IBM Security

IBM CLOUD IDENTITY

Authentication as a Service with Cloud Identity Verify

Postman Cookbook

Jon Harry

Version 1.2
September 2019

NOTICES

This information was developed for products and services offered in the USA.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

TRADEMARKS

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

IT Infrastructure Library is a Registered Trade Mark of AXELOS Limited.

ITIL is a Registered Trade Mark of AXELOS Limited.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Table of Contents

1 Introduction

IBM Cloud Identity Verify is Authentication-as-a-Service (AaaS). It provides a simple REST API which authorized clients can call to initiate a variety of 2nd Factor Authentication (2FA) mechanisms. This allows application developers to include strong authentication in their applications without the need to implement any of the associated infrastructure (such as an SMS or e-mail gateway, PUSH services) or manage user authentication data.

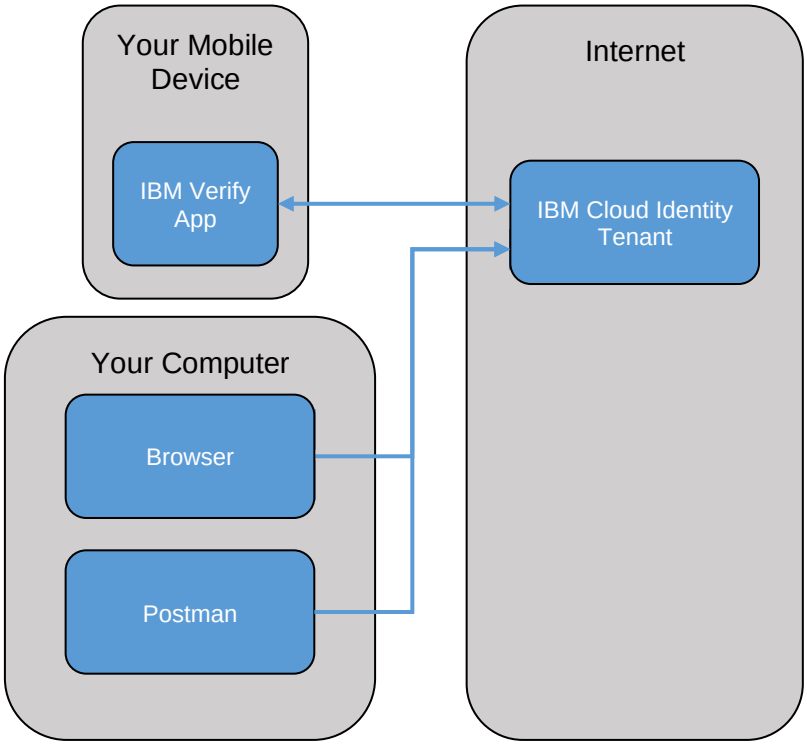
This cookbook provides a step-by-step guide to exploring the Authentication-as-a-Service capabilities provided by IBM Cloud Identity Verify. It uses the Postman utility to drive the REST APIs.

This guide is a follow-on from the **Cloud Identity Basics Cookbook** which describes how to apply for, and set up, an IBM Cloud Identity tenant and a 3rd party SaaS Application. You must complete at least Exercises 1 and 2 from the Basics Cookbook as a pre-requisite for this cookbook.

As with any Software-as-a-Service environment, changes to the service may mean that screenshots and methods described here may differ from the current service. Please make sure you are using an up-to-date version of this document to avoid issues.

1.1 High Level Architecture

The high-level architecture for the environment described in this document may be summarized as follows:



1.2 Required Components

1.2.1 IBM Cloud Identity Connect Tenant

This cookbook assumes that you will use a trial IBM Cloud Identity tenant to run the exercises. Instructions for obtaining this tenant are described in the pre-requisite Cloud ID Basics Cookbook.

1.2.2 Software requirements

You will need the following software to complete this cookbook:

- **Browser** – A browser is required for accessing the Cloud Identity admin console so that you can create an API Client. The browser requires internet connectivity.
- **Postman** – This cookbook uses the Postman utility to act as an API Client. You can install Postman from the following URL: https://www.getpostman.com.

1.2.3 IBM Verify App

To explore the use of TOTP and Mobile Push authentication as a Second authentication factors, you will need to have the IBM Verify application installed on a mobile device. iOS and Android are supported. The mobile device will need internet connectivity for Mobile Push authentication.

Install the IBM Verify app here:
iOS - https://itunes.apple.com/gb/app/ibm-verify/id1162190392
Android - https://play.google.com/store/apps/details?id=com.ibm.security.verifyapp

1.2.4 Other Pre-requisites

You will also need:

- **An e-mail address** to receive initial account information and e-mail One-Time Passwords. You can use a single e-mail address for all requirements in this document.
- **A mobile number** to receive SMS One-Time Passwords. You can use a single number for all requirements in this document.

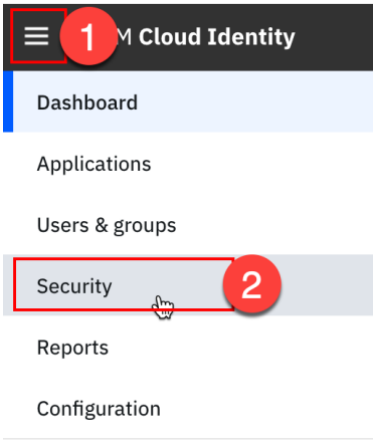
2 Configure Cloud Identity

2.1 Enable Authentication Factors

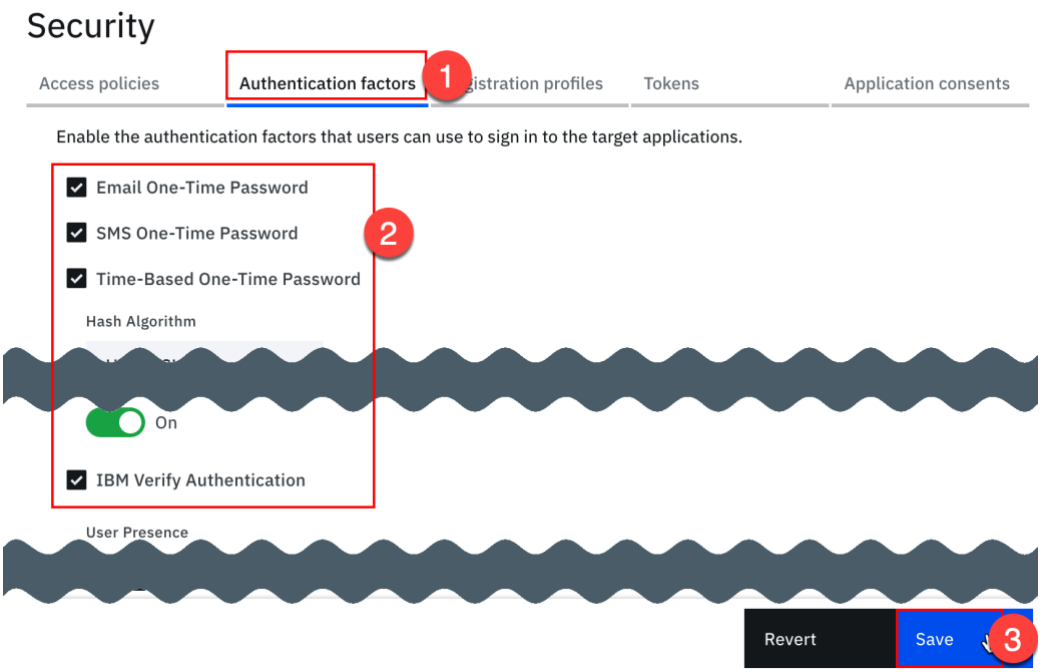
First, you will check that all available authentication factors are enabled in Cloud Identity. You may have disabled them while following other cookbooks.

In your browser, navigate to your Cloud Identity tenant admin UI. The URL will have the form: **https://<*yourtenantid*>.ice.ibmcloud.com/ui/admin**

Authenticate as an administrator.



Open the menu using the burger icon in the upper-left corner and select **Security**.

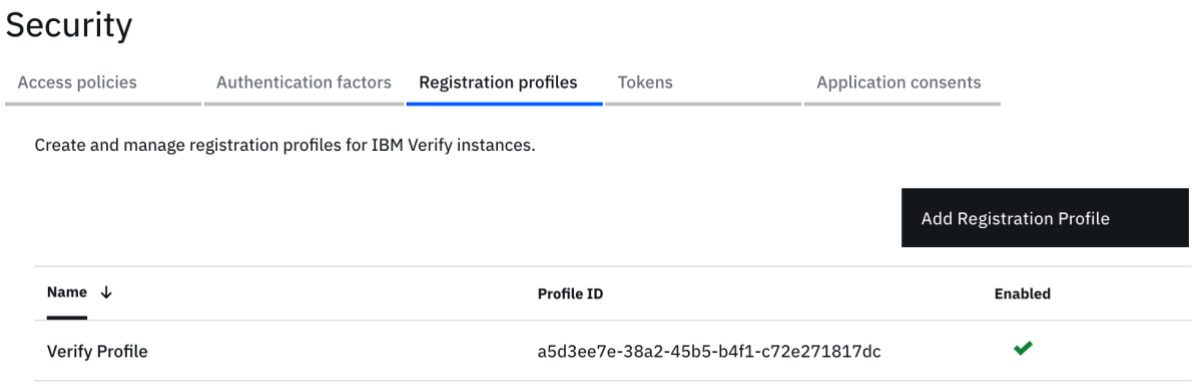


Select the **Authentication factors** tab.

Make sure that the checkboxes for all authentication factors are enabled and, if you made changes, click **Save**.

