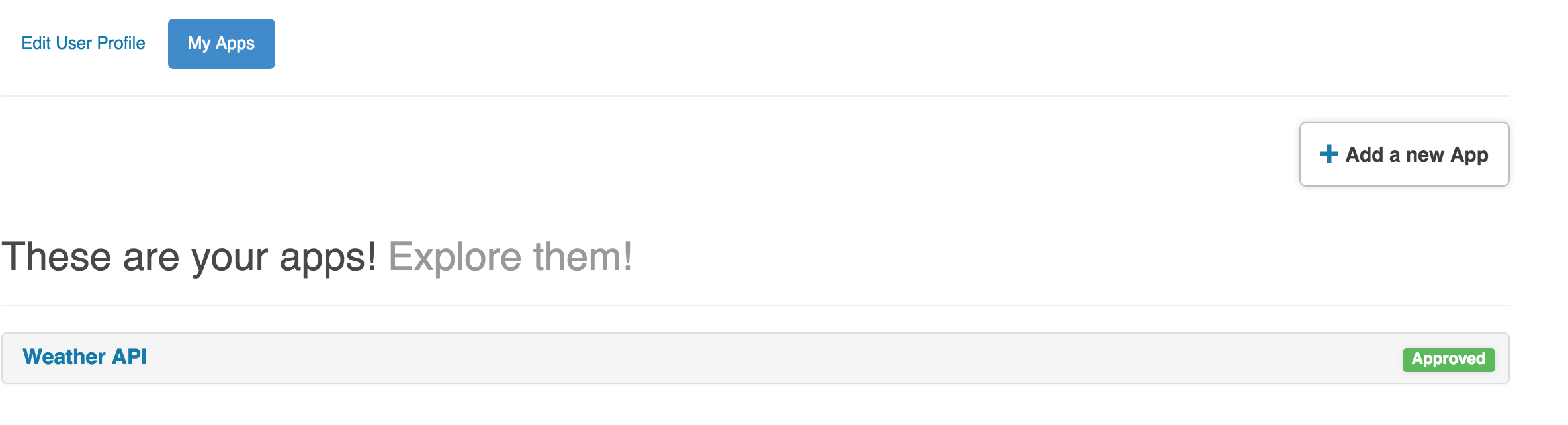


1. Select **My apps** below your username in the login menu

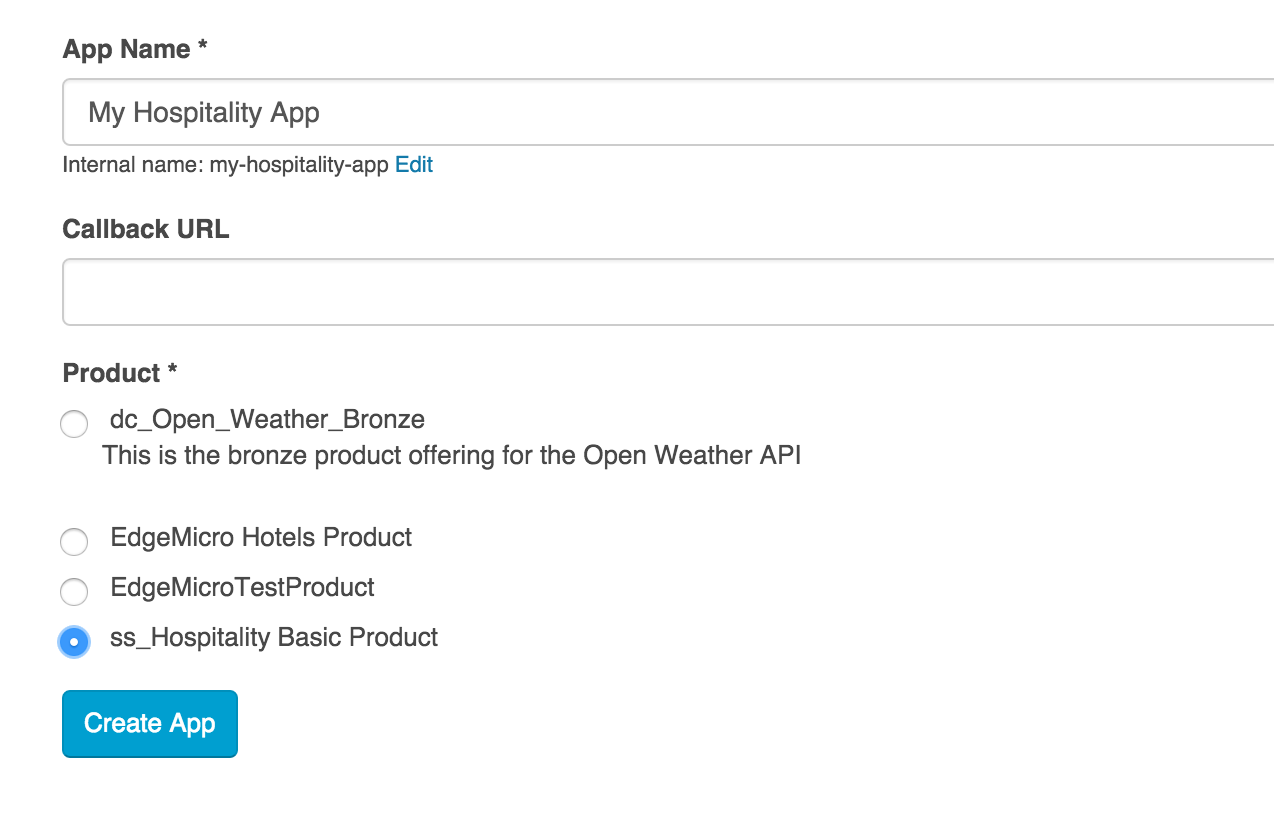


1. Click the **+ Add a** **new App** icon.

## 

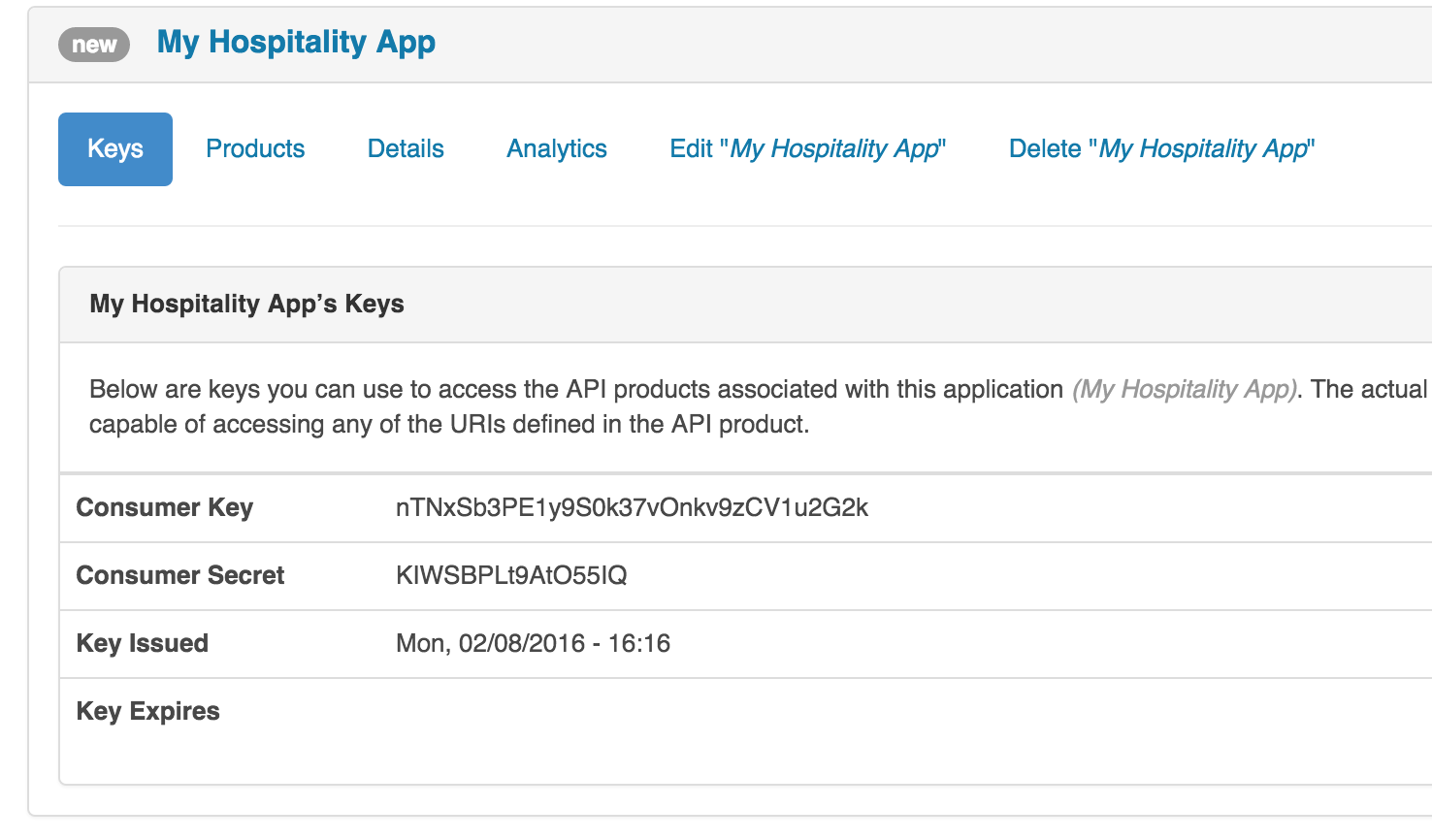


1. Enter details for the application



NOTE: Select the product that you created in the previous step.

1. View the Consumer Key (aka API Key) and Consumer Secret (aka API Secret)



1. Test the API
   1. Start the Trace session for the ‘**{your\_initials}\_hotels**’ proxy
   2. Now that the API Key Verification policy has been added to the proxy, try and send a test ‘/GET hotels’ request from Postman with the following query parameters: **zipcode=98101&radius=200&apikey={apikey from the dev portal}**

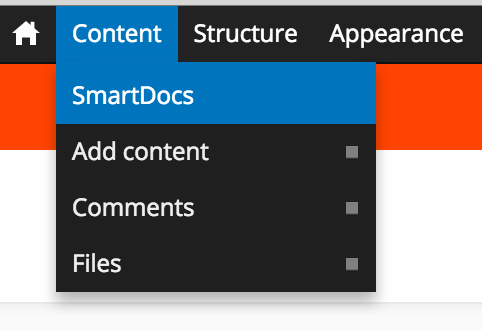
Note: Replace the URL of hotels API with **{your\_initials}**\_hotel

