**Oracle Cloud Test Drive for PaaS and IaaS**

Lab 1.0

API Platform Cloud Service API Manager Lab

Contents

[2 Lab Scenario 3](#_Toc508123898)

[2.1 Logging into the API Management Portal 3](#_Toc508123899)

[2.2 The APIPCS Management Portal Environment 4](#_Toc508123900)

[2.2.1 APIs 4](#_Toc508123901)

[2.2.2 Applications 4](#_Toc508123902)

[2.2.3 Gateways 4](#_Toc508123903)

[2.2.4 Oracle APIP Cloud Service Roles 4](#_Toc508123904)

[3 Implement an API 6](#_Toc508123905)

[3.1 Create an API 6](#_Toc508123906)

[3.2 Configure Endpoints 7](#_Toc508123907)

[3.2.1 Configure the API Request 7](#_Toc508123908)

[3.2.2 Configure the Service Request 9](#_Toc508123909)

[3.2.3 About the Response Flow (informational--no activity in this section) 11](#_Toc508123910)

[3.3 Deploy an API 11](#_Toc508123911)

[3.3.1 Deploy an API to the Gateway 11](#_Toc508123912)

[3.3.2 Invoke the API & Validate your API Deployment 13](#_Toc508123913)

[4 Manage API Policies 14](#_Toc508123914)

[4.1.1 APIPCS Policies 14](#_Toc508123915)

[4.1.2 Add a Key Validation Policy 14](#_Toc508123916)

[4.1.3 Add an Application Rate Limiting Policy 16](#_Toc508123917)

[4.1.4 Add a Header Validation Policy 16](#_Toc508123918)

[4.1.5 Add an Interface Filtering Policy 17](#_Toc508123919)

[4.1.6 Add a Resource-Based Routing Policy 17](#_Toc508123920)

[4.1.7 Add a Service Callout Policy 18](#_Toc508123921)

[4.1.8 Add a Groovy Script Policy 19](#_Toc508123922)

[4.1.9 Re-Deploying an API 19](#_Toc508123923)

[4.2 Testing and Validating an API and its Policies using Postman 20](#_Toc508123924)

[4.2.1 Creating an Application 20](#_Toc508123925)

[4.2.2 Registering and Application 20](#_Toc508123926)

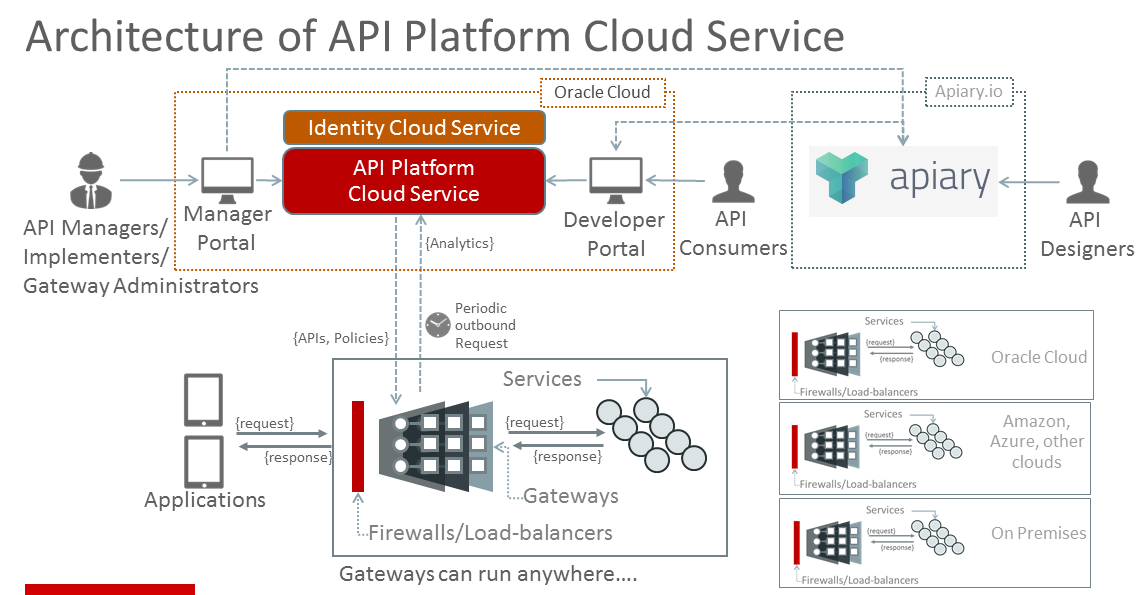
[4.2.3 Validate your API’s Policies 21](#_Toc508123927)

Introduction to the Lab and Logging-in

# Lab Scenario

The objective of this lab is to give you a comprehensive overview of the Oracle API Platform Cloud Service as an API Manager.

Oracle API Platform Cloud Service (APIPCS) makes it easy to manage, secure, and publish services for application developers. The Oracle APIPCS provides two separate portals for the different personas involved in application development & API management.

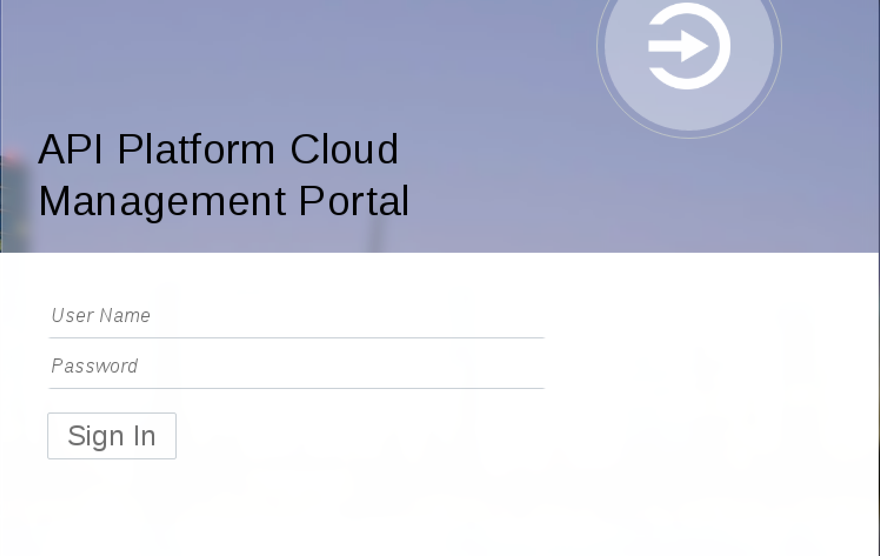


**Management Portal**: This portal is used by API Managers, the persons responsible for managing the API lifecycle, which includes designing, implementing, versioning APIs, managing grants and applications, providing API documentation, monitoring API performance and publishing APIs to be discovered by Application Developers (API Consumers).

**Developer Portal**: This portal is used by the Application Developers (API Consumers) to discover, view API documentation and register their interest in using APIs for a specific application.

In this scenario, you work for an energy company and will act as an API Manager and an Application Developer. Your organization has backend services that return billing and usage information for the most recent billing period when provided with a customer ID number. You have another service that estimates billing and usage information when provided with a customer ID number. You want to expose this information as a ‘single’ API to your company’s mobile developers, so that they can develop a new mobile application for your customers.

## Logging into the API Management Portal

Your instructor will assign you a URL and a username with a password to login. Please view the relevant “Access Credentials for all Labs” document.

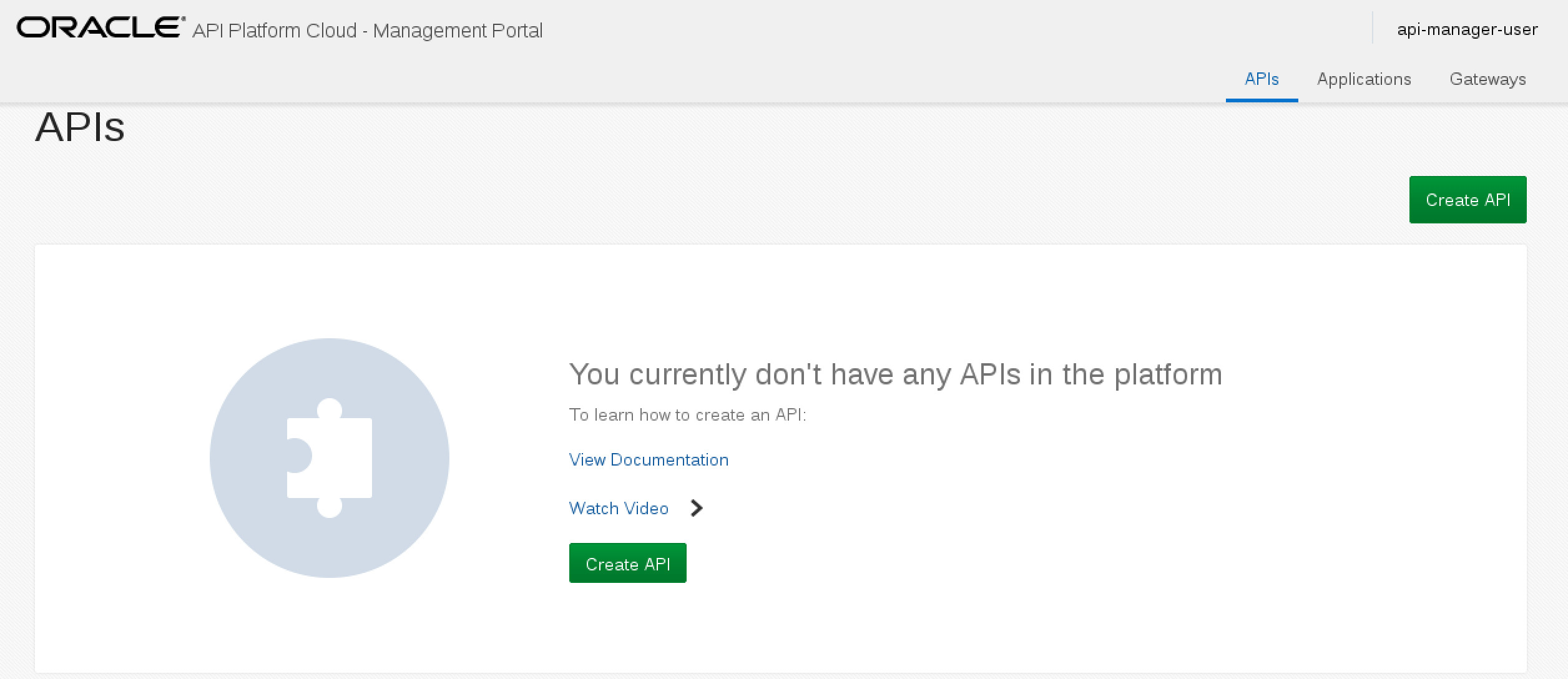
Your instructor will also assign a Prefix – you will not need it for now, only for later.

To avoid any possible cache issues:

* In Chrome - Select New Incognito Window
* In Mozilla - Select New private window
* In Safari – Select File > New Private Window

Login credentials will be provided to you for the labs

## The APIPCS Management Portal Environment

Here is what you will see as soon as you login into API Platform Cloud Service Management Portal:

Let’s go through each component quickly. There is nothing to do here yet.

### APIs

APIs are the set of managed services that can be deployed to Gateways and consumed by applications. The APIs list page allows you to create, view and mange APIs.

### Applications

Applications represent the applications API consumers use to send requests to your APIs. Consumers register applications to APIs they use. The Applications List page displays all applications created with the Management and Developer portals.

### Gateways

Gateways provide the security and access control runtime layer for APIs. Each API is deployed to a gateway node from the Management Portal or via the REST API. The Gateways list page, allows you may view & manage Gateways for your enterprise and view the deployed APIs and requests (API deployment requests or node registration requests) that need attention.

### Oracle APIP Cloud Service Roles

Administration, management, API discovery and development and monitoring tasks have been separated by Roles in APIPCS.

Roles determine which interfaces a user is authorized to access and the grants they are eligible to receive. Administrators assign one or more of these roles to API Platform Cloud Service users and groups: Administrator, API Manager, Application Developer, Gateway Manager, Gateway Runtime, and Service Manager.

The table below describes each of the available roles.