2.2 Check Verify Registration Profile

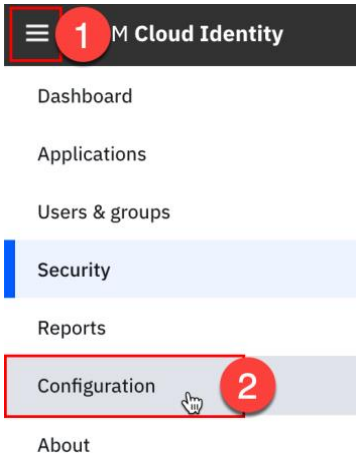
To use the IBM Verify Authentication factor, a Verify Registration Profile is required. A default profile should exist for all new tenants but you will check now.



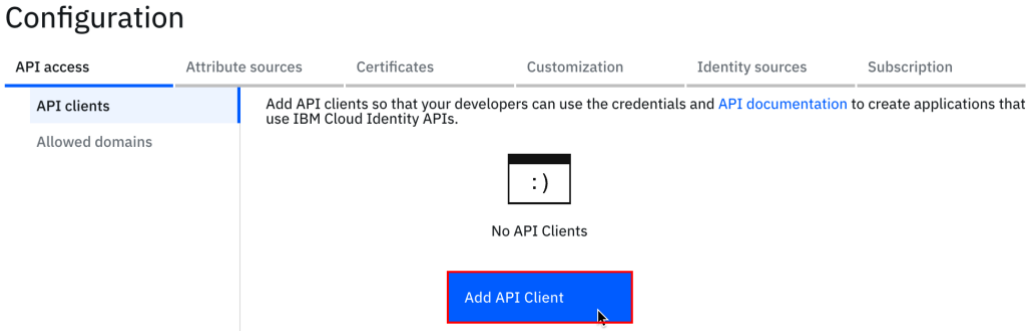
Still on the **Security** screen, select the **Registration profiles** tab and look for the *Verify Profile* profile. If this exists then you are ready to go. If another profile exists you can use this instead (make a note of the profile name). If no profile exists, you will need to add one now.

2.3 Create an API Client

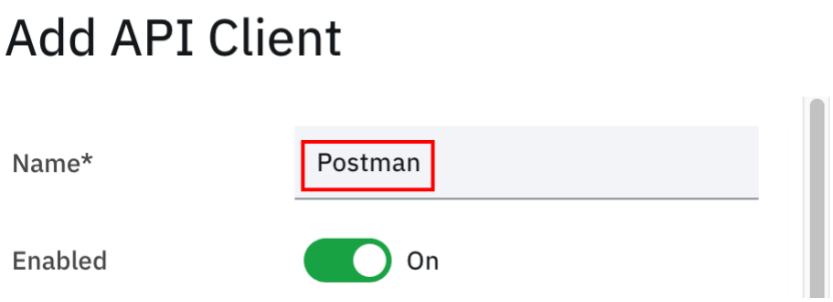
Access to Cloud Identity APIs (including Cloud Identity Verify AaaS APIs) is controlled using OAuth 2.0. Clients must be defined in Cloud Identity in order to receive a Client ID and Client Secret which they can use to obtain an OAuth Access Token. This Access Token is then presented with all API requests to authenticate and authorize them. Authorization is based on granted permissions in the client definition.



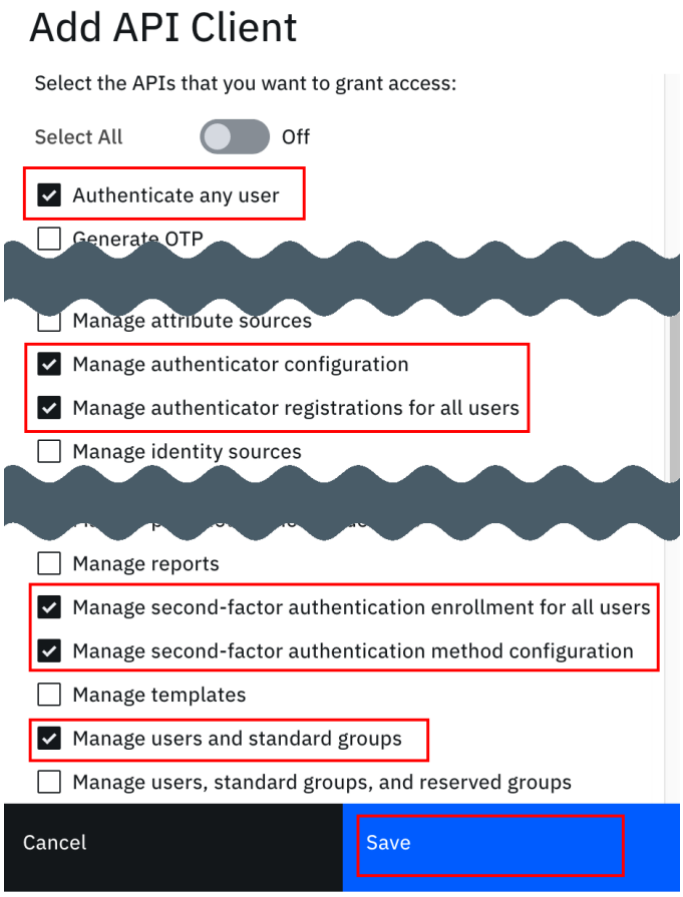
Open the menu using the burger icon in the upper-left corner and select **Configuration**.



API access is the default tab. Click **Add API Client** to add a new client.



Give your client a name.

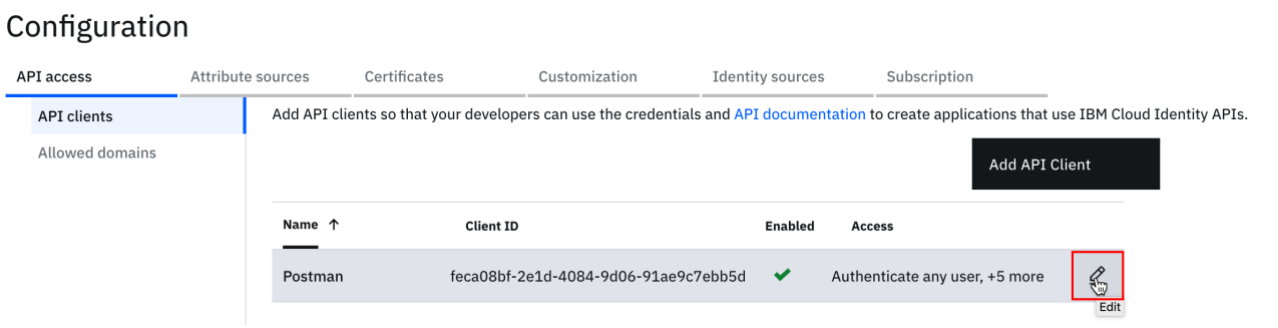


Enable the following permissions for the API Client:

- Authenticate any user
- Manage authenticator configuration
- Manage authenticator registrations for all users
- Manage second-factor authentication enrollment for all users
- Manage second-factor authentication method configuration
- Manage users and standard groups

Click **Save**.

> *Manage authenticator configuration, Manage second-factor authentication method configuration* and *Manage users and standard groups* are not strictly needed for an AaaS client but they allow some additional functions that are available in the Postman Collection and used in this cookbook.



Click the **Edit** icon for the new client. This opens the properties so you can get the Client ID and Client Secret needed to configure Postman.

Leave this window open. You will need it in the next section.

3 Set up Postman

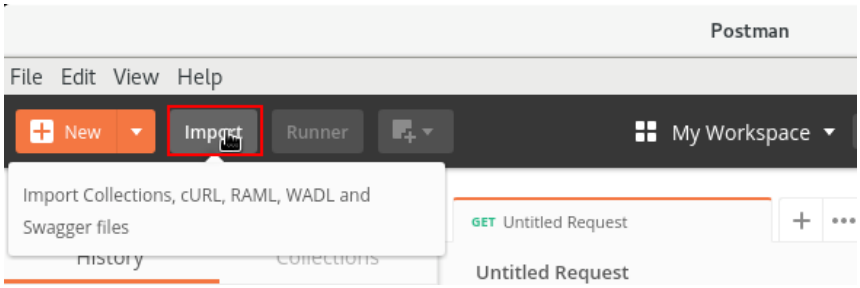
3.1 Import Collection and Environment to Postman

The Postman application provides the ability to import API Collections and Environments. You will now download files provided for this cookbook and import them into your Postman application.

In a browser, navigate to the following URL: **https://ibm.biz/civpostman**

This will open a folder in Box. Download the **CloudId-Demo-Template.postman_environment.json** and **CloudId-Demo.postman_collection.json** files to your test system.

Open the **Postman** application.



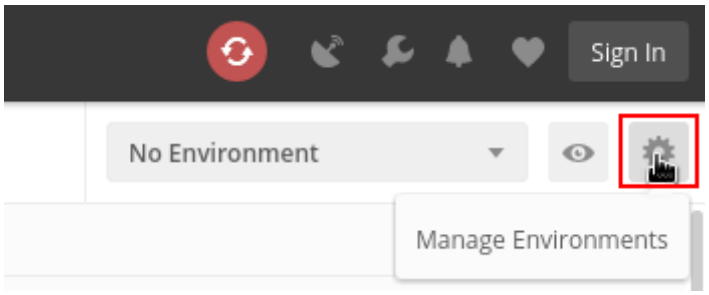
Click the **Import** button on the Postman toolbar. An import overlay is displayed.

Click **Choose Files** and select the file:
CloudId-Demo-Template.postman_environment.json which you just downloaded.

