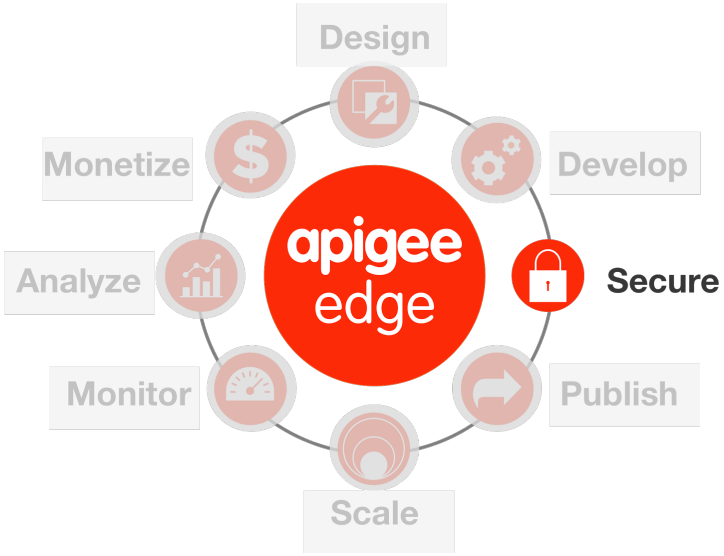
Lab 3 - Securing APIs



Overview
There are several out-of-the-box security policies that Apigee Edge provides to protect your APIs. These security policies must be used appropriately based upon your use cases. Apigee Edge supports:

API Keys
API key validation is the simplest form of app-based security that you can configure for an API. Apps simply present an API key, and Apigee Edge checks to see that the API key is in an approved state for the resource being requested. For this reason the security associated with API keys is limited. API keys can easily be extracted from app code and used to access an API. You may find that API keys work better as unique app identifiers than as security tokens. For more information, see API Keys - http://apigee.com/docs/api-services/content/api-keys

OAuth v2.0
Here's the definition of OAuth from the OAuth 2.0 IETF specification. "The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf."

All requests from the app for protected resources from the backend service is negotiated using an access token. Access tokens are long random strings generated by an authorization server after appropriate app and/or user credentials are presented. Tokens are used to validate requests for protected resources. If an app is compromised, the resource server can revoke it's access token. In that case, the end user does not need to change her username/password on the resource server, her app simply needs to renegotiate for a new access token.

To go beyond app identification using API Keys, Apigee Edge provides comprehensive support for OAuth v2.0. Apigee Edge includes an OAuth 2.0 authorization server implementation that lets you register apps and secure API proxies with OAuth 2.0 using the four grant types that are part of OAuth 2.0. For more information, see OAuth 2.0 (http://apigee.com/docs/api-services/content/oauth-introduction) and the OAuth the Big Picture Apigee eBook - https://pages.apigee.com/oauth-big-picture-ebook.html

OAuth v1.0a
OAuth 1.0a defines a standard protocol that enables app users to authorize apps to consume APIs on their behalf, without requiring app users to disclose their passwords to the app in the process. Apigee Edge enables you to protect APIs in a way that ensures that an app uses has authorized the app to consume an API. Edge also provides policy-based functionality for configuring the endpoints that app developers can use to obtain access tokens. For more information, see OAuth v1.0a policy - http://apigee.com/docs/api-services/reference/oauth-10-policy

SAML
Apigee Edge enables you to authenticate and authorize apps that are capable of presenting SAML tokens. A SAML token is a digitally signed fragment of XML that presents a set of "assertions". These assertions can be used to enforce authentication and authorization.

To use SAML terminology, Apigee Edge can function as a service provider (SP) or an Identity Provider (IP). When Apigee Edge validates SAML tokens on inbound requests from apps, it acts in the role of SP. (API Services can also act in the IP role, when generating SAML tokens to be used when communicating with backend services. For details on SAML validation, see SAML Assertion policies - http://apigee.com/docs/api-services/reference/saml-assertion-policy

Content Based Security
Message content is a significant attack vector used by malicious API consumers. API Services provides a set of Policy types to mitigate the potential for your backend services to be compromised by attackers or by malformed request payloads.

JSON threat protection
JSON attacks attempt to use structures that overwhelm JSON parsers to crash a service and induce application-level denial-of-service attacks.