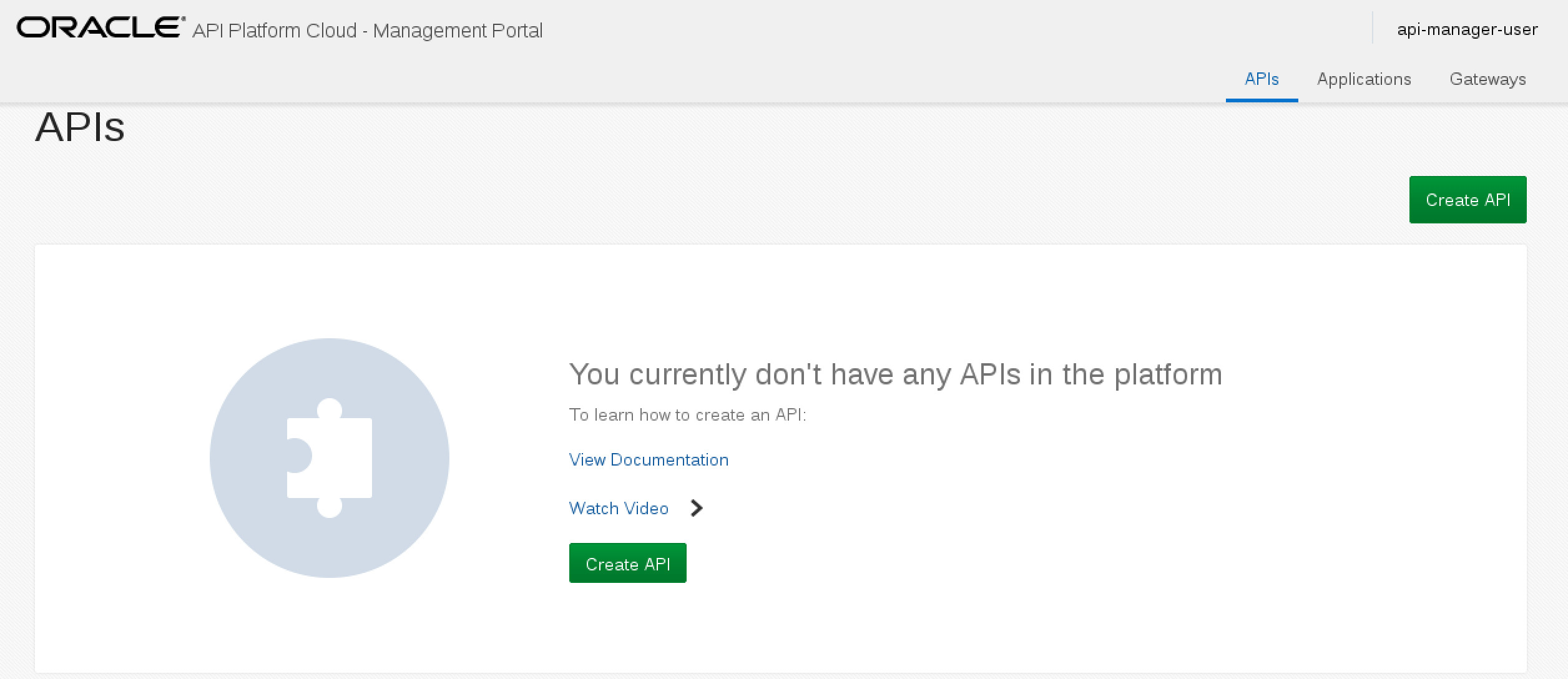
|  |  |
| --- | --- |
| Name | Description |
| Adminitrator | System Administrators responsible for managing the platform settings. Administrators possess the rights of all other roles and are eligible to receive grants for all objects in the system. |
| API Manager | People responsible for managing the API lifecycle, which includes designing, implementing, and versioning APIs. Also responsible for managing grants and applications, providing API documentation, and monitoring API performance. |
| Application Developer | API consumers granted self-service access rights to discover and register APIs, view API documentation, and manage applications using the Developer Portal. |
| Gateway Manager | Operations team members responsible for deploying, registering, and managing gateways. May also manage API deployments to their gateways when issued the Deploy API grant by an API Manager. |
| Gateway Runtime | This role indicates a service account used to communicate from the gateway to the portal. This role is used exclusively for gateway nodes to communicate with the management service; users assigned this role can’t sign into the Management Portal or the Developer Portal. |
| Service Manager | People responsible for managing resources that define backend services. This includes managing service accounts and services. |

Next, let’s start to implement our scenario.

Lab Execution

# Implement an API

You will now assume the role of an API Manager and create APIs for your organizations billing and usage backend services. Log into the Management Portal as described above.

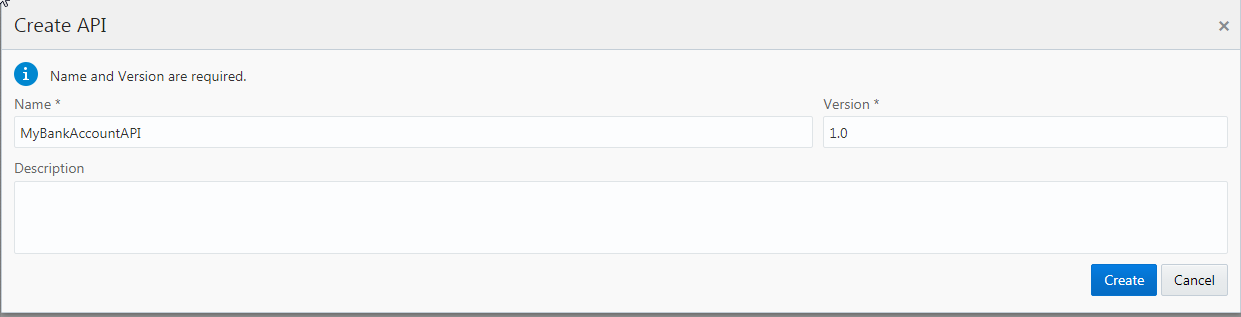
Log into the API Manager Portal as the **api-manager-user-XX (Where XX is your usernumber)**

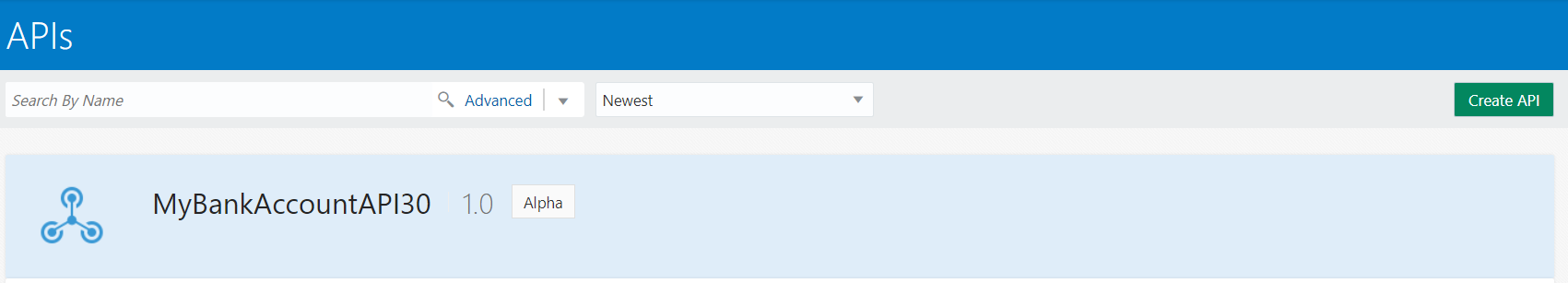
## Create an API

Now you’ll create your first API.

To create an API:

1. From the **APIs** tab, click **Create New API**.
2. Enter **MyBankAccountAPIXX** in the **API Name** field. (Where XX is your User Number)
3. Enter **1** in the **Version** field.
4. **(Optional)** Populate the description field in this form as you would like.
5. Click **Create**.





## Configure Endpoints

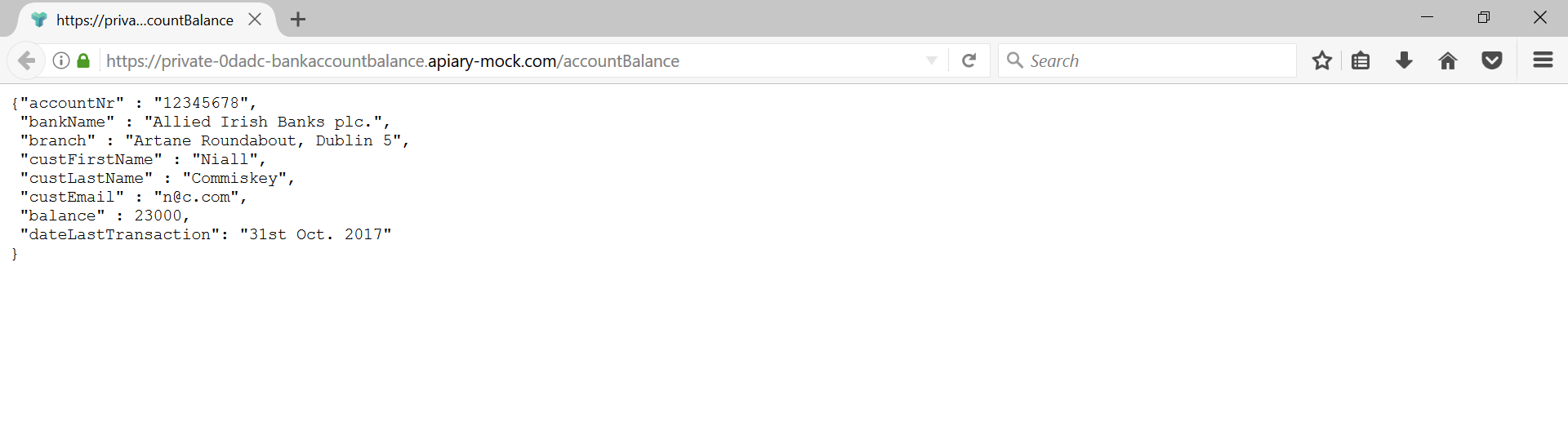
Endpoints are the locations at which an API sends or receives requests or responses. APIs in API Platform Cloud Service have two endpoints in the request flow: the API Request and the Service Request.

* The **API Request** is the location at which developers who consume your APIs will send their requests. This endpoint resides on the gateway on which the API is deployed. You will deploy the API later.
* The **Service Request** is the location of the backend service. If all policy conditions are met, this is the location to which the gateway passes the request.

### Configure the API Request

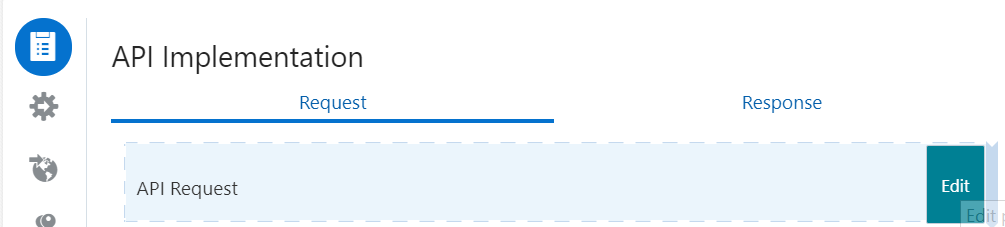
The API request is the endpoint to which requests for your API are sent.

For this lab, we are accessing the api <https://private-0dadc-bankaccountbalance.apiary-mock.com/accountBalance>. This is a mock service which when you call return account balance of a dummy account.

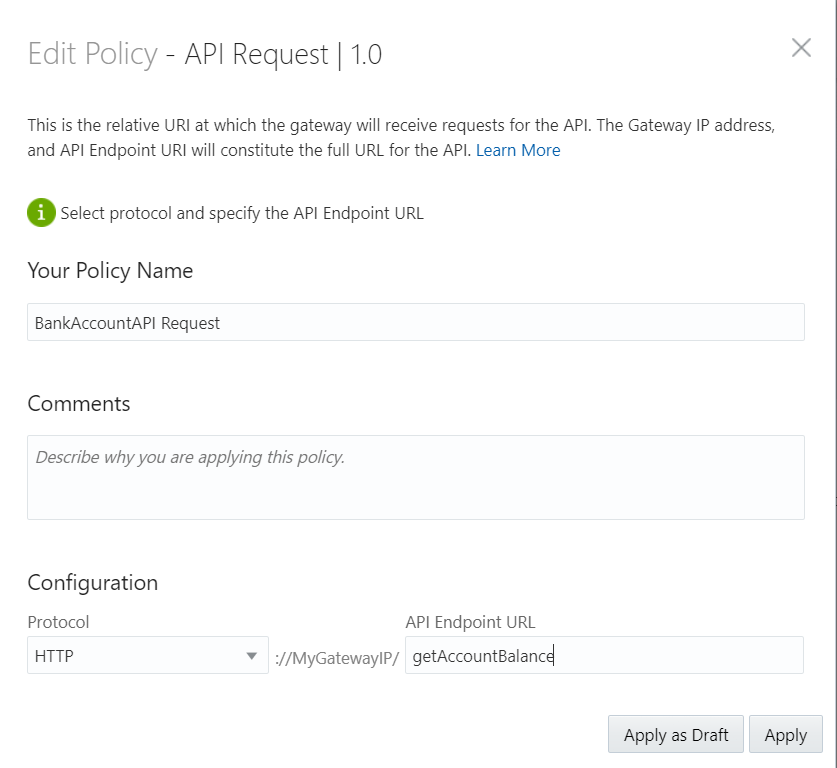


To configure the API Request:

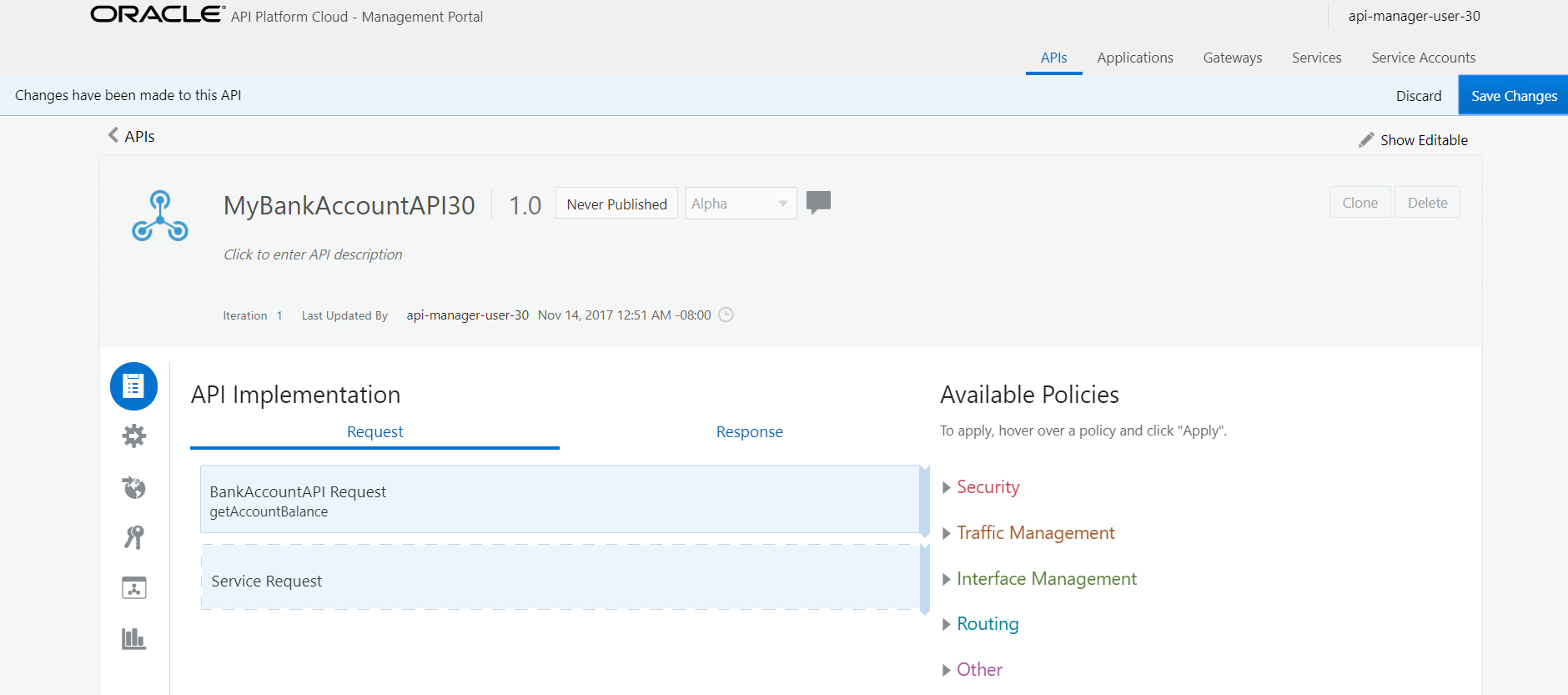
1. From the APIs tab, click the **MyBankAccountAPIXX** API you created in the previous task.
2. Hover over the API Request policy, and then click **Edit**.

  
  
**Note**: If the Apply Policy Dialog does not appear, disable any pop-up blockers that are active and try again.

1. On the Apply Policy Dialog, complete these fields:
   1. Enter **BankAccountAPI Request** into the **Policy Name** field and Click **Next**.
   2. Select **HTTP** from the **Protocol** list. This is the protocol on which the gateway receives requests for this API.
   3. Enter **getAccountBalanceXX (Where XX is your user number)** into the **API Endpoint URL** field
   4. Click **Apply**.



You will now receive the notification that changes have been made to this API, click Save Changes



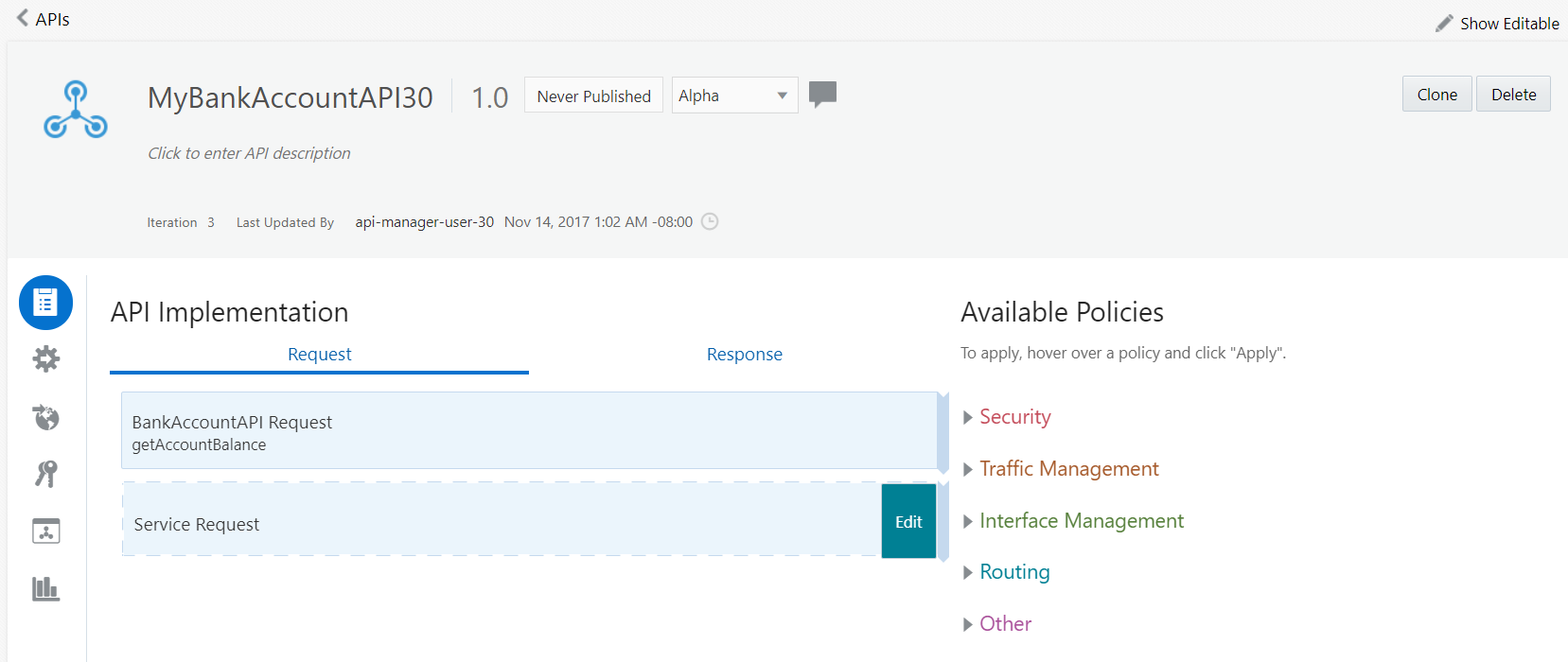
### Configure the Service Request

The service request is the URL at which your backend service receives requests. When a request meets all policy conditions, the gateway routes the request to this URL and calls your service.

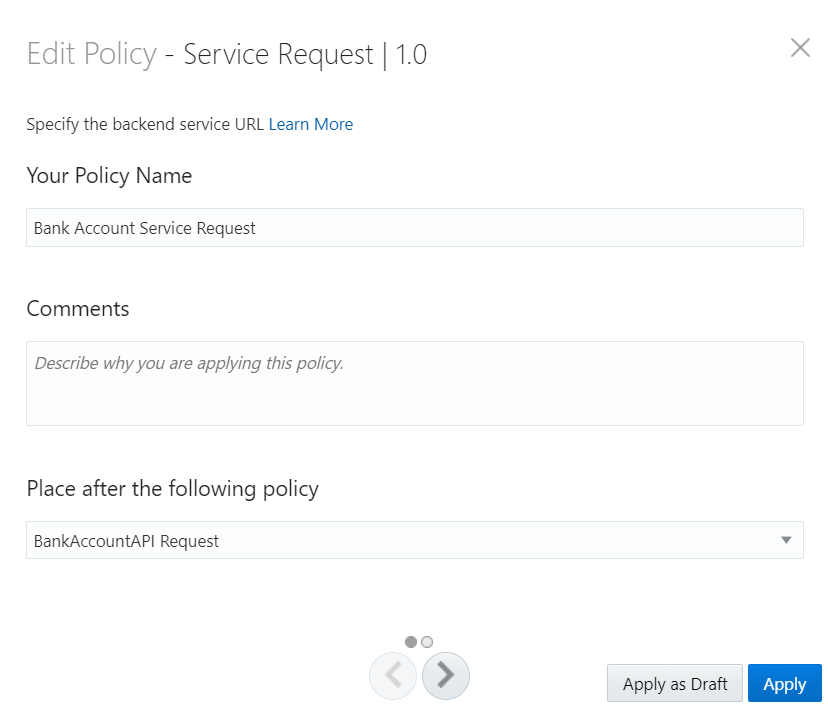
The service request URL can point to any of your service’s resources, not just its base URL. This way you can restrict users to access only a subset of your API’s resources.

To configure the service request:

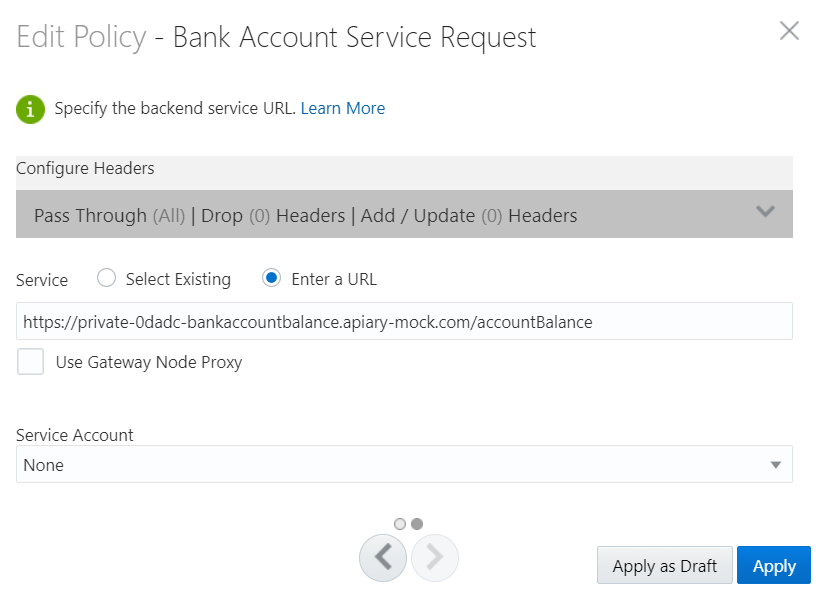
1. Hover over the Service Request policy, and then click **Edit**.