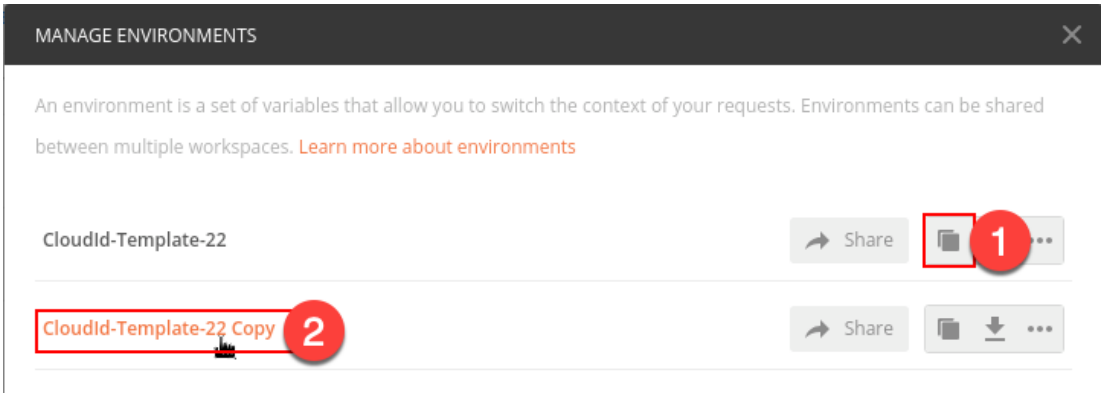
Click the **Import** and **Choose Files** again but this time select the file:
CloudId-Demo.postman_collection.json

3.2 Set up Environment

You will now create a new environment from the template you just imported and configure it for your Cloud Identity tenant.

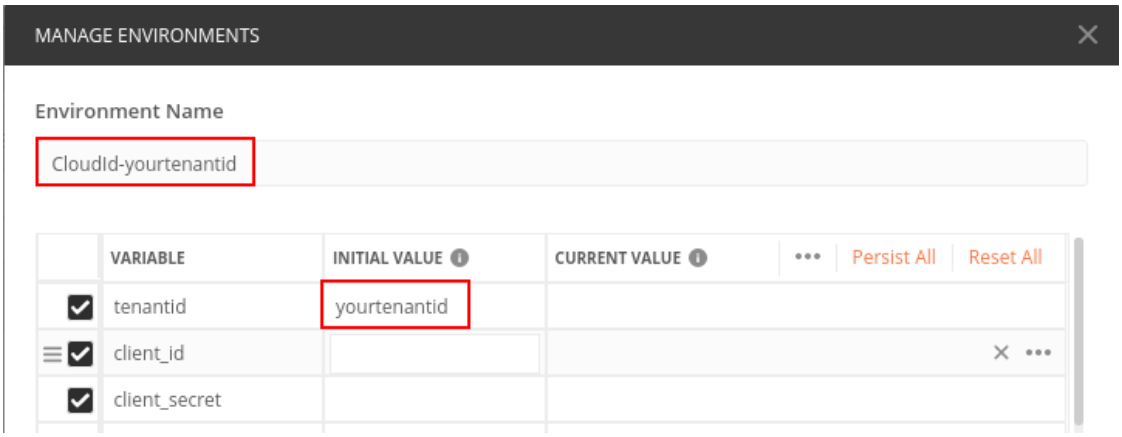


Click the **cog** icon to *Manage Environments*.



Click the **Duplicate** button associated with the environment template you just imported.

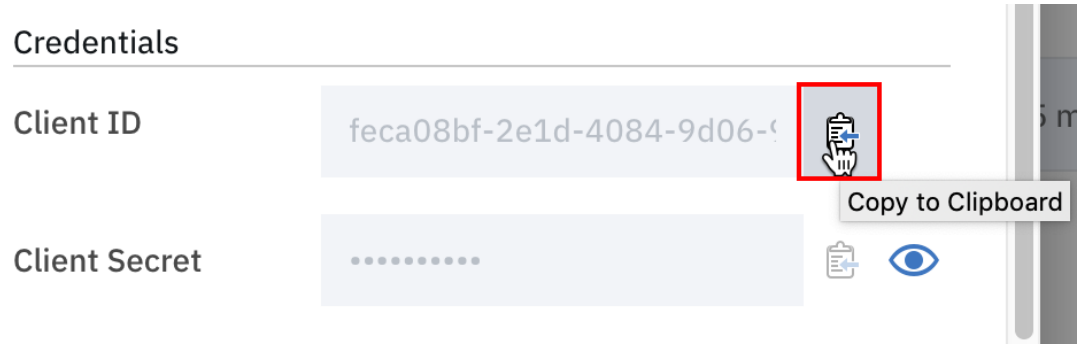
Click the name of the copy that is created.



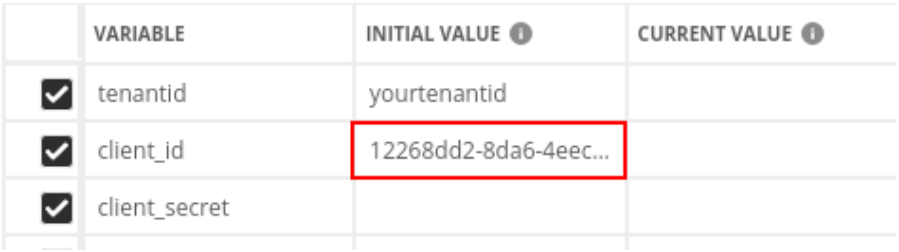
Change the *Environment Name* to a name of your choice.

Enter your Tenant ID in the *Initial Value* column for the *tenantid* variable.

You now need to complete the *client_id* and *client_secret*. These are available in the Cloud Identity API Client settings. Switch to the browser window where your Cloud Identity admin console is running:

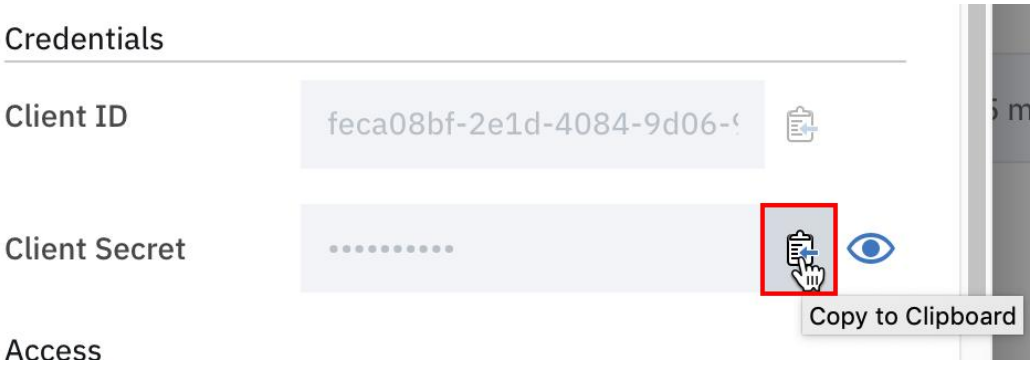


Click the **Copy** button next to *Client ID* to place it on the clipboard. Then switch back to the Postman application:



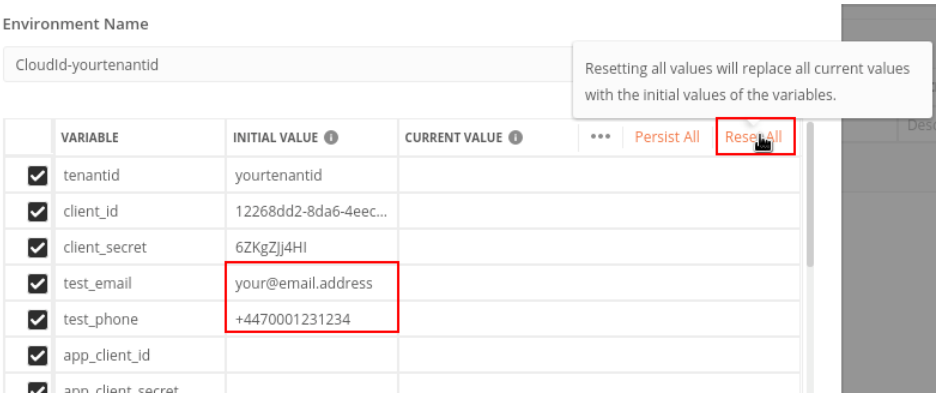
Paste the value into the *Initial Value* column for the *client_id* variable.

Switch back to the browser running the Cloud Identity administration console:



Click the **Copy** button next to *Client Secret* to place it on the clipboard. Then switch back to the Postman application:

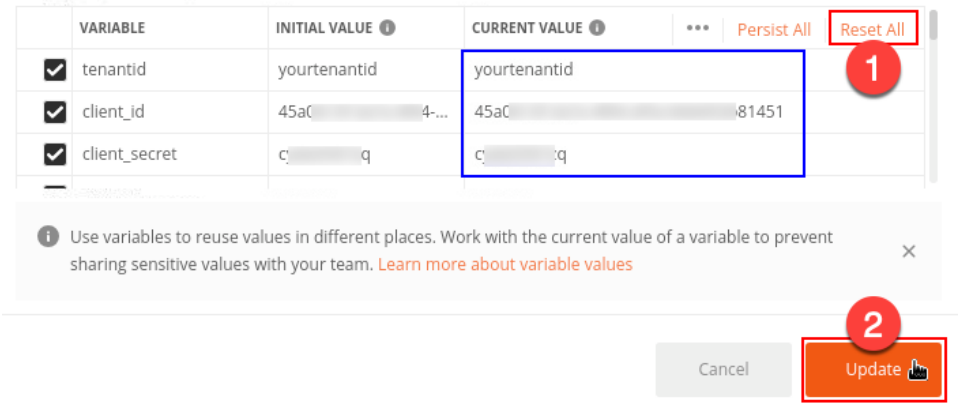
Paste the client secret into the *Initial Value* column for the *client_secret* variable.