Such attacks can be mitigated using the JSONThreatProtection Policy type.

See JSON Threat Protection policy - http://apigee.com/docs/api-services/reference/json-threat-protection-policy

XML threat protection
XML attacks attempt to use structures that overwhelm XML parsers to crash a service and induce application-level denial-of-service attacks.

Such attacks can be mitigated using the XMLThreatProtection Policy type.

See XML Threat Protection policy - http://apigee.com/docs/api-services/reference/xml-threat-protection-policy

General content protection
Some content-based attacks use specific constructs in HTTP headers, query parameters, or payload content to attempt to execute code. An example is SQL-injection attacks. Such attacks can be mitigated using the Regular Expression Protection Policy type.

See Regular Expression Protection policy - http://apigee.com/docs/api-services/reference/regular-expression-protection

Data Masking
Apigee Edge enables developers to capture message content to enable runtime debugging of APIs calls. In many cases, API traffic contains sensitive data, such credit cards or personally identifiable health information (PHI) that needs to filtered out of the captured message content.

To meet this requirement, Edge defines 'mask configurations' that enable you to specify data that will be filtered out of trace sessions. Masking configurations can be set globally (at the organization-level) or locally (at the API proxy level). Role-based capabilities govern which users have access to the data that is defined as sensitive.

For more information, see Data Masking - http://apigee.com/docs/api-services/content/data-masking

Objectives
The objective of this lesson is to get you familiar with Apigee's API Key verification policy.

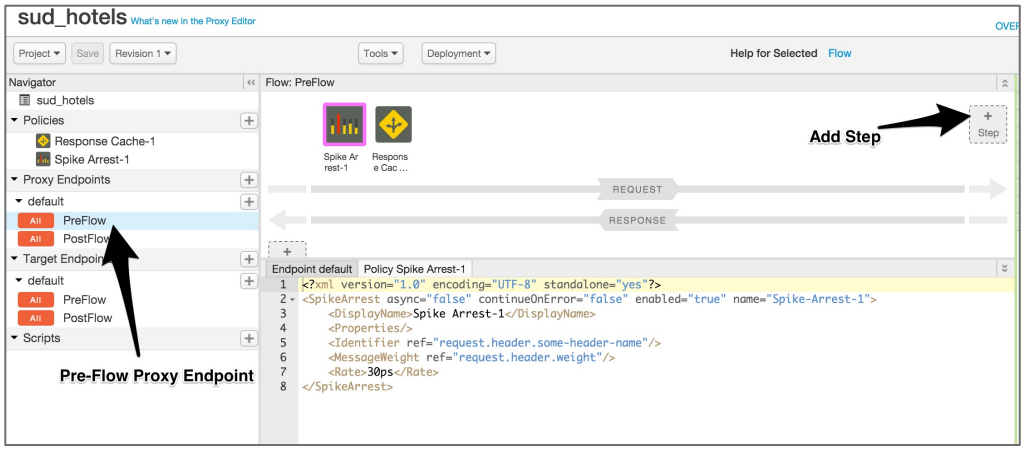
As discussed in the Overview section, API Key verification policy enables you to identify the API consumer and prevents unidentified API consumers from being able to call your APIs. You will implement an API Key Verification policy in this lesson to get familiar with several associated concepts such as: API Key, API Secret, API Products and Developer Apps.

Prerequisites
- Lab 2 is completed.