**Generate API Documentation**

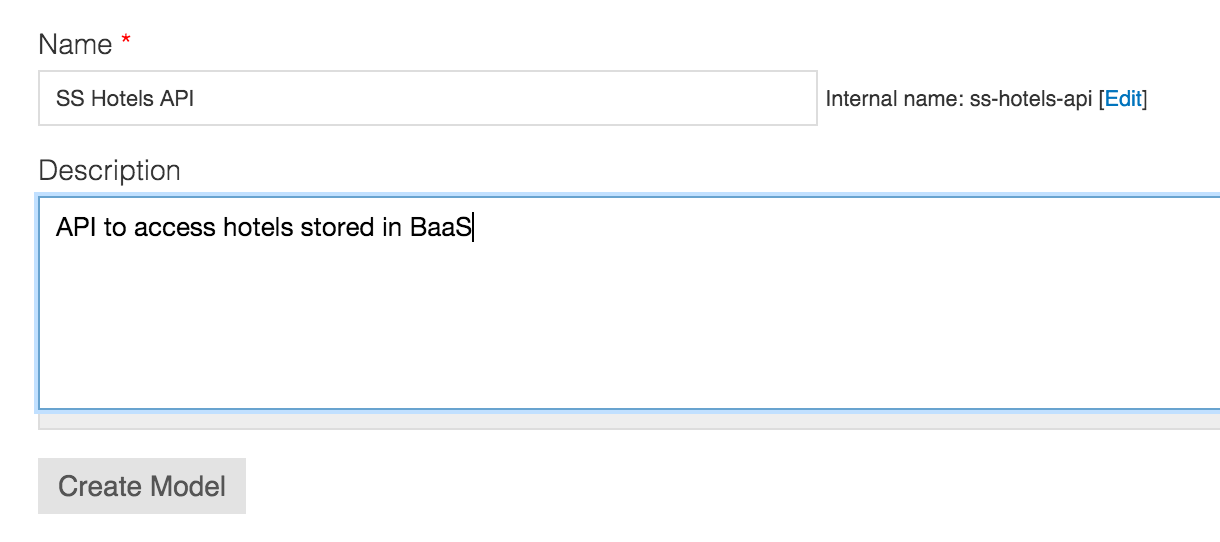
**1. Create a model**

When you create a model, it's stored in your Edge organization as the source for the API structure. For more information, see [About SmartDoc models and templates](http://apigee.com/docs/developer-services/content/using-smartdocs-document-apis#models).

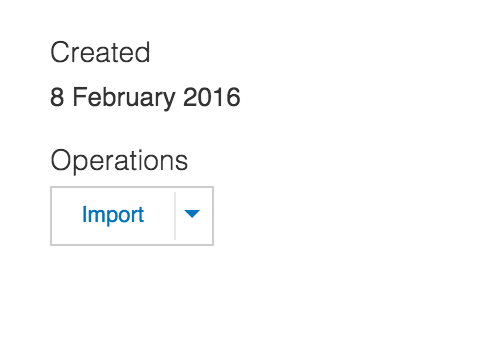
1. Log in to your portal as a user with admin or content creation privileges.
2. Select **Content > SmartDocs** in the Drupal administration menu.



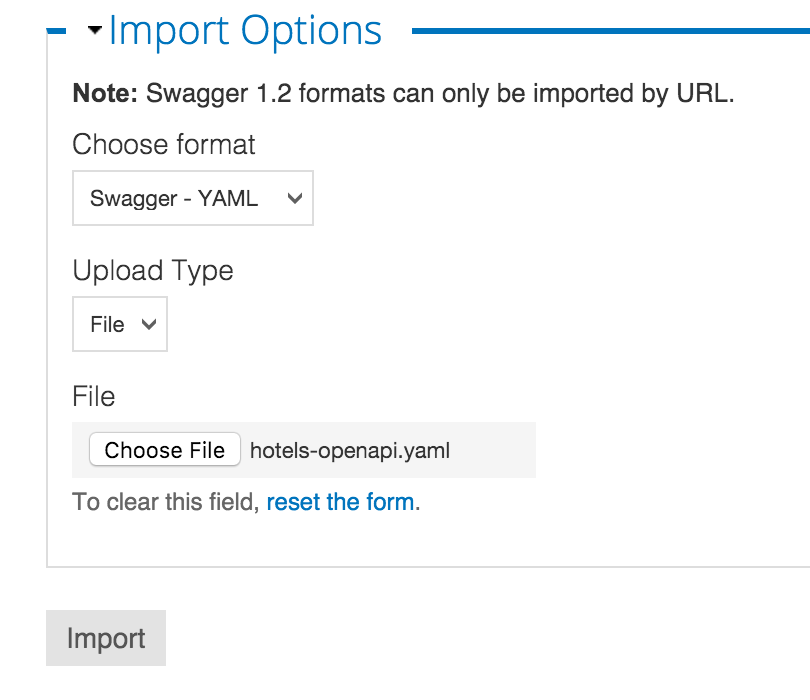
1. Select **New model** at the top of the page
2. Enter the following fields:
   * **Name**: The model name that will be displayed across the site.
   * **Internal name**: As you type the **Name**, the internal name displays. The internal name for the model that must be unique among all models.  The internal name must contain only lowercase letters, numbers, and hyphens with no spaces. Select**Edit** to edit this name.
   * **Description**: A description of the model.