Enter an e-mail address that you have access to in the *Initial Value* column for the *test_email* variable. You must be able to receive the e-mails sent to this address.

Enter a phone number that can receive SMS messages in the *Initial Value* column for the *test_phone* variable. You much have access to the device. The number format must include the country code – even if it is a US number.

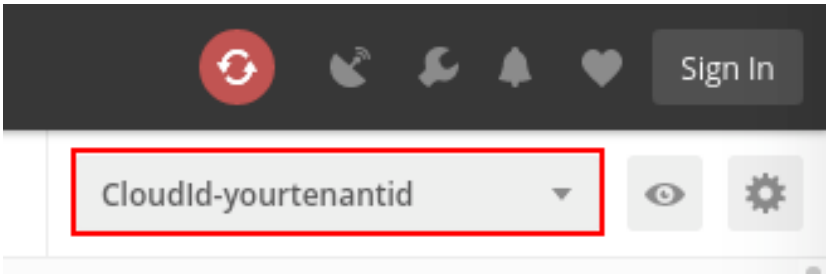
If you are using a Verify Registration other than *Verify Profile* then you will also need to update the *authclient_name* variable here.



Scroll to the top of the variable list and click **Reset All** to copy the values from the *Initial Values* column to the *Current Value* column.

Click **Update** to save the changes to the environment.

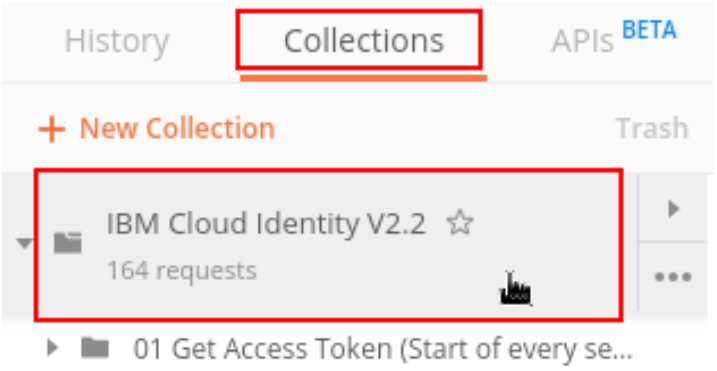
Use the **X** to close the *Manage Environments* overlay.



Select the environment you just created using the drop-down list.

3.3 Select Collection

You need to open the Cloud Identity collection you just imported.



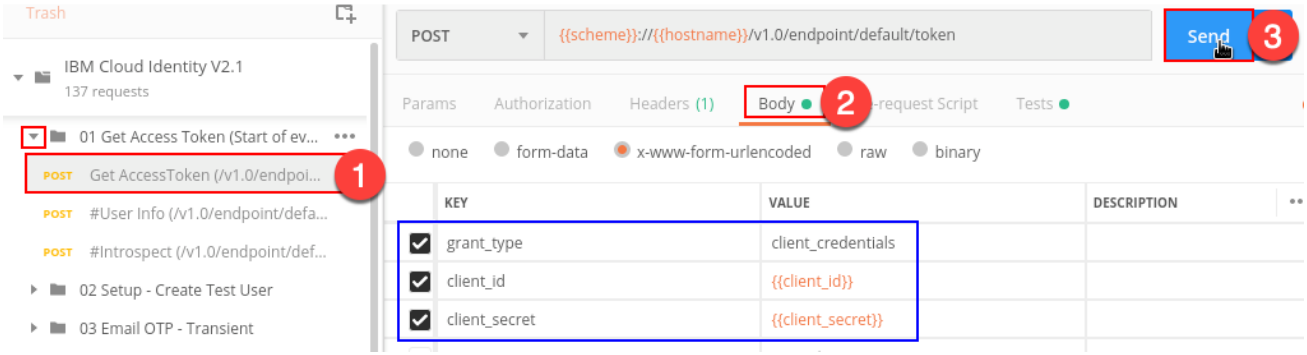
Select **Collections** on the left-hand panel and then click the **IBM Cloud Identity V2.2** collection to open it. If you don't see collections listed, try making the Postman window wider.

Postman is now configured and ready to use with your Cloud Identity Tenant

4 Get Access Token

The first thing that any IBM Cloud Identity API Client must do is to obtain an OAuth Access Token. This is required to get access to any of the REST endpoints.

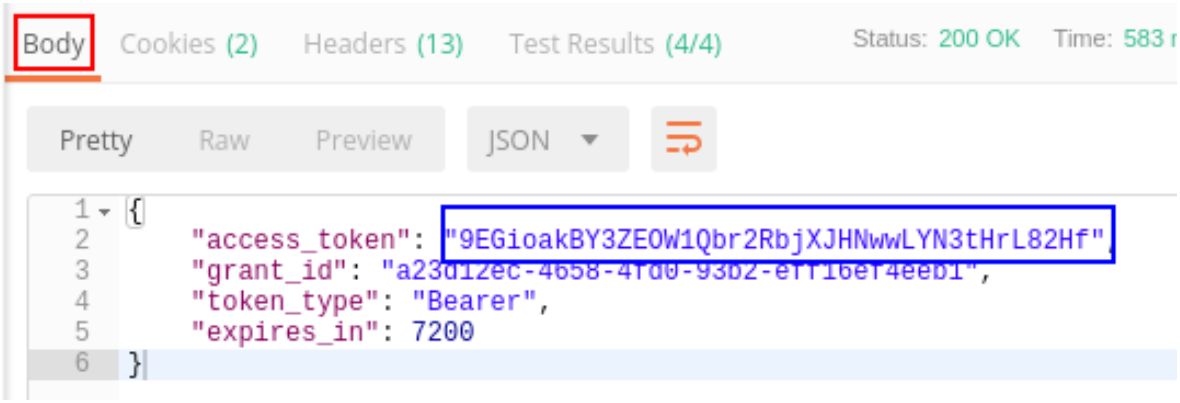
IBM Cloud Identity supports the *client_credentials* OAuth grant flow which allows a client with a registered *Client ID* and *Client Secret* to request an Access Token. You will now execute this flow in Postman.



In the Postman application, expand the **01 Get Access Token** folder and select the **Get AccessToken** request.

Select the **Body** tab and note the *grant_type*, *client_id*, and *client_secret* attributes being sent in the body of the POST to the Cloud Identity OAuth Token endpoint.

Click **Send** to send the request. The response is loaded below the request in the Postman window:



You can see the *access_token* received in the response from the Token endpoint. The test script associated with this request stores this as a new variable in the environment for use in later requests.

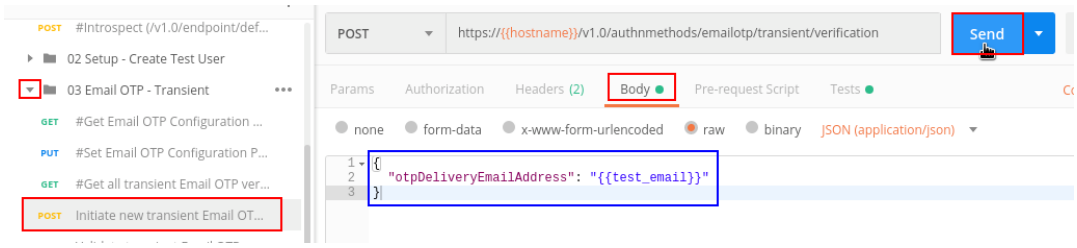
5 Transient One Time Password flows

IBM Cloud Identity Verify supports simple One Time Password (OTP) flows where the client supplies the delivery address for the One Time Password. These are known as *transient* flows because they don't require any persistent user data to be stored. No user entry is needed in the Cloud Identity Cloud Directory.

5.1 Email OTP

5.1.1 Initiate transaction

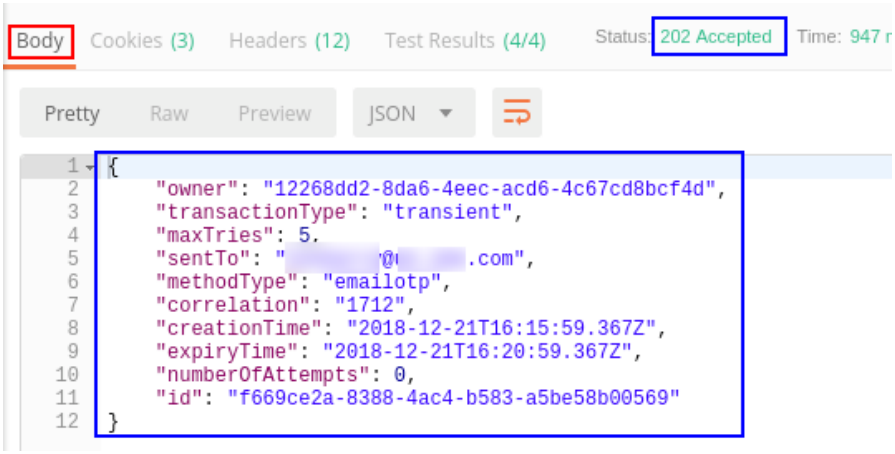
The first authentication transaction you will run is a transient e-mail One Time Password flow.



Expand the **03 Email OTP – Transient** folder and select the **Initiate new transient Email OTP** request.

Select the **Body** tab and notice that the only attribute being sent in the POST request is the delivery e-mail address. This is being taken from the *test_email* variable in the environment.

Click **Send**. The response is loaded below the request in the Postman window:



The status code of **202** is returned. This indicates that the request was accepted. An e-mail containing a random OTP will be sent to the specified e-mail address.