1. On the Apply Policy Dialog, complete these fields:
   1. Enter **Bank Account Service Request** into the **Policy Name** field, optionally **Comments** and Click **Next**



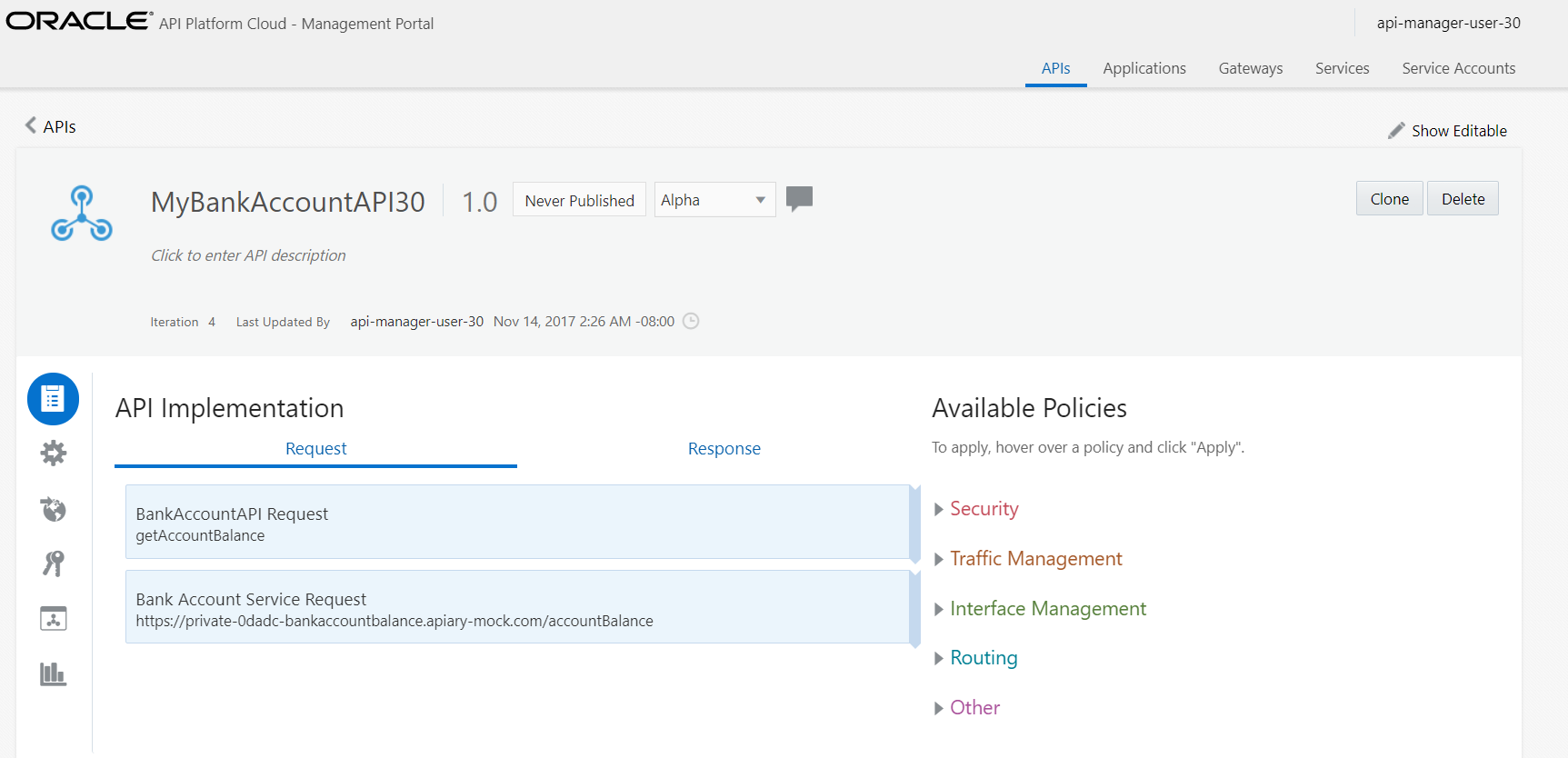
* 1. Enter <https://private-0dadc-bankaccountbalance.apiary-mock.com/accountBalance> in the **Backend Service URL** field.



* 1. Click **Apply**.

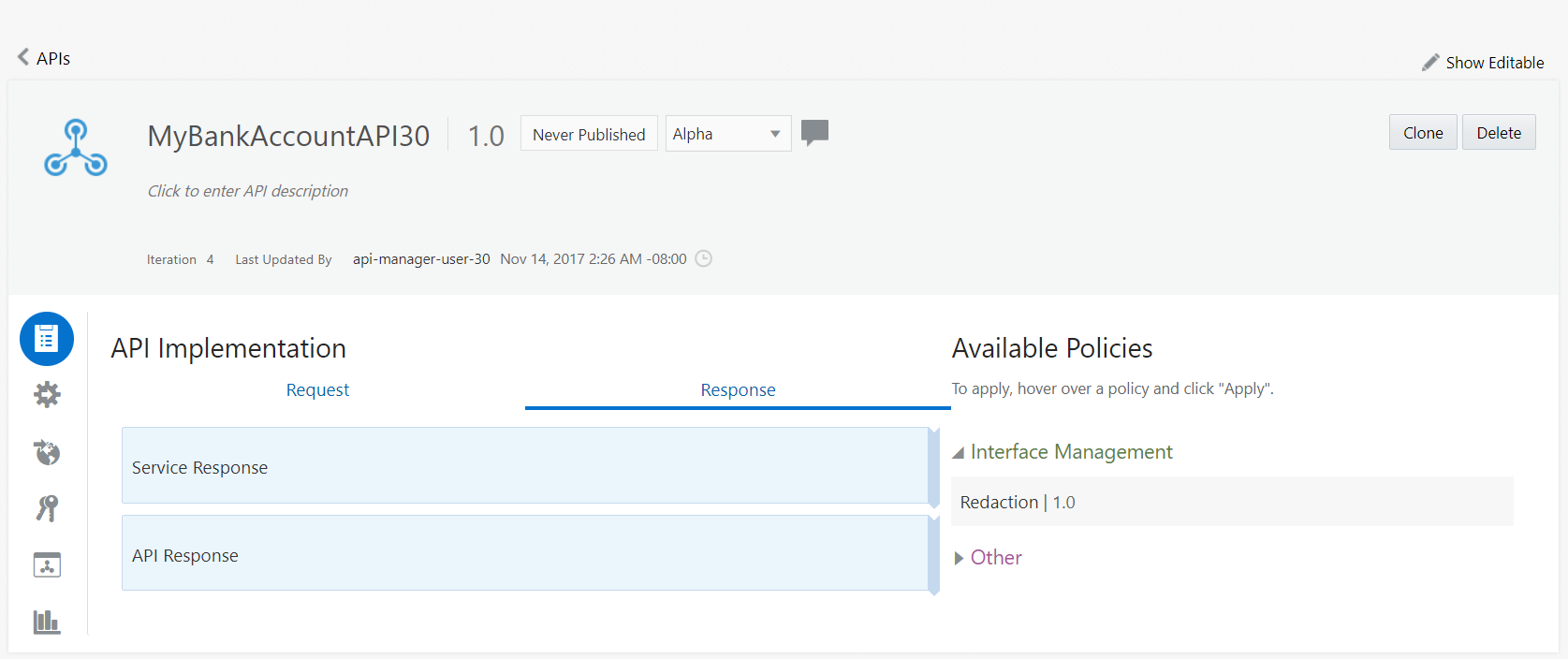
1. Click **Save** **Changes**.

You will now receive the notification that changes have been made to this API.



### About the Response Flow (informational--no activity in this section)

Click the **Response** tab to view a top-down visual representation of the response flow. The Service and API Response entries cannot be edited.



The service response happens first. The response from the backend service is always the first entry in the outbound flow. You can place additional policies in this flow. Policies are run in order, with the uppermost policy run first, followed by the next policy, and so on, until the response is sent back to the client.

The API Response entry is a visual representation of the point in the outbound flow when the response is returned to the client.

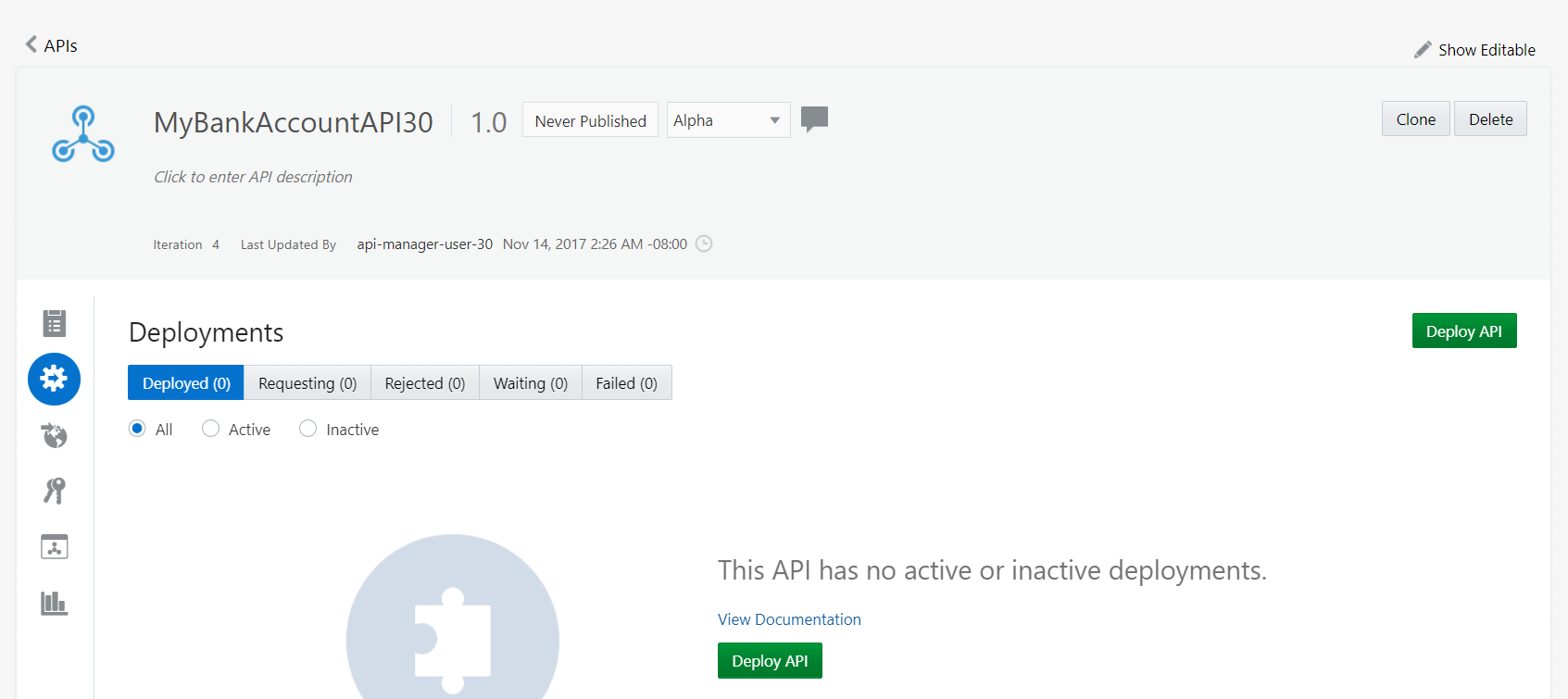
## Deploy an API

To expose the new API that it can be discovered and consumed by Application Developers and other consumers of the API, the API must be deployed to a Gateway. After successfully deploying the API, it receives requests at the endpoint you configured and routes successful requests to the backend service.

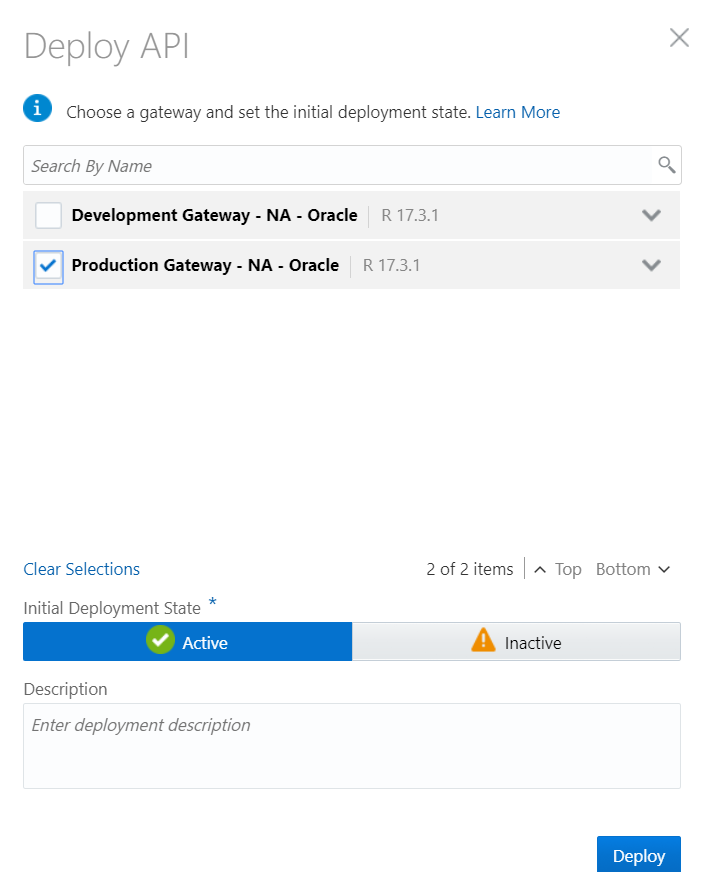
### Deploy an API to the Gateway

To deploy the API:

1. Click the **Deployments** tab.



1. Click **Deploy API**.
2. Select the **Production Gateway**.
3. Select **Active** as the Initial Deployment State.
4. **(Optional)** Complete the description field in as you like.
5. Click **Deploy**.