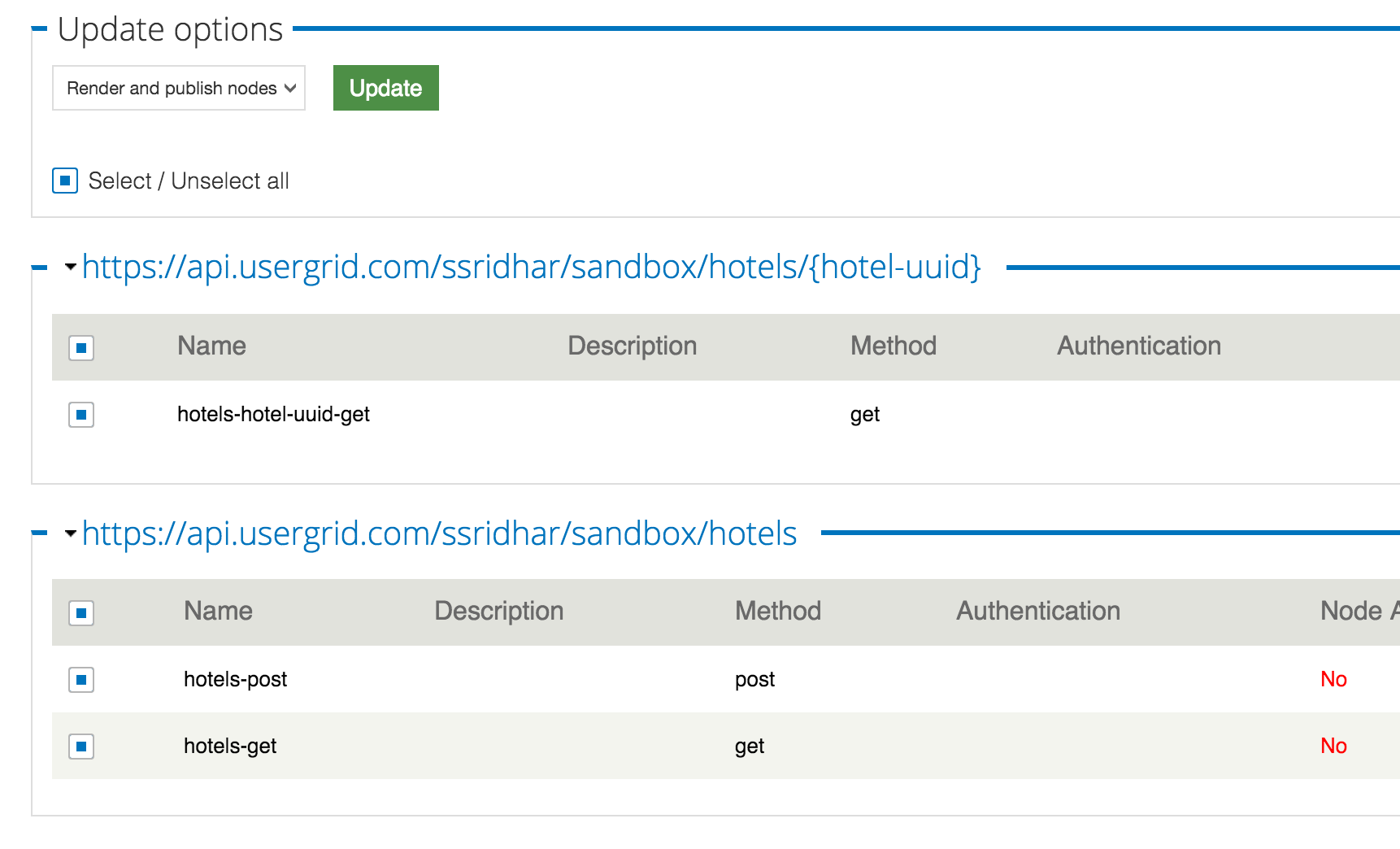
1. Select **Create Model**.
2. Click on ‘Import’ to import an API specification



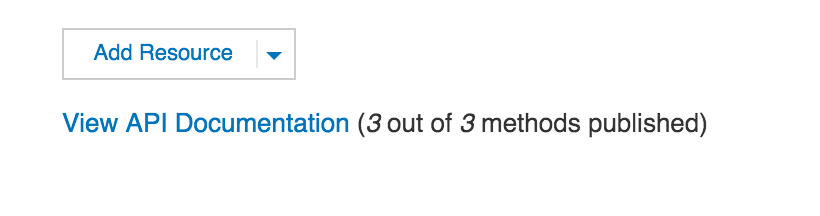
1. The API specification can be of Swagger 1.2, Swagger 2.0 (JSON or YAML) or WADL formats. In this example, we will use Swagger 2.0 – YAML. Select the **hotels-openapi.yaml** file from the lab material