At this point the application would show the user a challenge page telling them to check their e-mail and enter the received OTP into a form.

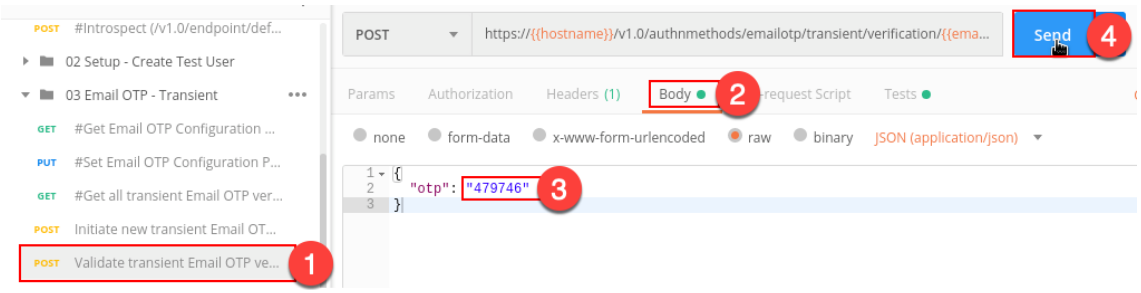
In the response body, you can see the *correlation id*. This will be sent in the e-mail to the user along with the OTP code. The correlation ID is usually shown to the user on the challenge page so they can match the received OTP with the challenge.

The *id* in the response body identifies the transaction and must be sent in the validation request. This ID has been stored in the *emailotp_txnid* variable.

You now need to wait for the e-mail to arrive.

5.1.2 Validate transaction

When the end user receives the One Time Password message, they will provide it to the application. The application now needs to check whether the OTP is correct. This is done using a validation request.



Select the **Validate transient Email OTP verification** request and select the **Body** tab.

Edit the body to send the OTP code from the e-mail you received. Then click **Send**.

Assuming the OTP is validated successfully, you will receive a *200 OK* response. The application can now continue with knowledge that 2FA is complete.

5.1.3 Manage Transactions

At this point the E-mail OTP transaction is complete but it is still held in Cloud Identity Verify. You can view the specific transaction by making a **Get transient Email OTP verification txn by id** request. If you want to clean out the completed transaction, you can make the **Delete transient Email OTP verification txn** request. There is also a **Get all transient Email OTP verification txns** request which returns all transactions (both pending and complete).

5.2 SMS OTP

5.2.1 Initiate transaction

The next authentication transaction you will run is a transient SMS One Time Password flow.

Expand the **04 SMS OTP – Transient** folder and select the **Initiate new transient SMS OTP** request.

Select the **Body** tab and notice that the only attribute being sent in the POST request is the delivery SMS number. This is being taken from the *test_phone* variable in the environment.

Click **Send**. The response is loaded below the request in the Postman window.

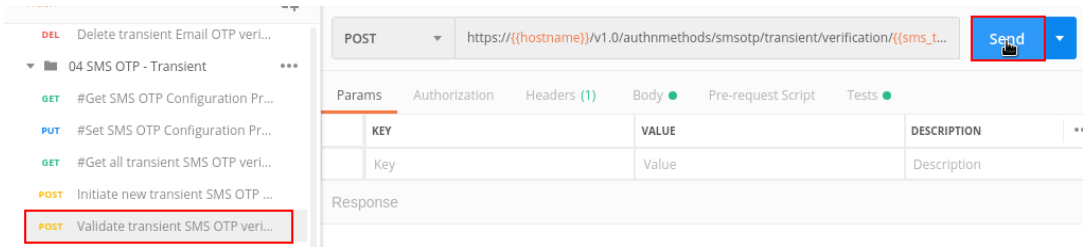
The status code of **202** is returned. This indicates that the request was accepted. An SMS containing a random OTP will be sent to the specified phone number. As before, you will see the correlation ID and the transaction ID (which is stored in the environment).

At this point the application would show the user a challenge page telling them to check their phone and enter the received OTP into a form.

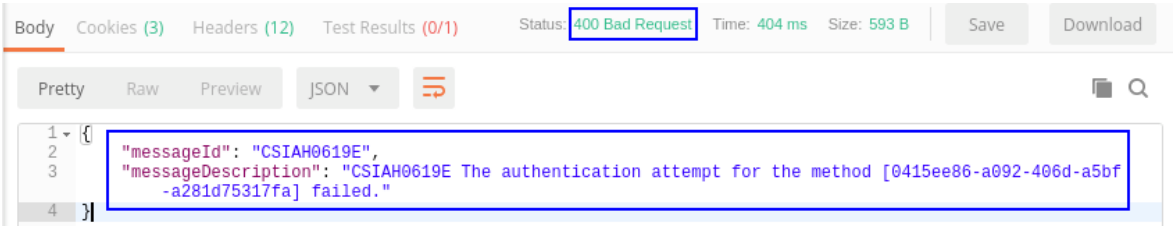
You now need to wait for the SMS to arrive.

5.2.2 Submit incorrect code

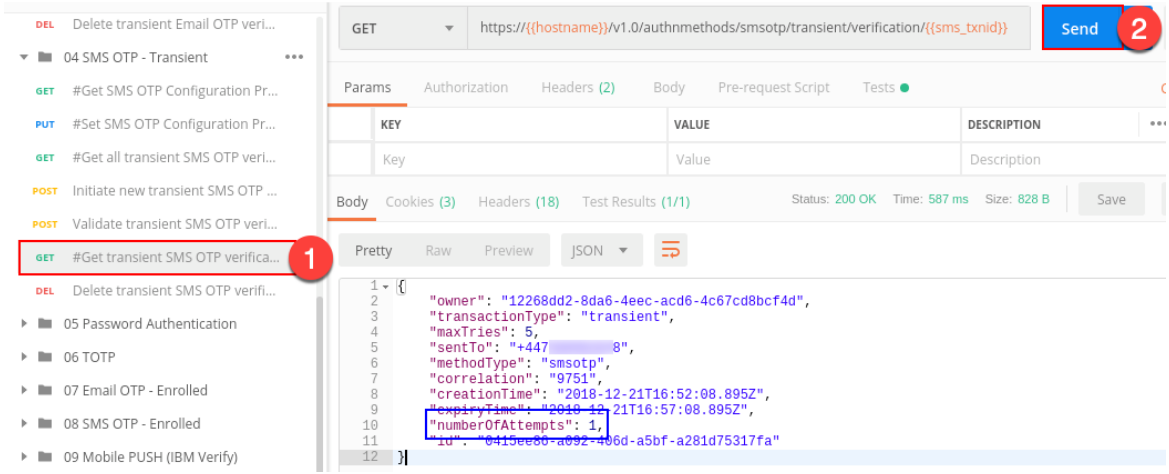
Let's see what happens if an incorrect code is entered.



Select the **Validate transient SMS OTP verification** request and click **Send**. This will send an incorrect code to Cloud Identity Verify. The response is shown:



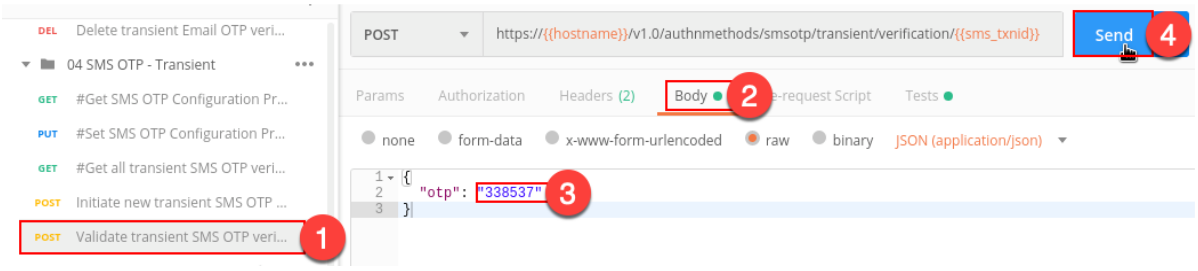
The failed verification results in a *400 Bad Request* response and the body of the message confirms that the authentication attempt failed. You can get more detail by requesting the transaction:



Select the **Get transient SMS OTP verification by id** request and click **Send**. The response shows that one attempt has been made but the transaction is not complete.

5.2.3 Validate Transaction

Now you will submit the correct code. Check your phone for the SMS message from Cloud Identity and read the OTP code.



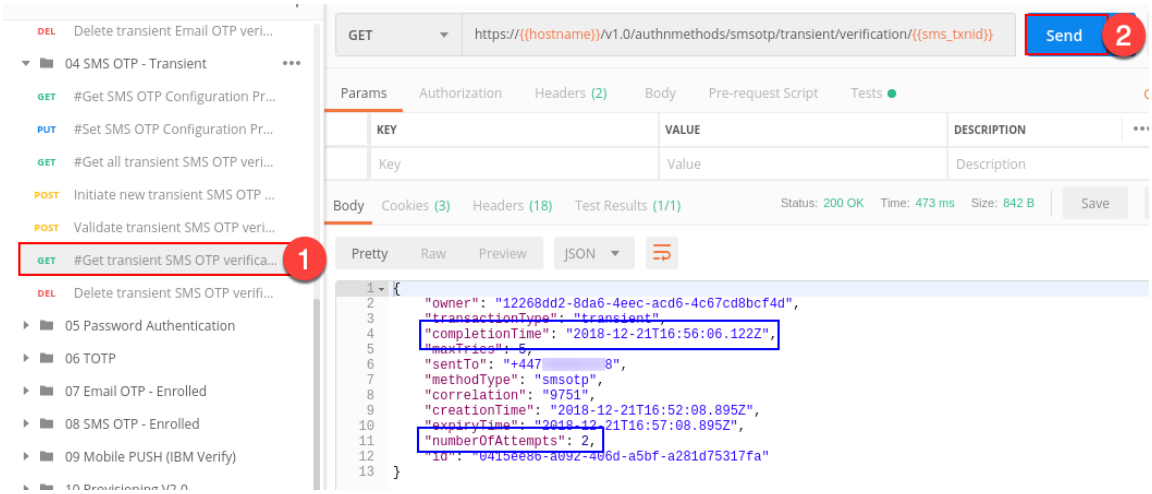
Select the **Validate transient SMS OTP verification** request and select the **Body** tab.