**Note**: You may have to refresh the page after a few seconds for your API to appear on the Deployed tab.

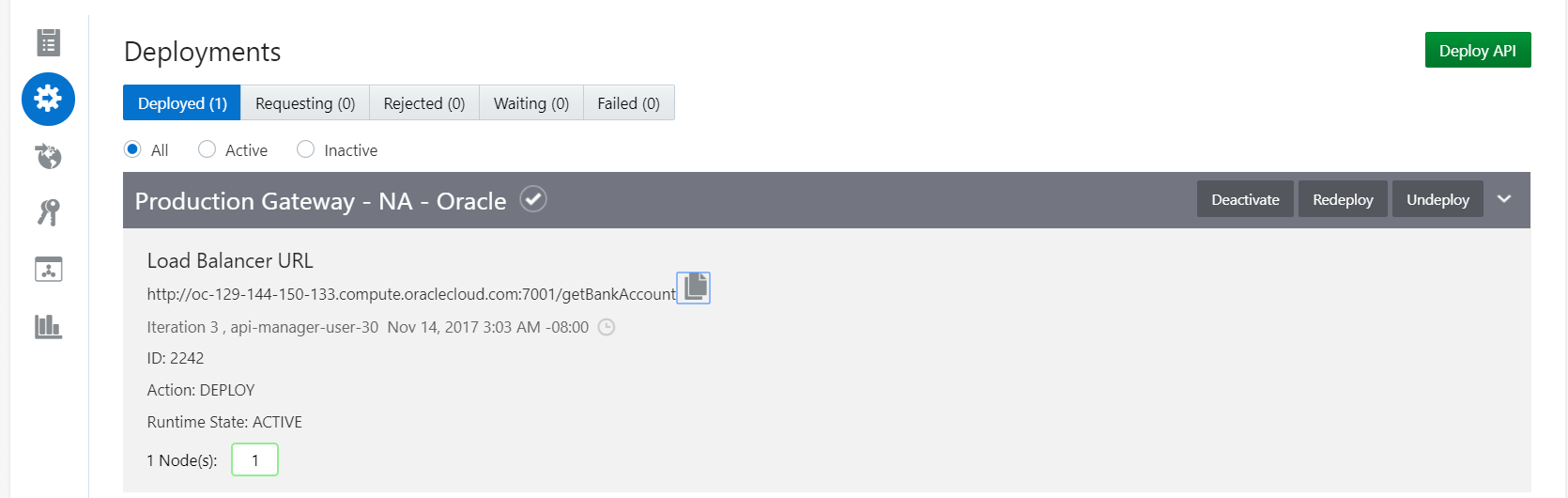
In this task, you’ve set the initial deployment state as **Active**. This means the API is active and can begin receiving requests as soon as it is deployed. You can also choose **Inactive** as a default deployment state. If deployed in an inactive state, the API will not service requests until it is changed to an active state. This is useful if you want to verify that the deployment is successful before enabling an API or if you want to enable it later.

### Invoke the API & Validate your API Deployment

Now that the API is deployed, you can invoke it.

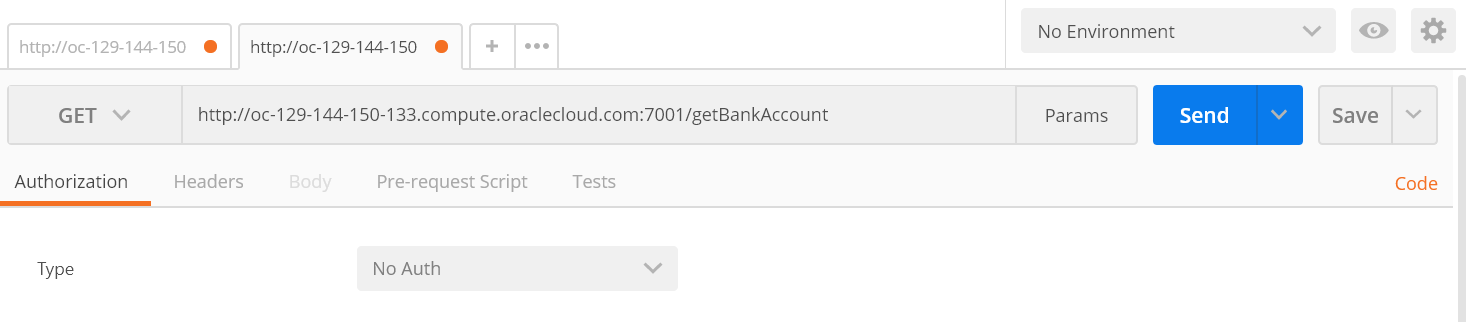
You can use any api testing tool to test the API. We have used Postman which can easily downloadable from Google Chrome Apps.

Once the service is deployed, copy the url from Deployment tab



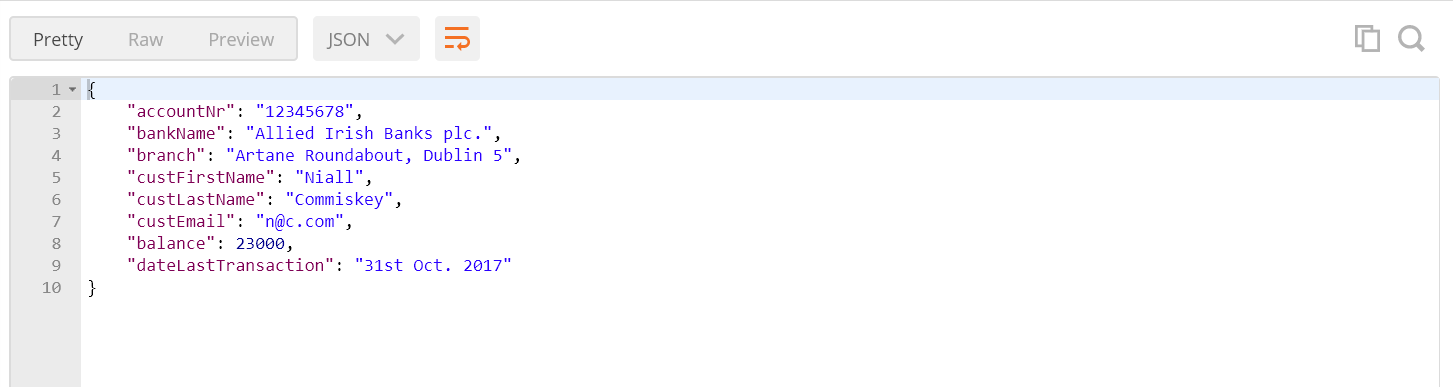
To send a request to your deployed API:

1. Run the **Postman Application**
2. Ensure the method is set as **GET**.
3. Enter your url in the request URL



1. Click **Send**.

Because you have not applied any policies, the request is passed to the backend service. You receive a response like the following screenshot. Later you’ll add policies and send additional requests to validate each of the policies you’ll add.



# Manage API Policies

Policies in API Platform Cloud Service serve a number of purposes. You can apply any number of policies to an API definition to secure, throttle, limit traffic, route, or log requests sent to your API. Depending on the policies applied, requests can be rejected if they do not meet criteria you specify when configuring each policy. Policies are run in the order they appear on the Request and Response tabs. A policy can be placed only in certain locations in the execution flow. This lab explicitly indicates where you should place each policy.

### APIPCS Policies

After you create an API, apply policies to configure the request and response flows.

Policies in the request flow secure, throttle, route, manipulate, or log requests before they reach the backend service; polices in the response flow manipulate and log responses before they reach the requesting client.

The table below describes each of the available policies.

|  |  |  |
| --- | --- | --- |
| **Policy Type** | **Types** | **Description** |
| Security | Key Validation Policies  Basic Authentication Policies  Service Level Auth Policies  IP Filter Validation Policies  OAuth 2.0 Policies  CORS Policies | Policies that determine who can send requests to your services. |
| Traffic Management | API Throttling–Delay Policies  Application Rate Limits Policies  API Rate Limiting Policies | Policies that manage the volume of traffic sent to your services. |
| Interface Management | Interface Filtering Policies  Redaction Policies  Header Validation Policies  Method Mapping Policies | Policies that manage the service interfaces clients are permitted to access. |
| Routing | Header-Based Routing Policies  Gateway-Based Routing Policies  Application-Based Routing Policies  Resource-Based Routing Policies | Policies that route requests to different service URLs depending on the requesting application, the resource requested, and other conditions. |
| Other | Service Callout 1.0 Policies  Logging Policies  Groovy Script Policies | Policies not belonging to one of the above categories. |

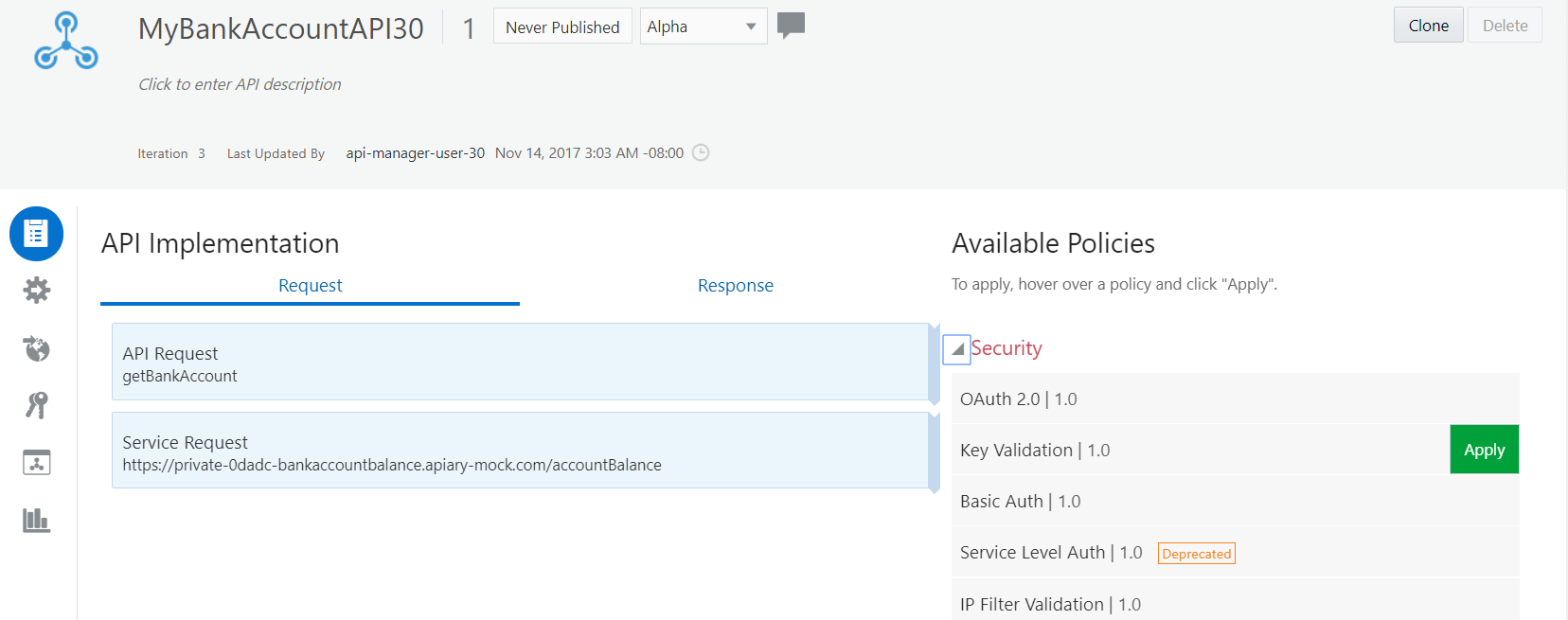
### Add a Key Validation Policy

In this section, you’ll add a validation policy to your API to ensure that Application Developers consuming it have registered it to an application, the application is registered, and a current, valid application key is used.

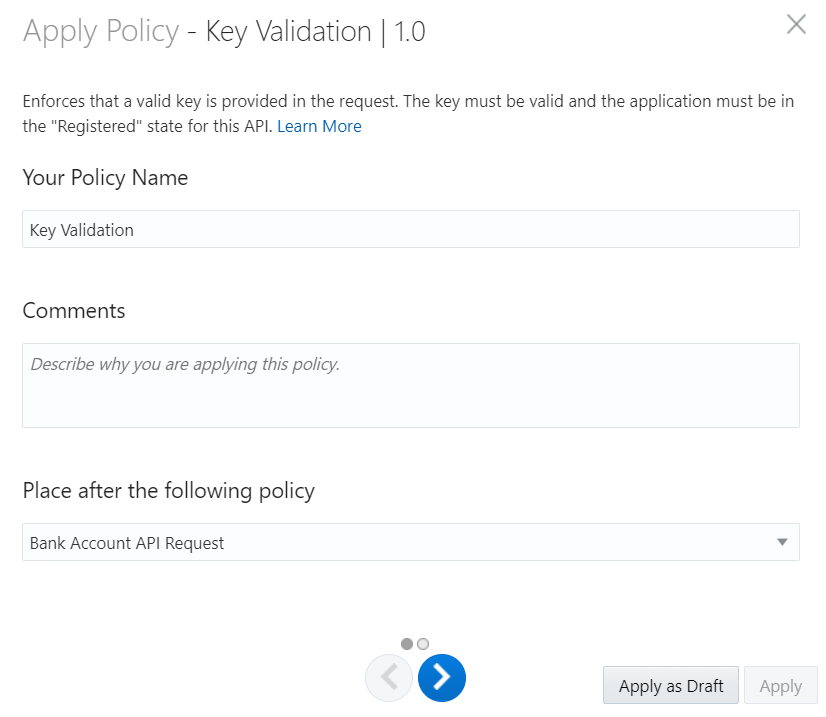
You specify the header or query parameter used to pass the key. At runtime, if the key is not present in the given header or query parameter, or if the application is not registered, the request is rejected; the client receives a 400 Bad Request error if the applicable header or query parameter is passed or a 403 if a valid key is not passed.