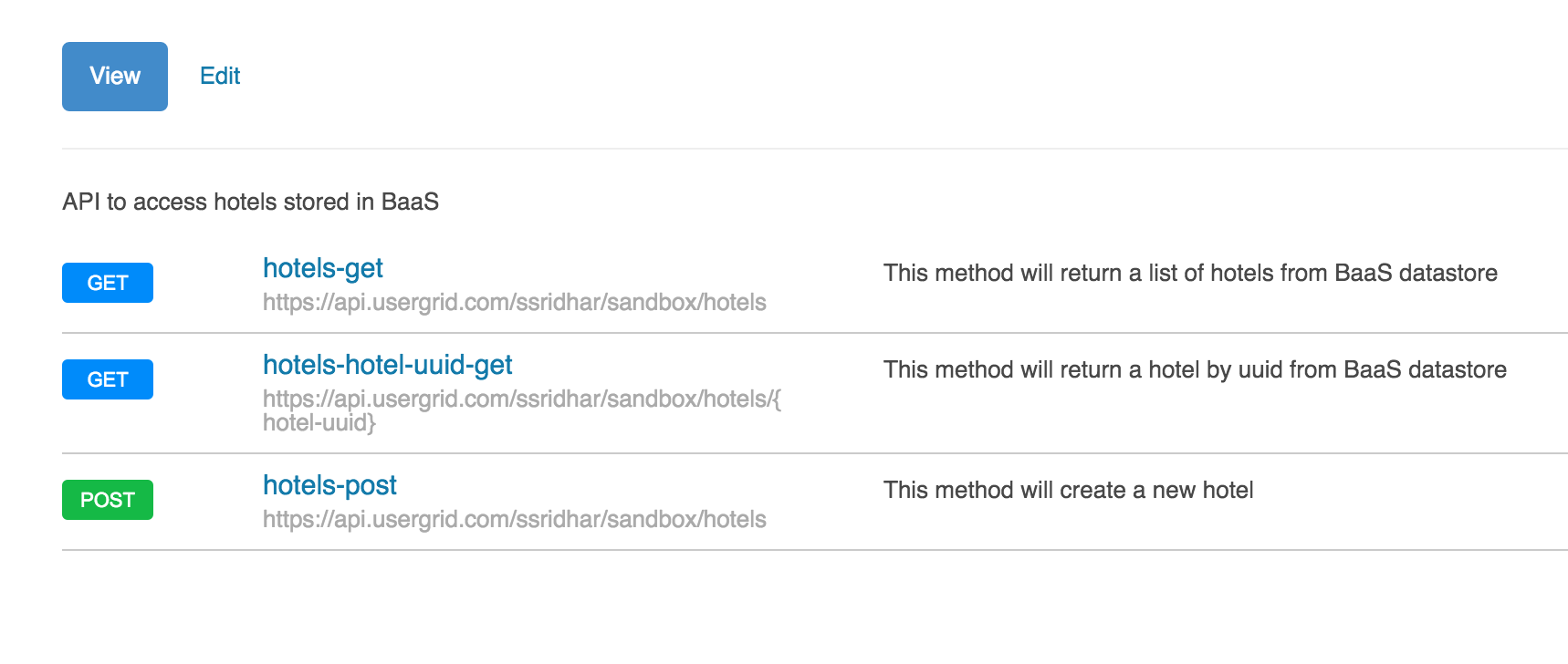
1. Click on ‘**Import’**
2. Select all the Operations//Methods that should be published (we will select all)