Edit the body to send the OTP code from the SMS you received. Then click **Send**.

Assuming the OTP is validated successfully, you will receive a *200 OK* response. The application can now continue with knowledge that 2FA is complete.

5.2.4 Manage Transactions

At this point the SMS OTP transaction is complete but it is still held in Cloud Identity Verify. You can view the specific transaction by making a **Get transient SMS OTP verification txn by id** request:



This time you can see that there is a completion time shown and number of attempts is was 2.

If you want to clean out the completed transaction, you can make the **Delete transient SMS OTP verification txn** request. There is also a **Get all transient SMS OTP verification txns** request which returns all transactions (both pending and complete).

6 Create a Test User in Cloud Identity Cloud Directory

You will now create a test user in the Cloud Identity Cloud Directory. This needs to be done because the authentication methods covered in the remainder of this cookbook all require a user object for storage of enrollment data.

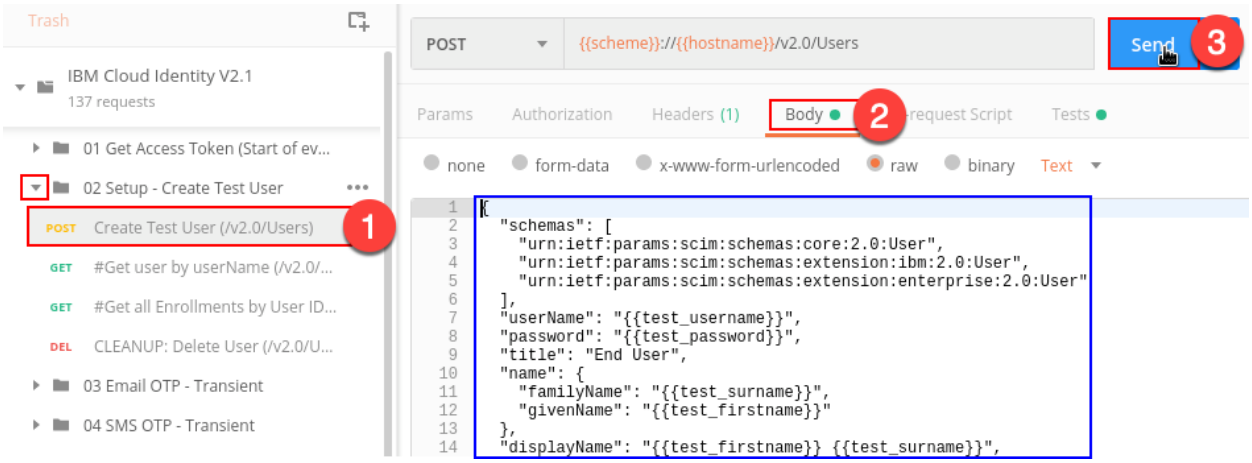
You will create a standard Cloud Directory user but all authentication methods (except password authentication) can also work with federated users. A federated user is a placeholder associated with a user who is asserted from an Identity Source (such as an on-premise IAM system or social login provider).

IBM Cloud Identity supports the SCIM protocol for user and group management. Access to the SCIM endpoint is protected by OAuth 2.0 and authorization is based on the permissions given to the API client. When you created the Postman API client, you gave the *Manage Users and Groups* permission.

6.1 Create User

You will now create a user.

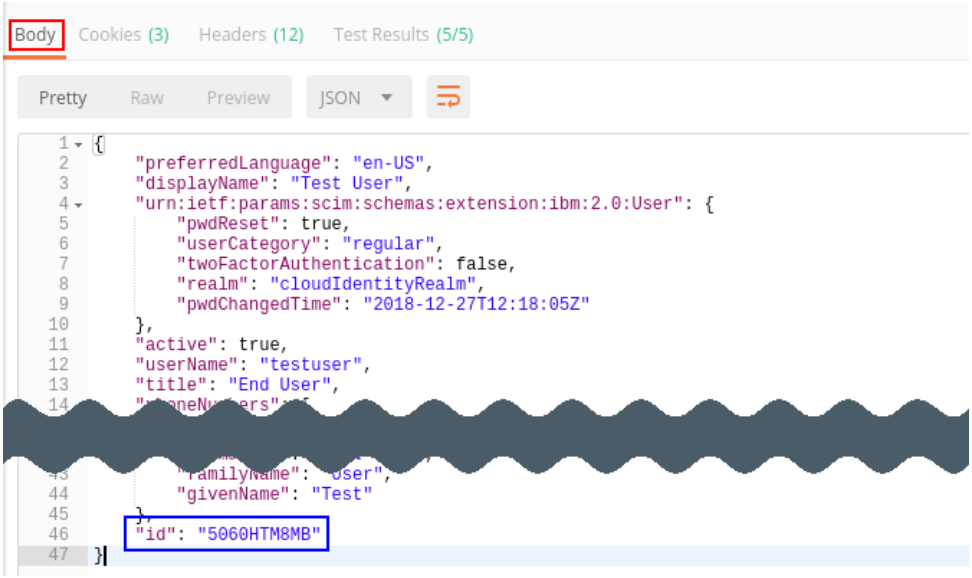
> If you prefer to use an existing user for your tests, rather than creating a new user, you can update the *test_username* and *test_password* variables in the Postman environment and skip this create step. You can also modify the *test_* variables to change the properties of the user that is created here.



In the Postman application, expand the **02 Setup – Create Test User** folder and select the **Create Test User** request.

Select the **Body** tab to examine the body of the request being sent. This is a standard SCIM user creation using standard SCIM schema. The parameters are being set using variables from the Postman environment.

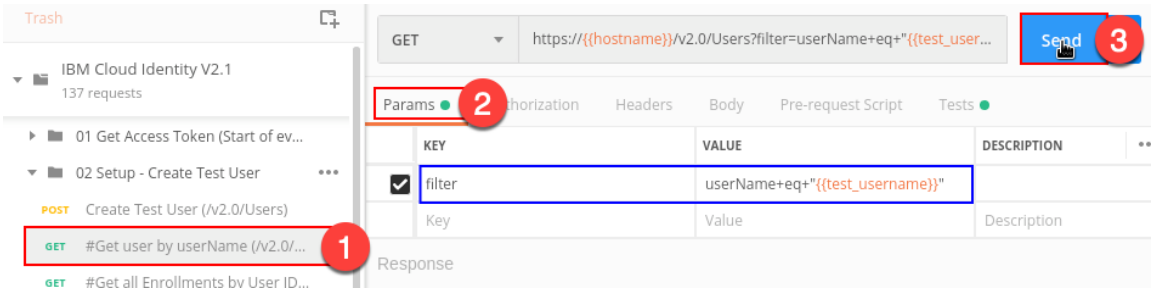
Click **Send**.



In the response to the create request, you receive the full SCIM object for the new user. The most important parameter is the *id*. This is the unique identifier for the user that is used when calling other APIs. This id has been stored in the Postman environment as *test_userid*.

6.2 Get user by username

The Cloud Identity SCIM interface supports user lookup. This is important because if you have a username and you want to find the associated user id, this call can get it for you. You will now use this API to retrieve the user you just created (or lookup the existing user you've configured).



Select the **Get user by userName** request.

Select the **Params** tab and notice the definition of the search filter. This is similar to an LDAP search filter and is performing a lookup based on *userName* SCIM attribute.

Click **Send**. The user is returned and the *test_userid* variable is populated with the unique ID of the user.

6.3 Other Provisioning APIs

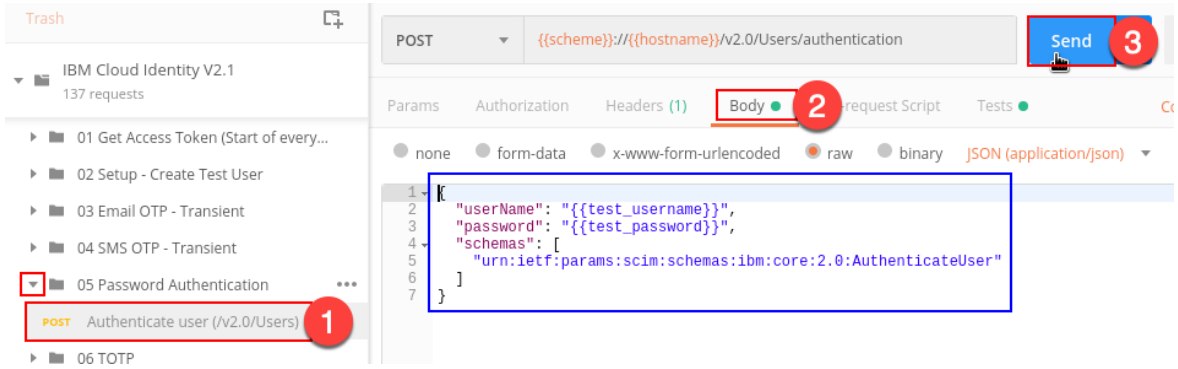
Other provisioning APIs are available in Cloud Identity but are not required for this exploration of Cloud Identity Verify Authentication-as-a-Service. You can find these APIs in the *11 Provisioning 2.0* folder of the Postman collection.