To configure the key validation policy:

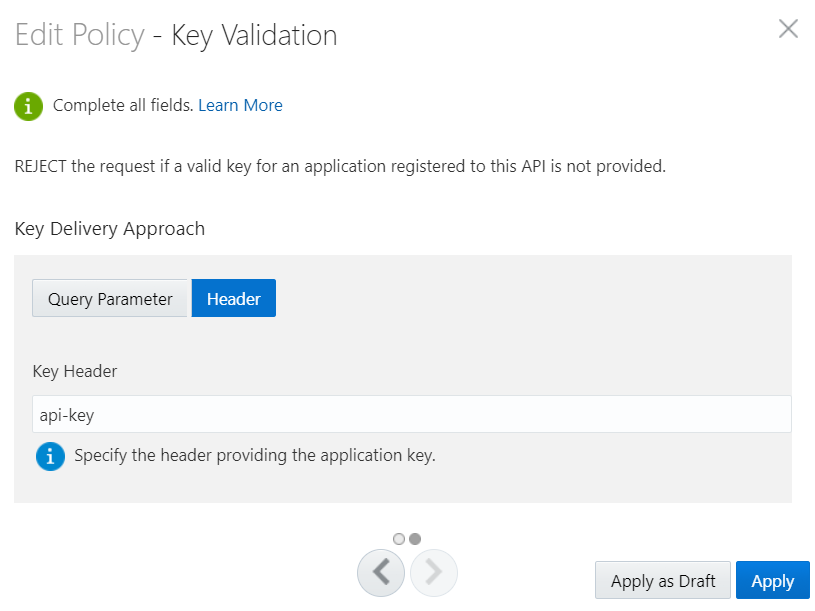
1. Back in the Management Portal, click the **API Implementation** tab.



1. In the Available Policies section, expand **Security**, hover over **Key Validation**, and then click **Apply**.
2. On the Apply Policy Dialog, complete these fields:
   1. Enter **Key ValidationXX** the **Name** and **(Optional)** **Description** fields as you would like.
   2. Ensure that Bank Account A**PI Request** is selected from the “**Place Policy After..”** list and click **Next**.



* 1. Click the **Header** tab. ( If you do not have PostMan to test your service then click **QueryParameter** instead of Header.
  2. Enter the text **api-key** into the **Header** field. The application key must be sent in a header named **api-key** for the request to be passed to the next policy or to the service request endpoint.



* 1. Click **Apply**.

1. Click **Save Changes**.

**Note**: You may have to scroll to the top of the browser window to see the Save button.

### Add an Application Rate Limiting Policy

Follow this task to add an Application rate limiting policy. Application rate limiting policies are used to control how many requests are routed to an API from each application during a given period. If this threshold is exceeded, the gateway rejects subsequent requests from that application during this period.

**You also apply this as a draft policy.** You’ll apply it and redeploy the API later in the lab.

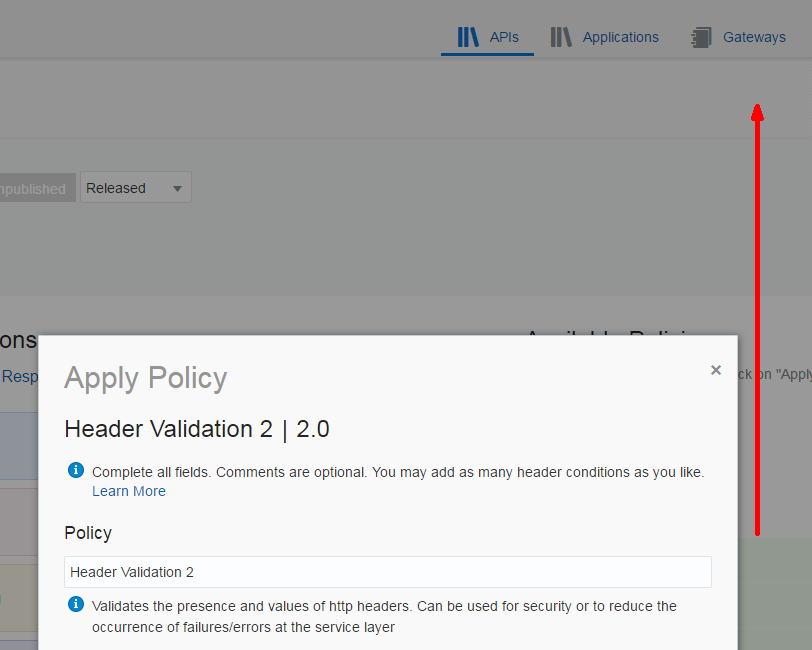
To add an Application rate limiting policy:

1. In the Available Policies section, expand **Traffic Management**, hover over Application Rate Limiting, and then click **Apply**.  
     
   **Note**: Make sure you’ve added the **Application Rate Limiting** policy, not the API Rate Limit policy. These policies behave differently; if you use the wrong one you may experience results different than those described in this document.
2. On the Apply Policy Dialog, complete these fields:
   1. **(Optional)** Complete the name and description fields as you would like.
   2. Ensure that **Key Validation** is selected from the **Place Policy After** list.
   3. Enter **10** into the **Rate Limit Per Application** field. This is the number of requests that will be passed during each time interval.
   4. Select **Minute** from the **Time Interval** list.  
      You can add additional conditions. This is useful if you want to limit the number of requests during multiple time periods (seconds, minutes, and hours, for instance) without creating an additional rate limiting policy for each period.
   5. Click **Apply as Draft**.
3. Click **Save Changes**.

### Add a Header Validation Policy

Complete this task to add a header validation policy, which is used to enforce the presence or values of HTTP headers sent with requests. Requests not meeting the policy conditions are rejected.

To add a header validation policy:

1. In the Available Policies section, expand **Interface Management**, hover over **Header Validation**, and then click **Apply**.  
   **Note**: Header Validation and Header Validation 2 are the same policy. It doesn’t matter which one you apply.
2. On the Apply Policy Dialog, complete these fields:
   1. **(Optional)** Complete the name and description fields as you would like.
   2. Ensure that **Application Rate Limiting** is selected from the **Place Policy After** list.
   3. Select **PASS** and **ANY** from the lists in the Header Conditions section. This passes all requests that include headers matching any of the conditions you specify. Requests without matching headers are rejected.  
        
      **KNOWN ISSUE: If you are unable to select anything in the lists on the Apply Policy dialog, scroll to the top of the browser window and drag the dialog to the top of the window:**
   4. Enter **tenant-id** into the **Name** field.
   5. Select **>=** from the Operator list.
   6. Enter **1** into the **Value** field.
   7. Click **Apply**.
3. Click **Save Changes**.

### Add an Interface Filtering Policy

Complete this task to add an interface filtering policy, which is used to filter requests based on the resources and methods specified in the request. In this lab, you’ll configure this policy to pass GET requests to two resources, **/usage/current/\*** (which returns a customer’s billing information for the current period) and **/estimate/\*** (which returns an estimate of a customer’s next bill). Requests using other methods or to other resources are rejected.

To add an interface filtering policy:

1. In the Available Policies section, expand **Interface Management**, hover over **Interface Filtering**, and then click **Apply**.
2. On the Apply Policy Dialog, complete these fields:
   1. **(Optional)** Complete the name and description fields as you would like.
   2. Ensure that **Header Validation** is selected from the **Place Policy After** list.
   3. Select **PASS** from the list.
   4. Enter **/usage/current/\*** into the **Resources** field and press **Enter**.
   5. Enter **/estimate/\*** into the **Resources** field and press **Enter**.
   6. Click in the **Methods** field, and then select the **GET** method.
   7. Click **Apply**.
3. Click **Save** **Changes**.

### Add a Resource-Based Routing Policy

Complete this task to add a resource-based routing policy, which can be used to route requests to specific resource paths to different service response URLs. Apply this policy to combine two backend services, one returning a customer’s current energy usage information and another returning a customer’s estimated usage for the next period, into a single interface used by application developers.

To add a resource-based routing policy:

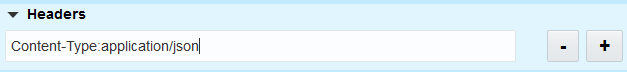
1. In the Available Policies section, expand **Routing**, hover over Resource-based Routing, and then click **Apply**.
2. On the Apply Policy Dialog, complete these fields:
   1. **(Optional)** Complete the name and description fields as you would like.
   2. Select **Interface Filtering** from the **Place Policy After** list.
   3. Enter **/estimate** in the **Resource Path(s)** field and press **Enter**. You can add multiple paths to a single condition, but you’ll add only one for this policy.
   4. Enter <http://eserv2.oracle.com:7778> in the **Set Service URL To** field. When any of the resource paths above are included in the request, it is routed to this service request URL instead of the API’s default service request URL.
   5. Leave the **Keep Default Service Request URL** selected in the **Otherwise** section. If you wish, you can specify a service URL other than the default to which requests are routed if none of the conditions in this policy are met.
   6. Click **Apply**.
3. Click **Save Changes**.

### Add a Service Callout Policy

Complete this task to add a service callout policy, which sends a request to another service from the request flow. You can specify a payload to send with your request to the external service. You can choose to pass or reject requests based on the response code received from that service. In this task you’ll send a request to another of the energy company’s services; this service reads an account number sent in the payload and returns a 200 status code if the account number exists and a 404 if it does not. If an account number is not found, the service callout policy does not pass the initial request to the backend service.

In this example, you hard-code an account number in the payload to the service. Callouts to this service will always succeed. In a real-world scenario, you can dynamically determine this account number using Groovy script notation, but this is not supported for the beta.

To add a service callout policy:

1. In the Available Policies section, expand **Other**, hover over **Service Callout**, and then click **Apply**.
2. On the Apply Policy Dialog, complete these fields:
   1. **(Optional)** Complete the name and description fields as you would like.
   2. Ensure that **Header Validation** is selected from the **Place Policy After** list.
   3. Ensure that **PASS** and **Specific** are selected from the lists.
   4. Enter **200** into the **Status Code(s)** field and press **Enter**.
   5. Click in the Methods field, and then select **POST**.
   6. Enter [http://eserv2.oracle.com:7778/**account**](http://eserv2.oracle.com:7778/account)in the **Service URL** field.
   7. In the **Headers** section, click the **Add a new header** (**+**) icon.
   8. Enter **Content-Type:application/json** into the field.  
      
   9. Paste the following sample into the **Request Payload** field:  
      {

"account\_number":"4859634"

}

* 1. Click **Apply**.

1. Click **Save Changes**.

### Add a Groovy Script Policy

Follow this task to add a Groovy script policy, which uses the logic in a Groovy script to pass or reject requests based on aspects of the calling context, such as headers, query parameters, etc.

This task uses a Groovy script to check that a Content-Type header is sent with the request. If one is not sent, the script adds it with a value of **application/json**. For convenience, the script is provided below.

To add a Groovy script policy:

1. In the Available Policies section, expand **Other**, hover over **Groovy Script**, and then click **Apply**.
2. On the Apply Policy Dialog, complete these fields:
   1. **(Optional)** Complete the name and description fields as you would like.
   2. Ensure that **Resource Based Routing** is selected from the **Place Policy After** list.
   3. Paste the following script into the Groovy Script field:  
      if (context.clientRequest.getHeader("Content-Type") == null)  
       context.southboundCallout.withHeader("Content-Type", "application/json")
   4. Click **Apply**.
3. Click **Save Changes**.