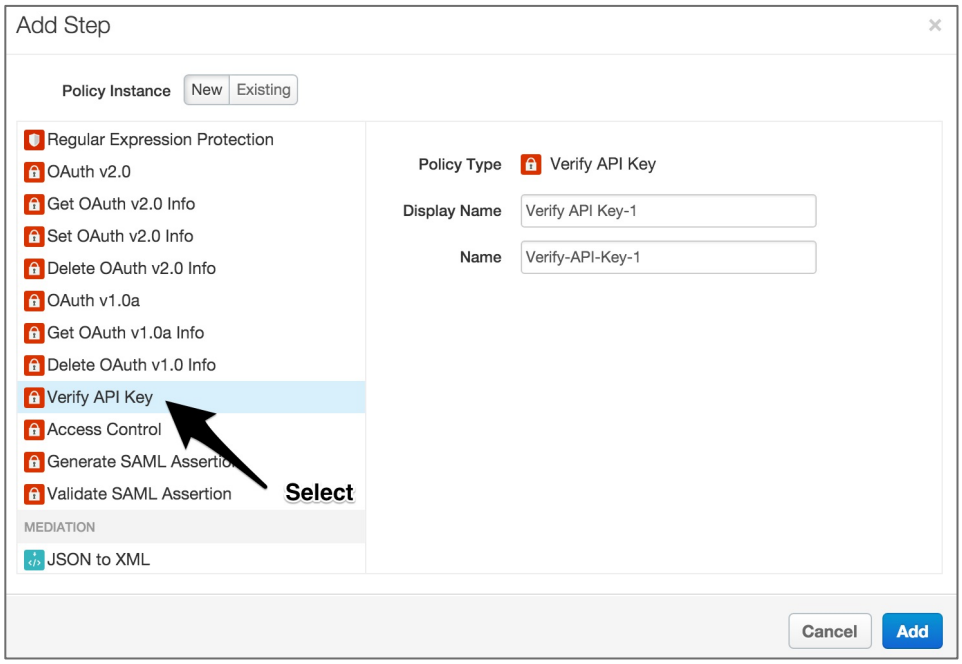
Estimated Time: 20 mins

1) **Adding an API Key Verification Policy**

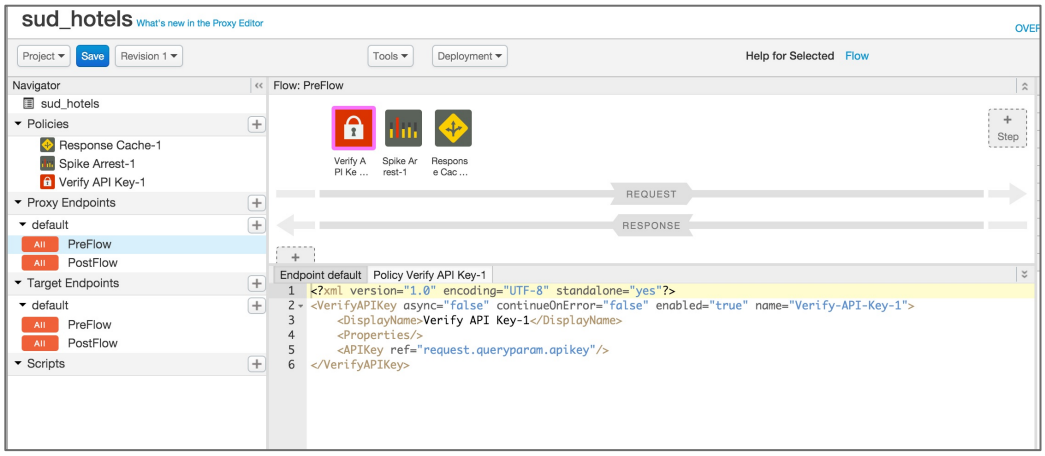
   a) Go to the Apigee Edge Management UI browser tab.
   b) Select your API proxy.
   c) Switch to "Develop" tab.
   d) Go to Preflow under "Proxy EndPoints" section and Click "**+ Step**" on the Request pipeline.



e) Select the “Verify API Key” from the list of policies and click on Add.



a) This will add the “Verify API Key” policy to Proxy PreFlow section on Request pipeline. Now drag the newly added “Verify API Key” policy to the front and place it before the “Spike Arrest” policy. Your policy configuration should look like this –



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 Spike Arrest and 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.