7 Password Authentication

In this section you will see how to perform password authentication for a user defined in the Cloud Identity Cloud Directory. Password authentication is often used as the first authentication factor because it doesn't rely on knowing who the user claims to be in advance (unlike sending a One Time Password for example).

> Password authentication is only available for full Cloud Identity users and users associated with an *LDAP pass-through* Identity Source such as the MaaS360 Cloud Extender. No passwords are stored for federated users and so they cannot complete password authentication.

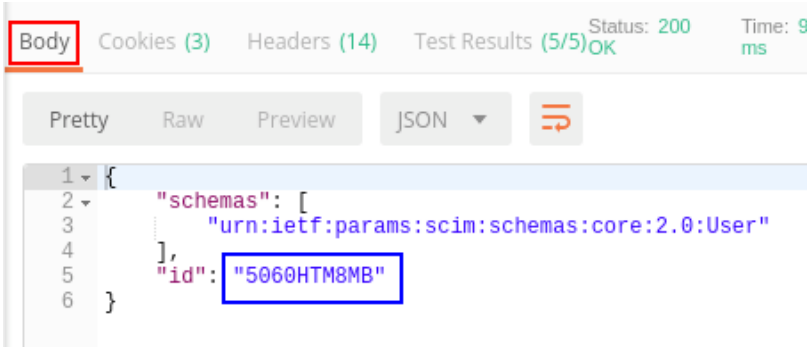
Password Authentication is performed by POSTing to the SCIM User endpoint with a custom schema indicating that password authentication is being performed.



Expand the **05 Password Authentication** folder and select the **Authenticate user** request.

Select the **Body** tab to review the format of the request. You can see that the *username* and *password* attributes are being populated from the Postman environment. Of course, in a real application, the end user would be prompted for these values.

Click **Send** to send the username/password authentication request.



When a correct username and password are provided in the request, a *200 OK* response is returned with the user's unique ID returned in the SCIM response. This ID can be used to lookup the user object via SCIM (to get more information about them) or to lookup enrolled second factor methods for the user.

The ID returned here is stored in the *test_userid* variable in the Postman environment (although it is probably already set from the create and/or lookup commands in the previous section).

The calls you might want to try are:

- 10 Provisioning v2.0→Get user by Id
- 2 Setup – Create Test User→Get all Enrollments by User ID

> You can also get the full user record directly from a successful Authentication call by specifying *returnUserRecord=true* in the query string of the request. This saves an additional call.

8 Time-based One Time Password (TOTP)

In this section you will enroll your test user for Time-based One-Time Password (TOTP) authentication and then test this authentication by calling a validation API.

For this section you will need a TOTP-capable client. The IBM Verify application is recommended:
iOS - https://itunes.apple.com/gb/app/ibm-verify/id1162190392
Android - https://play.google.com/store/apps/details?id=com.ibm.security.verifyapp

Other clients are available; for example, Google Authenticator.

8.1 Enroll TOTP

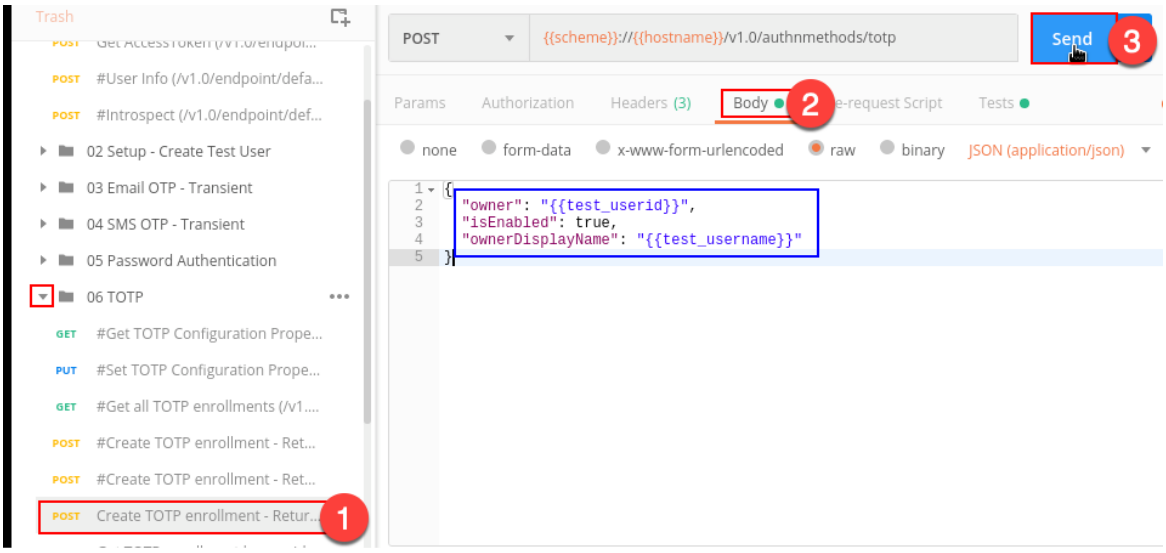
TOTP enrollment is performed by providing the user with a shared secret which is used as the seed for the One Time Password generator. This secret is generated by Cloud Identity during enrollment and associated with the users Cloud Directory entry. It is returned in a standard URI format which can be consumed by TOTP-capable applications.

There are several ways that the API caller can receive the enrollment URI. This is determined by the request and the accept header:

URI	Accept Header	Response
…/totp	application/json	URI Text in JSON
…/totp?qrcodeInResponse=true	application/json	Base-64 encoded QR Code image in JSON
…/totp	image/png	QRCode image in PNG format

You will use the option to request a QRCode image here. This will be displayed by Postman and allows easy registration using a TOTP mobile application.

> Note that the TOTP secret can only be received once. This is to prevent an attacker from registering the same TOTP secret on their own device at a later time (without the knowledge of the legitimate user). If you attempt the create enrollment call again you will receive an error.



Expand the **06 TOTP** folder and select the **Create TOTP enrollment – Return QR Code image** request.

Select the **Body** tab and review the request parameters. You can see that the *test_userid* is provided as the *owner* of the enrollment and that the enrollment will be enabled. The *ownerDisplayName* specifies how the TOTP registration will be displayed in the client application. It is being set to the test user's username (but this could be changed).

Click **Send**. A QRCode image is returned and is displayed by Postman:

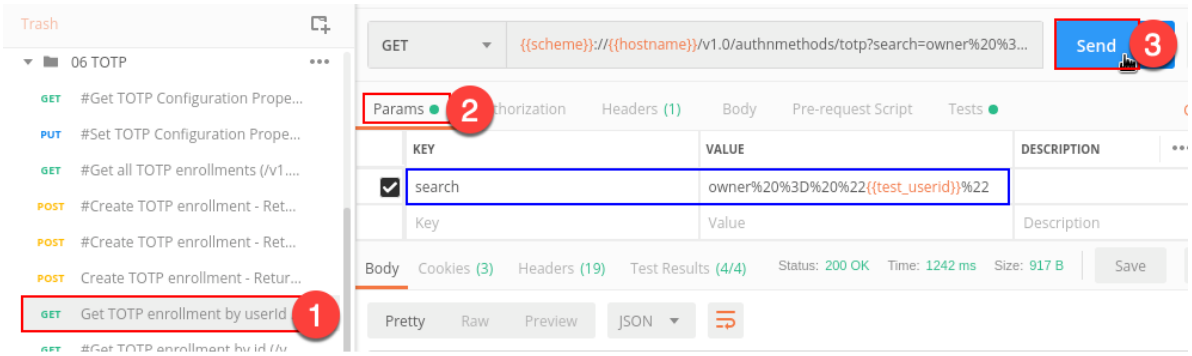
This QR Code contains the following URI:
otpauth://totp/<*domain*>:<*userID*>?secret=<*secret*>&issuer=<*issuer*>&algorithm=<*hash algorithm*>

Open the IBM Verify App (or other TOTP-capable app) on your mobile device and scan the QRCode to register.

8.2 Lookup TOTP Enrollment (to get ID)

In order to verify a TOTP code, the application needs to know the Enrollment ID for the TOTP enrollment.

If the enrollment step had requested a JSON response, this ID would have been included but, because you requested a QR Code image response, you didn't get the ID. In this case a call can be used to lookup TOTP enrollments for a provided User ID.



Select **Get TOTP enrollment by userId** request. Select **Params** tab to see the search being used to lookup the TOTP enrollments. The search is: *owner = "<userID>"*.

Click **Send**.

The response contains information about the TOTP enrollment including the id:

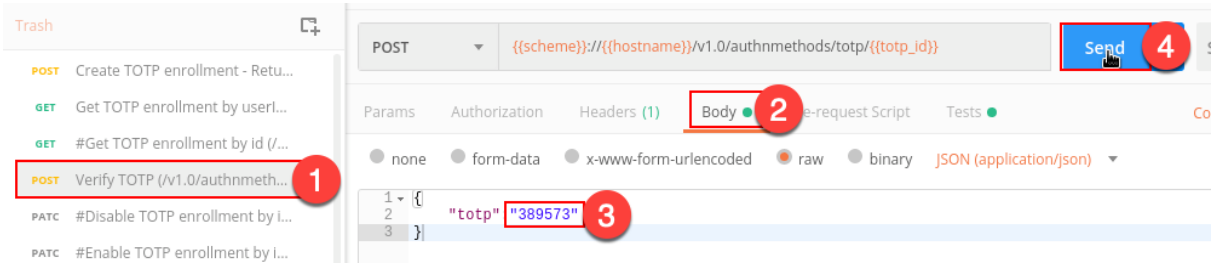
This id is stored in the *totp_id* variable for use in TOTP verification requests for this user.