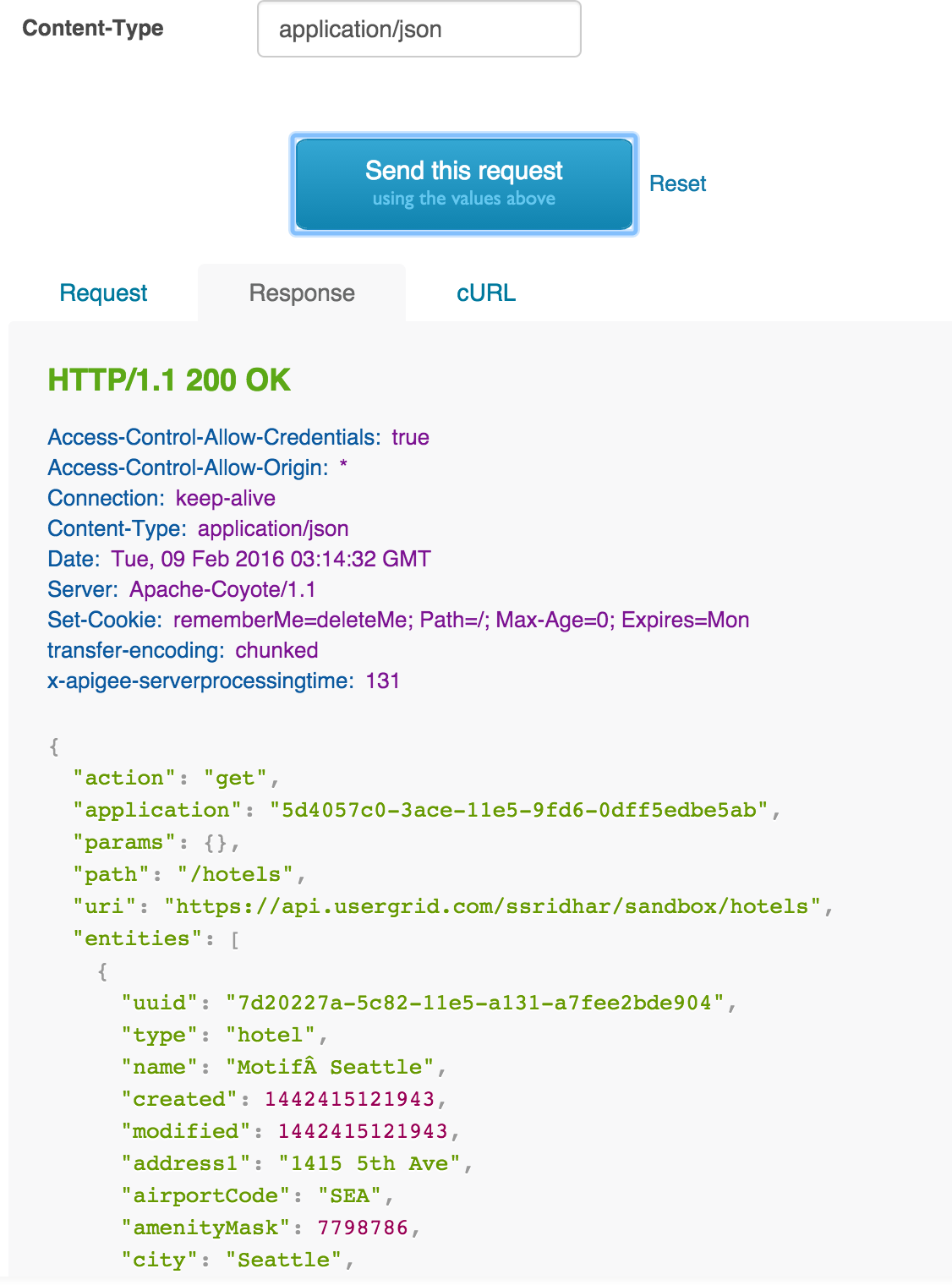
1. Click on ‘**Update**’
2. Click on ‘View API Documentation’ to see the published documentation



1. Click on ‘**hotels-get**’



1. Enter ‘**application/json**’ as the Content-Type. Leave the radius and zipcode empty
2. Click on ‘**Send this request**’



**Summary**

In this exercise, you learnt about how API keys can be used as an application identifier, how APIs can be packaged in the form of Products. You also saw how to publish API documentation to the developer portal and finally, how developers are onboarded and how developer applications are registered.