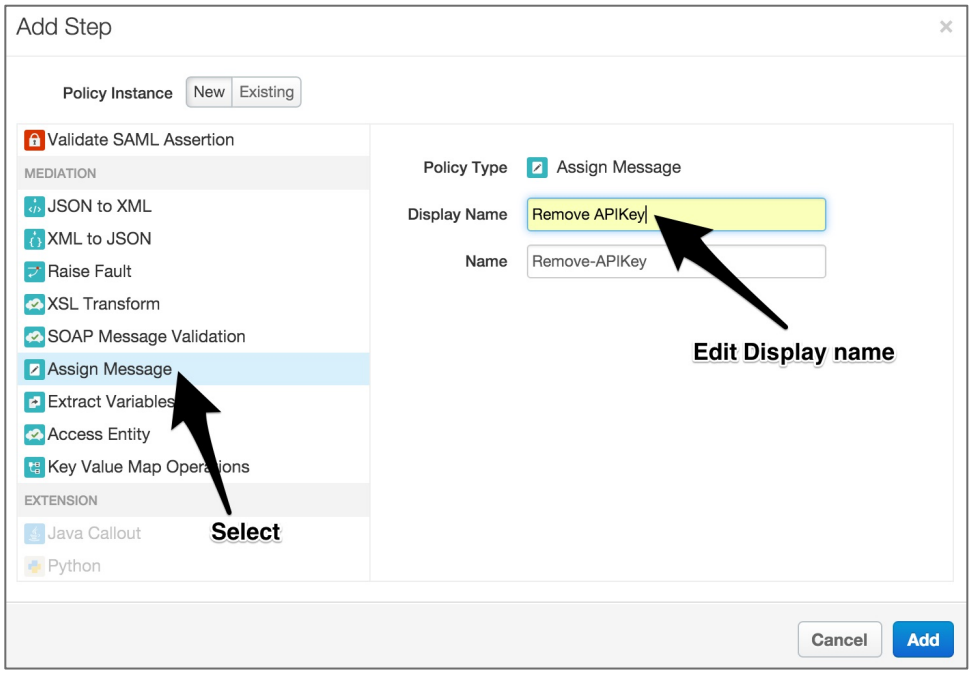
f) Examine the XML configuration in the 'Code' panel (or properties using the 'Property Inspector' panel) associated with the 'Verify API Key' policy. The XML for the policy should look something like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<VerifyAPIKey async="false" continueOnError="false" enabled="true"
name="Verify-API-Key-1">
   <DisplayName>Verify API Key-1</DisplayName>
   <Properties/>
   <APIKey ref="request.queryparam.apikey"/>
</VerifyAPIKey>
```

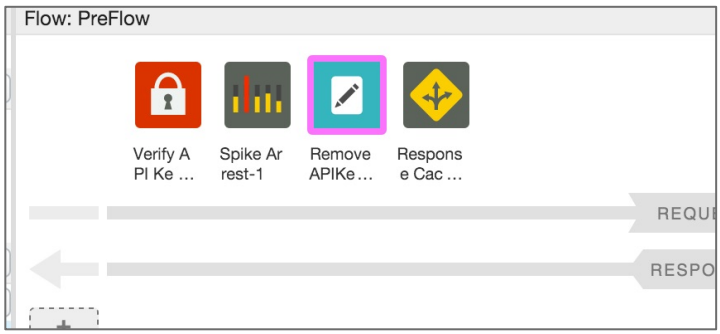
Note the <APIKey> element, which identifies where the policy should check for the API key. In this example, the policy looks for the API key in a query parameter named 'apikey'. API keys can be located in a query parameter, a form parameter, or an HTTP header. Apigee Edge provides a message variable 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

2) Removing the API Key from the query parameters

a) Go to Preflow under "Proxy EndPoints" section and Click "**+ Step**" on the Request pipeline.
b) Select the "Assign Message" from the list of policies.
c) Edit Display name to "Remove APIKey" and click Add.



d) The “Remove APIKey” policy will get added after the “Response Cache” policy. **Drag and move** the “Remove APIKey” policy before the “Response Cache” policy. Your policy configuration should look like this -



e) Select the “Remove APIKey” policy, change the XML configuration of the policy as follows:

As a security measure, the “Remove APIKey” 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’ right after the 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.

Alternatively you can copy the XML from here -

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

3) 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.

a) Switch to Trace tab of your API proxy and make a call. Alterntively call your API from a brower using the following URL.