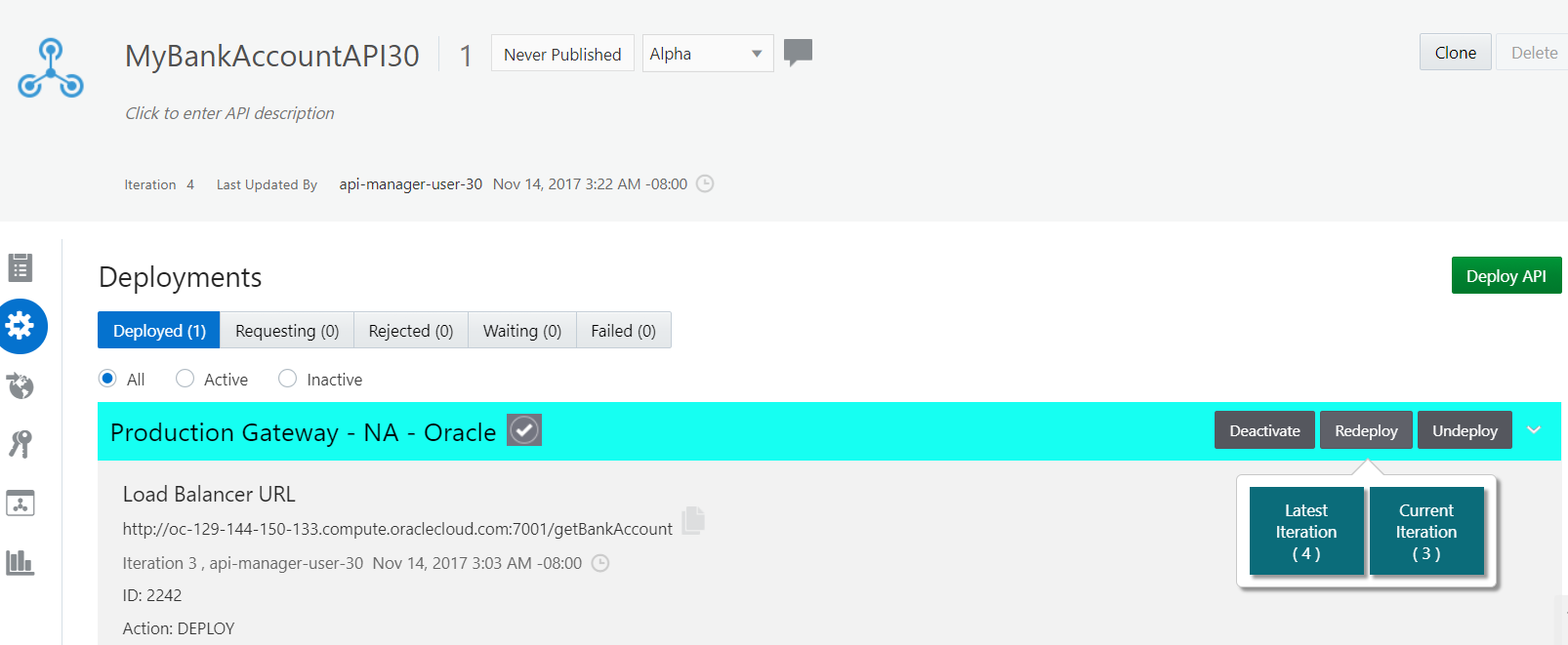
After you’ve configured all of the policies, your screen should look something like this:

### Re-Deploying an API

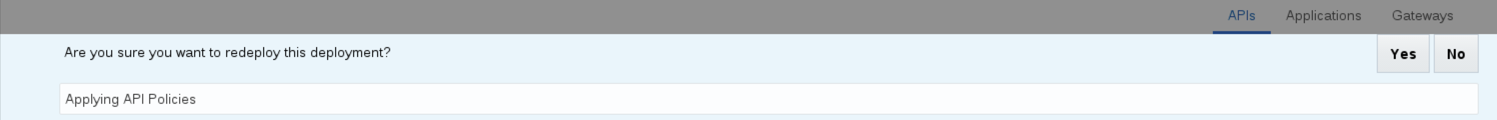
In this exercise, you are an API Manager with the GW Deploy grant for the production gateway. The deployment request is automatically approved; you will not have to switch to a Gateway Manager to approve the request. Because we’ve already deployed an API, we’ll redeploy the API to the gateway, which enables the policies you created earlier. **Note**: This process flow is grant dependent. This process varies for environments with different grant schemes.

To redeploy an API:

1. Click the **Deployments** tab.
2. Hover over the **Production Gateway** deployment, and click **Redeploy** when it appears.



1. Select the **Latest Iteration**
2. When prompted, Enter **Applying API Policies** into the **Reason Field**, then click **Yes**.



**Note**: You may have to refresh the page after a few seconds for your API to appear on the Deployed tab.

## Testing and Validating an API and its Policies using Postman

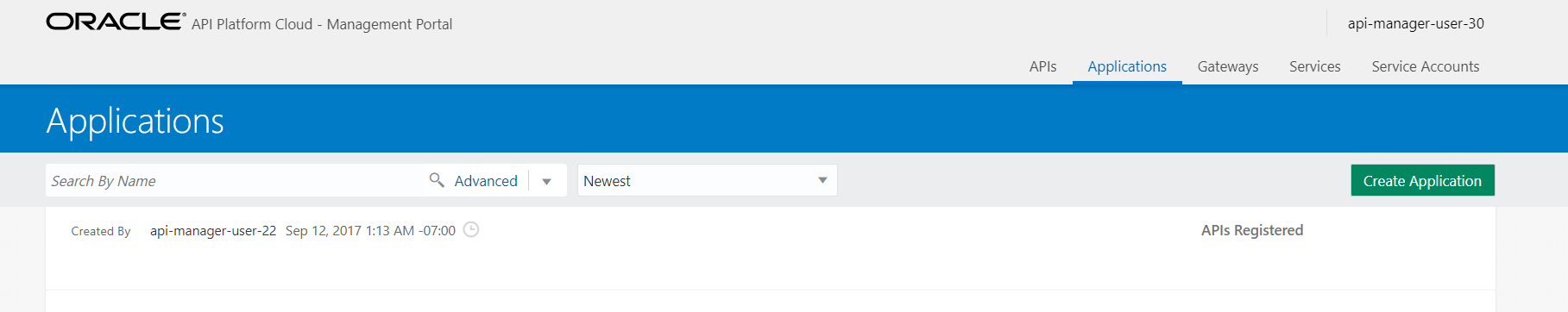
To Test or Consume an API with the Key Validation policy, validate the Application Key (api-key header variable), you must first create and Application and Register the API with the application.

### Creating an Application

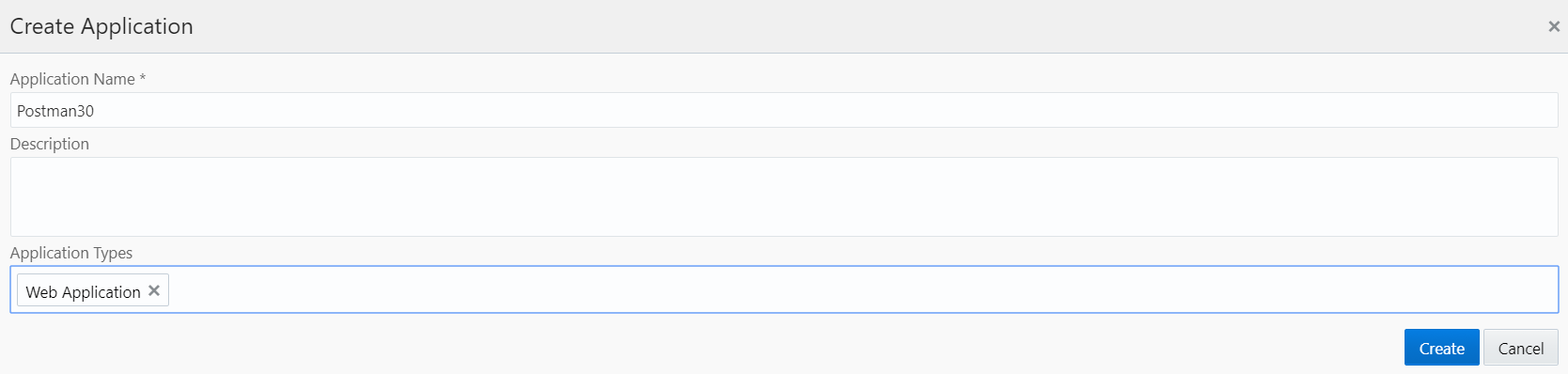
Applications represent the applications API consumers use to send requests to your APIs. Consumers register applications to APIs they use.

To create an Application:

1. From the **Applications** tab, click Create **Application**.



1. Enter **PostmanXX (where XX is your user number)** in the **Application Name** field
2. **(Optionally)** populate the **Description** field
3. Select **Web Application** in the **Application Types** field
4. Click **Create**.



### Registering and Application

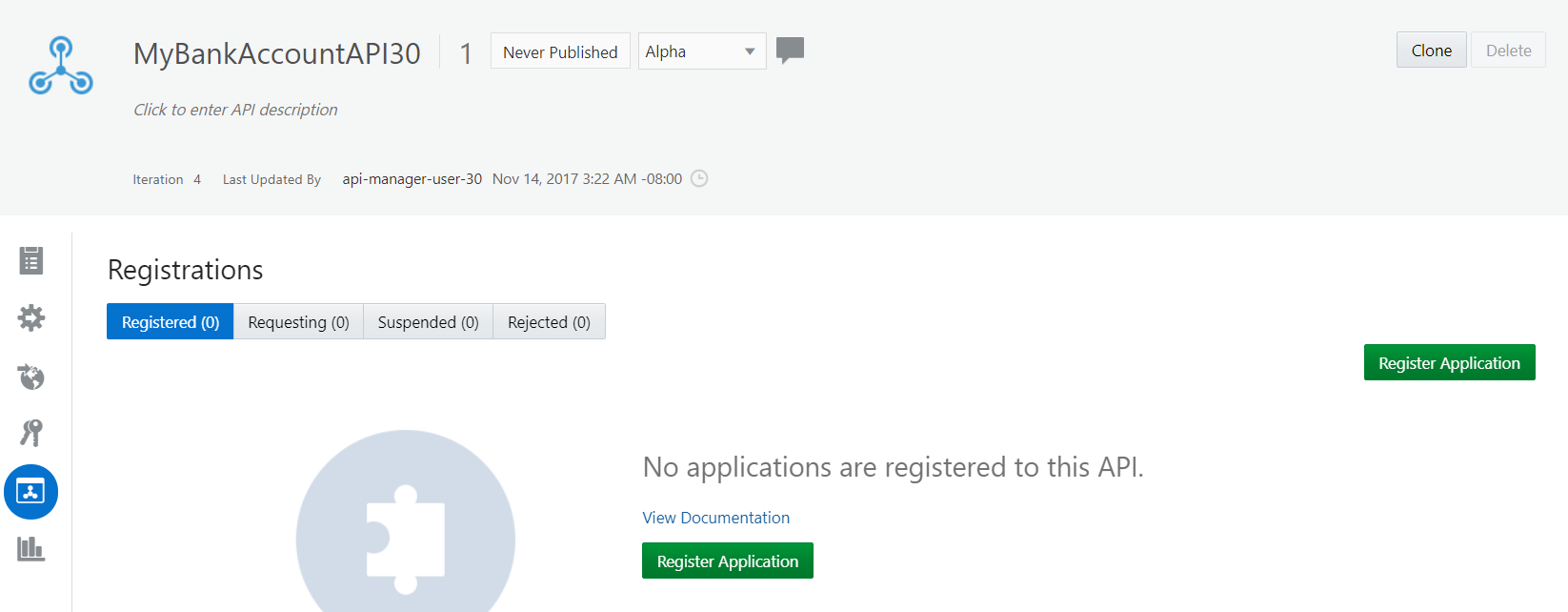
Registrations allow you to determine which applications are allowed to send requests to your APIs.

API consumers register their applications to APIs they want to use. Application developers use the Developer Portal to register to APIs; API managers use the Developer Portal and the Management Portal.

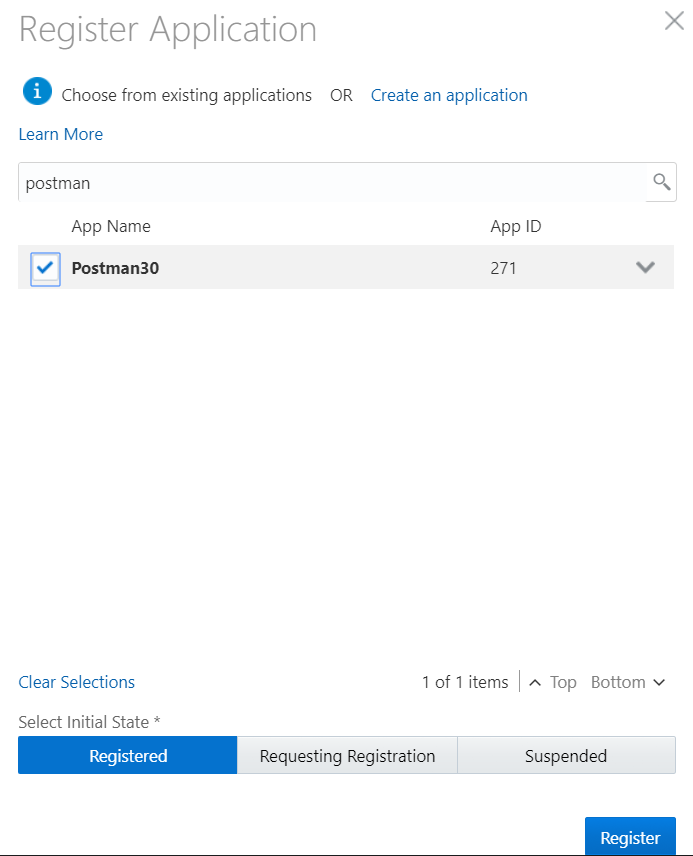
API Platform Cloud Service uses application registrations to determine which users have the right to use an API. Each application is assigned a key.

To register the Application with an API:

1. From the APIs tab, click yourAPI(MyBankAccountAPIXX) you created



1. Click the **Registrations** tab
2. Click **Register Application**
3. Select the **PostmanXX** Application
4. Click **Register**



Your Application and API registration is now complete and you may test and validate the API and its policies.

### Validate your API’s Policies

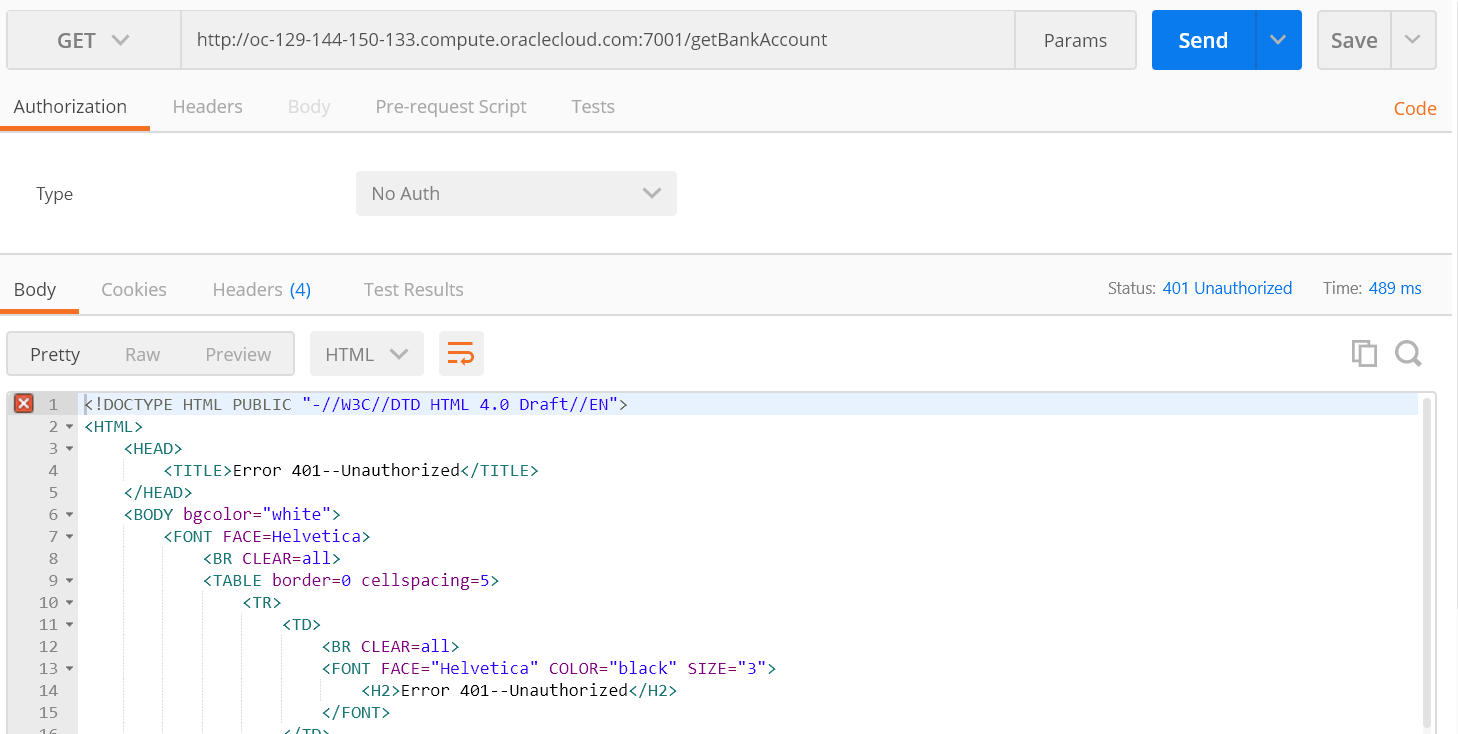
To Test and Validate you API’s policies from Postman successfully, you will need to capture your Application Key, as our policy requires a valid App Key in the api-key header to be submitted along with the service request.

To send a request to your deployed API:

1. Run the **Postman Application**
2. Ensure the method is set as **GET**.
3. Rerun your request that you had run earlier.

Run a Validate Test – to Fail

1. Click **Send** without adding any Header variables



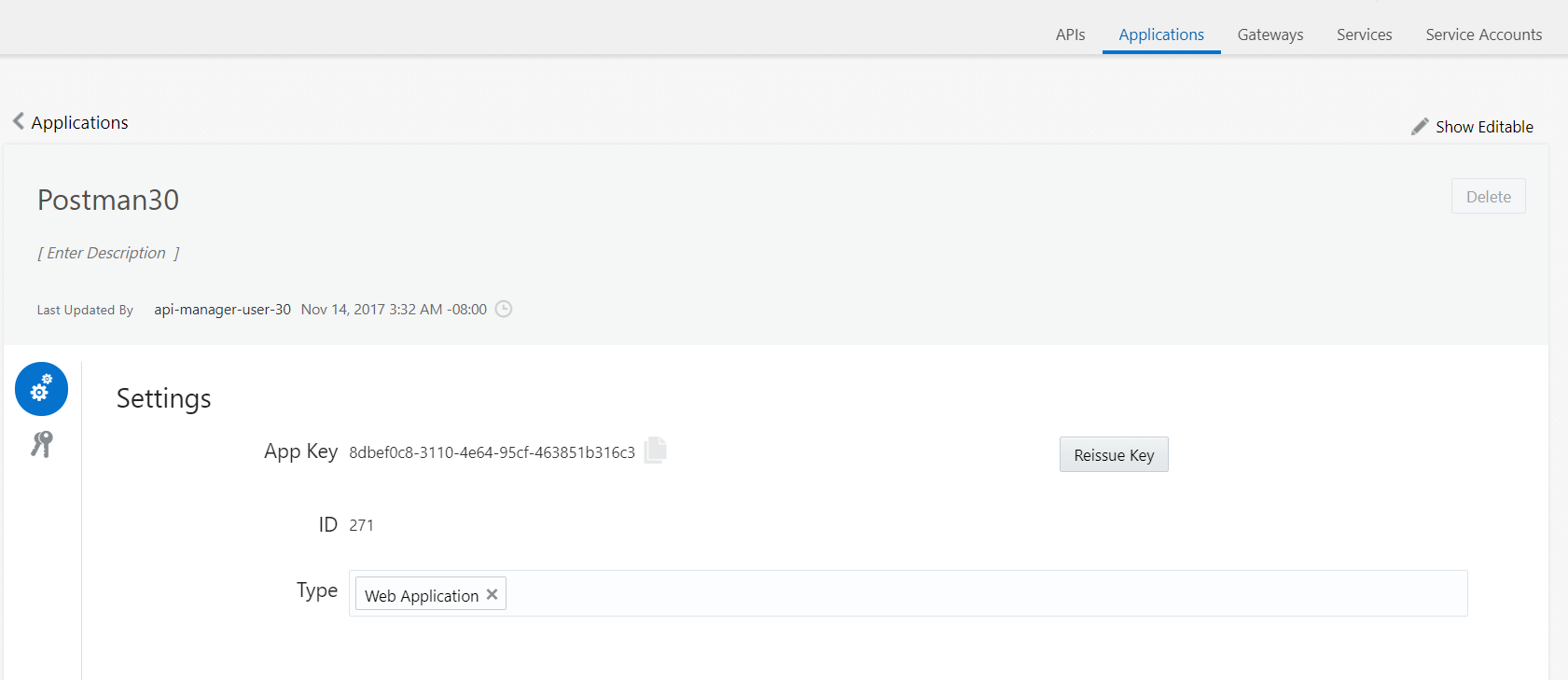
Since the API has got key validation policy enabled, it is throwing 401 – Unauthorized error.

Let’s now run the test with valid key.

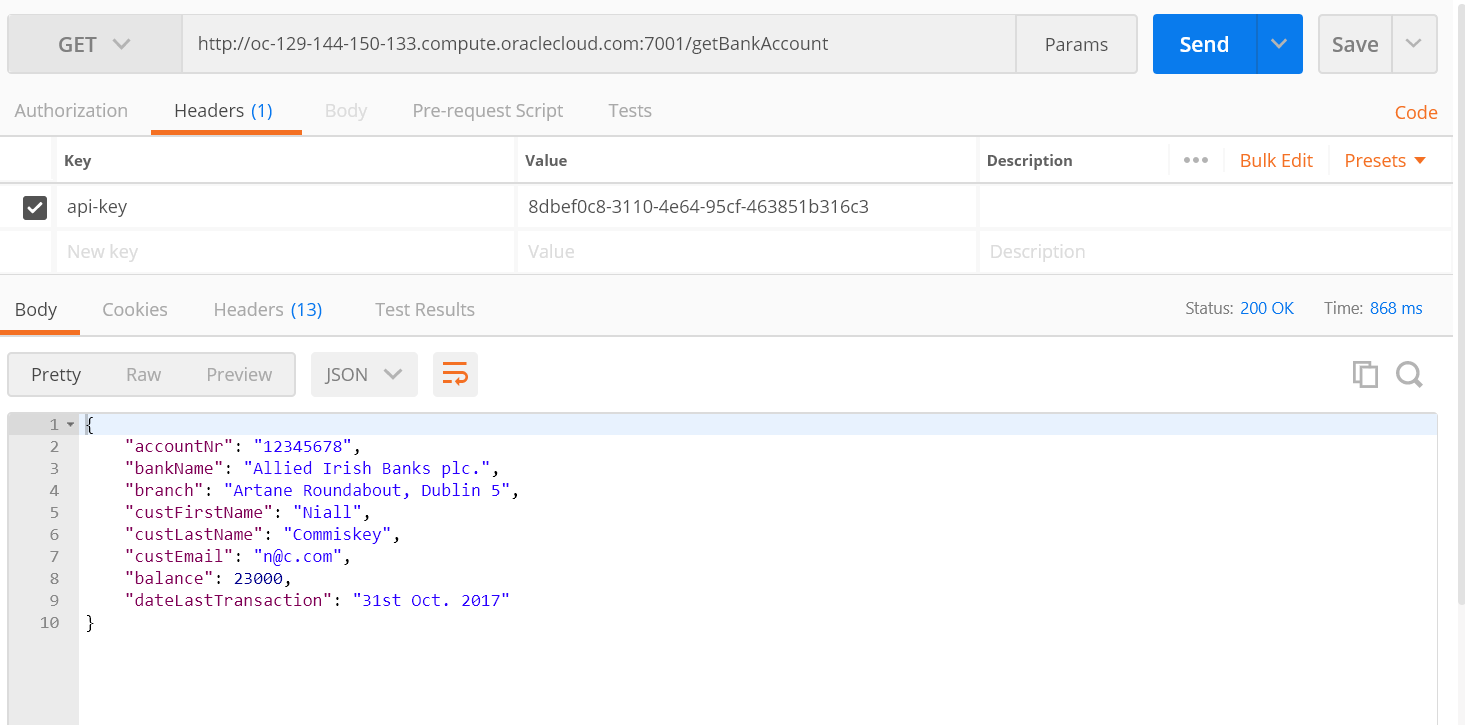
To run a success test, you will need to capture your Application Key, as our policy requires a valid App Key in the api-key header to be submitted along with the service request.

To get you App Key:

1. From the **Applications** tab, click the **PostmanXX** application you created in the previous task.
2. View and **Copy** your **App Key** value



1. From the **Postman** client, select the **Headers** tab
2. Add the following header variable:
   1. Enter api-key in the Key field
   2. Enter (Paste) <your application key> in the Value field
3. Click **Send**.



**<OPTIONAL>** Add a Header Validation Policy to verfiy that the **tenant-id** is **greater than 10**. Test and validate both Failure and Success scenarios.

Congratulations you have completed the APIPCS API Manager lab.