```
http://{your-edge-org}-test.apigee.net/v1/{your_initials}_hotels
```

Replace **{your-edge-org}** with the actual name of your Edge organization. Replace **{your_initials}** with your initials. This should match with your API proxy name.

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

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

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

b) Review the Trace for the proxy and the returned response to ensure that the flow is working as expected.
c) Stop the Trace session for the proxy.

4) Obtaining an API Key

Up to now you have been playing the role of an *API Developer* configuring various policies in your proxy.

To understand the concept of an API Key and associated concepts such as *API Products, Developer* and *Developer Apps*, you need to now think about the role of an App Developer. This Developer needs an API Key that can be used whenever their app needs to use the proxy.

To make it easier for Developers to consume APIs, Apigee Edge provides the capability of publishing APIs. *Publishing* is the process of making your APIs available to Developers for consumption. Publishing APIs can be broadly defined by the following tasks:

- Create the *API Products* on Apigee Edge that bundle your APIs.

- Register *Developers* on Edge. Only a registered App Developer can register an App.
- *Developers* register *Developer Apps* on Edge to access API products. In response, the developer receives an API key. Now that the developer has an API key, they can make requests to your APIs.

For more, see Publishing Overview - http://apigee.com/docs/developer-services/content/publishing-overview

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.

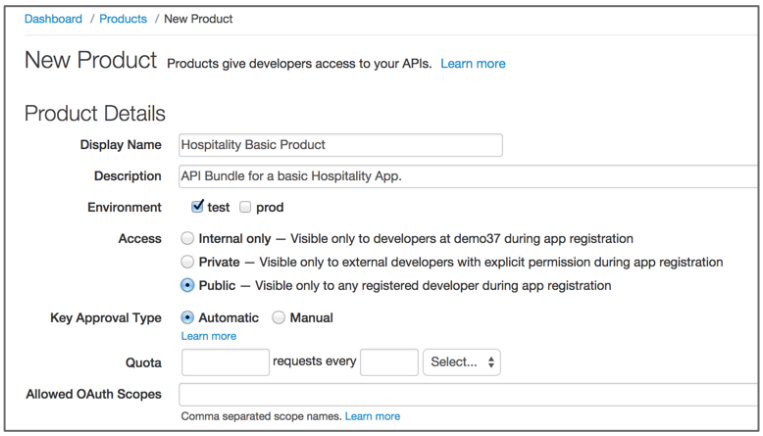
With the above brief introduction to API Products, Developers and Developer Apps, you will now create one of each to obtain a valid API Key that can be used to call your proxy

a) Publishing an API Product

i. From the Apigee Edge Management UI, go to Publish → API Products
ii. Click on **'+ Product'** button to add a new product
iii. In the 'Product Details' section of the new product screen, enter or select the following values for the various fields:

- Display Name: **{your_initials}_Hospitality Basic Product**
- Description: **API Bundle for a basic Hospitality App.**
- Environment: **Test**
- Access: **Public**

- Key Approval Type: **Automatic**



iv. In the 'Resources' section select the following values for the various fields:

- API Proxy: **{your_initial}_hotels**
- Revision: **1**
- Resource Path: **/**

v. Click on 'Import Resources' to add the '/' resource of your proxy to the API product

vi. **Repeat** the above two steps for the '/**' resource

vii. Click 'Save' to save the API Product. The new product should now be listed on the 'Products' page.

b) Registering a Developer

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.

Developers typically have several ways of registering:

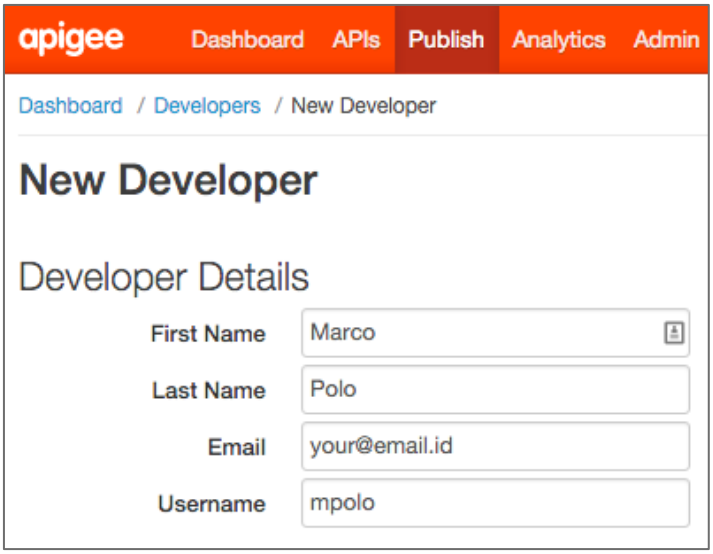
- If you have a paid Edge account, through a Developer Services portal. See Add and manage user accounts - http://apigee.com/docs/developer-services/content/add-and-manage-user-accounts for more.
- By accessing a form that uses the Edge management API to register the developer. See Using the Edge management API to Publish APIs (http://apigee.com/docs/developers-services/content/using-edge-management-api-publish-apis) for more.

- By a back-end administrator using the Edge management UI.

You will be learning more about how Developers can go through a self-service registration process using Developer Services Portal in later lessons. For the continuity of this lesson, the following steps describe the process of registering Developers and Developer Apps using the Apigee Edge Management UI.

i. From the Apigee Edge Management UI, go to Publish → Developers
ii. Click on **'+ Developer'** button to add a new product
iii. Add a new developer with the following properties:

- First Name: **Your First Name**
- Last Name: **Your Last Name**
- Email: **{your email_id}**
- Username: **{firstname_lastname}**



iv. Click 'Save' to save the Developer. The new developer should now be listed on the 'Developer' page.

c) Registering a Developer App

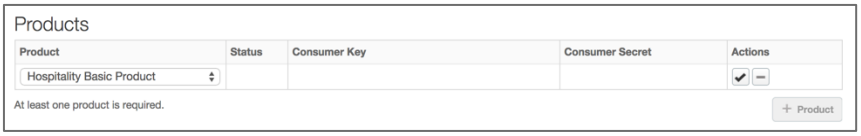
Now that you have an API product and a developer, you can register a Developer App with the API product. Registering the Developer App generates the API key for the API products associated with the app. You can then distribute the key to app developers so they can access the features in the API products from the app.

As mentioned earlier, you will learn about self-registering apps as a developer using Developer Services Portal in later lessons. For the continuity of this lesson, the following steps describe the process of registering Developer Apps using the Apigee Edge Management UI.

i. From the Apigee Edge Management UI, go to Publish → Developer Apps
ii. Click on **'+ Developer App'** button to add a new product
iii. In the 'Developer App Details' section, enter or select the following values for the various fields:

- Display Name: **{Your_Initials}_iExplore App**
- Developer: **{Your_name}**
- Callback URL: **Leave it blank**

iv. In the 'Products' section, click on the '+ Product' button
v. From the 'Product' drop-down, select the product you created.
vi. Click the 'check-mark' button in the 'Actions' column to accept the changes



vii. Click 'Save' to save the Developer App. The new app should now be listed on the 'Developer Apps' page
viii. From the 'Developer Apps' page, select your App.
ix. In the 'Products' section, next to the entry for **'{your_initials}**_Hospitality Basic Product,' click 'Show' in the 'Consumer Key' and 'Consumer Secret' columns to display the generated keys

Note: Since you selected 'Key Approval Type: Automatic' when you created the API product, the API key is automatically approved and you can view it immediately

If you had selected 'Approval Type: Manual,' you would need to click 'Approve' in the 'Actions' column to approve the API key.

The way your proxy is configured, as of now, the Consumer Key (i.e. the API Key) is the only key that the your App will need to access the proxy resources. You will use the Consumer Secret (i.e. the API Secret) in the next section when the security policy is changed from API Key Verification to an OAuth Token Validation policy.

5) Re-testing the API Key Verification Policy

Now that you have a valid API Key associated with a Developer App with access to the API Product that bundles your API proxy, test the API Key Verification policy to ensure that everything works as expected.

a) Copy the 'Consumer Key' associated with the app by going to Publish → Developer Apps → {Your_Initials}_iExplore App.

b) Start the Trace session for the **'{your_initials}**_hotels' proxy.
c) Send a test request with appending the "apikey" query parameter :

```
apikey={Your_Initials}_iExplore App Consumer Key
```

Alterntively call your API from a brower using the following URL.

```
http://{your-edge-org}-test.apigee.net/v1/{your_initials}_hotels?
apikey={Your_Initials}_iExplore App Consumer Key
```

Replace **{your-edge-org}** with the actual name of your Edge organization. Replace **{your_initials}** with your initials. This should match with your API proxy name. Also replace the **{Your_Initials}_**iExplore App Consumer Key with the actual Consumer Key. As you copy-paste, be sure to remove any spaces before and after the Consumer Key

d) Review the Trace for the proxy and the returned response to ensure that the flow is working as expected.
e) Stop the Trace session for the '{**your_initials**}_hotels' proxy

Summary

That completes this hands-on lesson. In this lesson you learned about the various out of the box security related policies that are available in Apigee Edge and to leverage the API Key verification policy to secure your APIs.