Also note the *isValidated* flag. This is set to *false* until the first time a TOTP code is successfully verified using this enrollment.

8.3 Verify TOTP Authentication

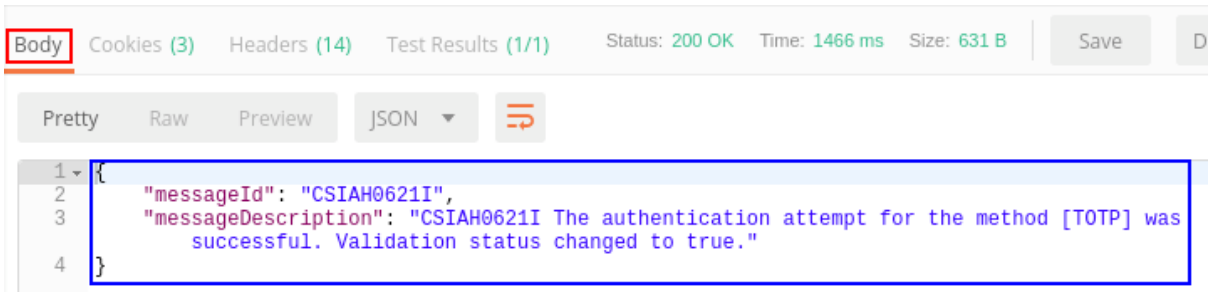
When an application using Cloud Identity Verify wants to perform TOTP authentication, it will present a challenge to the user asking them to provide the current code shown in their TOTP app. When the user provides the code, the application sends the code on to Cloud Identity Verify and requests verification.

You will now perform this call using Postman.



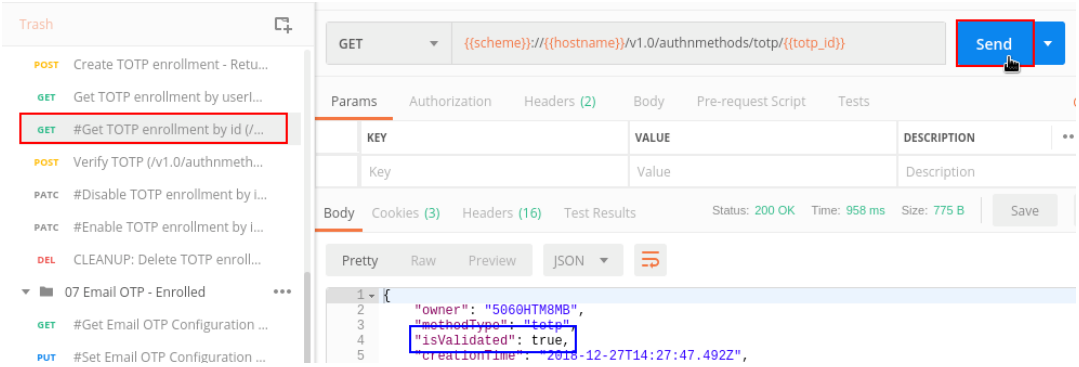
Select the **Verify TOTP** request and then select the **Body** tab.

Open the IBM Verify app on your mobile device and select the test user's account so that you can see the TOTP code displayed. Wait until a new code is displayed (to give maximum time). Then quickly enter the code into the request body in Postman and click **Send**. You need to click Send before the code expires.



If the code is successfully verified, a *200 OK* response is returned. You can see from the message body that the validation status has been changed to true.

If you get the enrollment again (which you can now do by ID rather than searching by username) you will see that *isValidated* is now set to *true*:

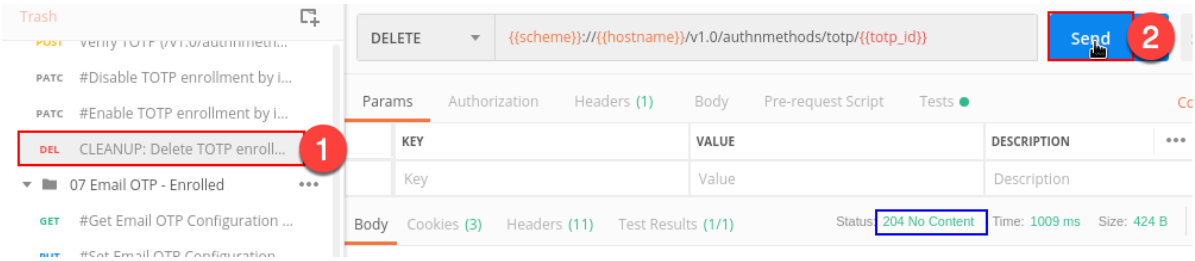


8.4 Manage Enrollments

The Postman collection includes PATCH requests which can be used to disable and enable a TOTP enrollment. This can be useful if a mobile device is thought to be lost. The TOTP enrollment associated with the device can be disabled rather than deleted (allowing to the possibility to re-enable again if found).

You can use these calls to disable your enrollment and see how this affects verification attempts.

It is currently only possible to create a single TOTP enrollment per user. In order to allow TOTP to be enrolled as part of Mobile PUSH authentication later in this guide, you will now delete the TOTP enrollment for the test user.



Select the **CLEANUP: Delete TOTP enrollment** request and click **Send**. You will receive a *204 No Content* response to indicate success.

At this point you can also manually remove the TOTP registration from the IBM Verify mobile app. To do this, select your test user's account, press **Settings**, press **Remove this account,** and press **Remove** to confirm.

9 Enrolled e-mail and SMS One Time Passwords

Earlier in this cookbook you saw how you can send One Time Passwords to arbitrary e-mail addresses or phone numbers without the need for user enrollment. This is useful when user e-mail and phone number information is managed by (or at least is available to) the calling business application.

For cases where the business application does not want to manage e-mail or phone number information, Cloud Identity Verify offers the option to manage this information as e-mail or SMS One Time Password "enrollments".

You will now set up an enrolled e-mail and SMS One Time Passwords. The following two sections are essentially identical; you may choose to only complete one of them.

9.1 Email OTP

9.1.1 Initiate Enrollment

Before a user can use an enrolled e-mail One Time Password authentication, they must register their e-mail address with Cloud Identity Verify. This is done using an enrollment REST call.

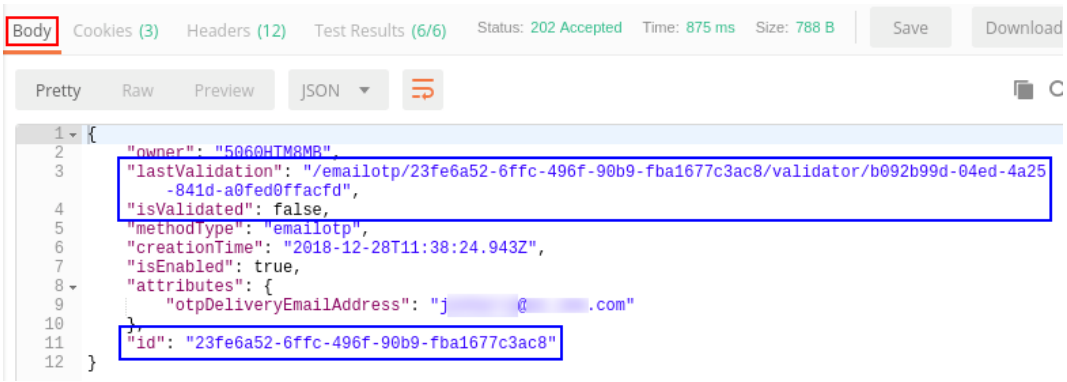
Expand the **07 Email OTP – Enrolled** folder and select the **Create Email OTP enrollment** request.

Select the **Body** tab and review the parameters being sent. The *owner* parameter is set to the unique ID for the user (which you would get from a user search or following a password authentication). The *otpDeliveryEmailAddress* is set to the e-mail address you added to the environment. The *isEnabled* flag is *true* which means this enrollment will be enabled on creation.

Click **Send**.

At this point Cloud Identity Verify sends a validation OTP to the e-mail address provided. This is done to check that the user has access to the e-mail account. Before the new e-mail address can be used for One Time Password verification, the validation must be completed.

The response is shown below the request:

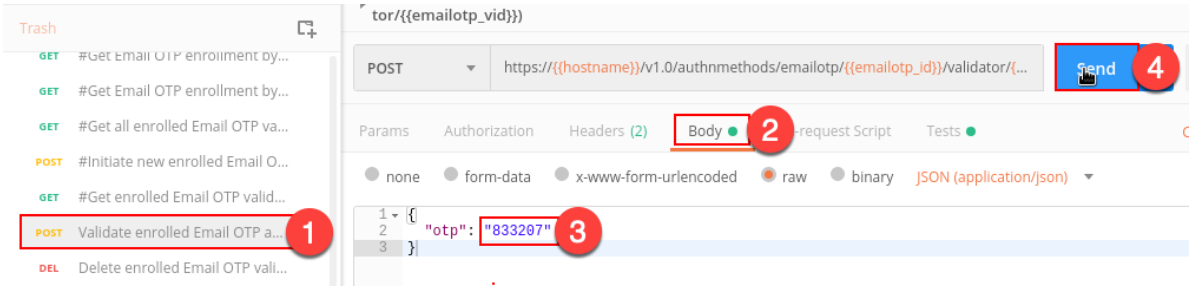


In the response you can see the *id* of the new enrollment. This has been saved to variable *emailotp_id*.

You can also see that the *isValidated* flag is *false*. This indicates that this enrollment has not yet been validated. Finally, you can see the *lastValidation* URL. This is the URL that should be used to validate this One Time Password enrollment.

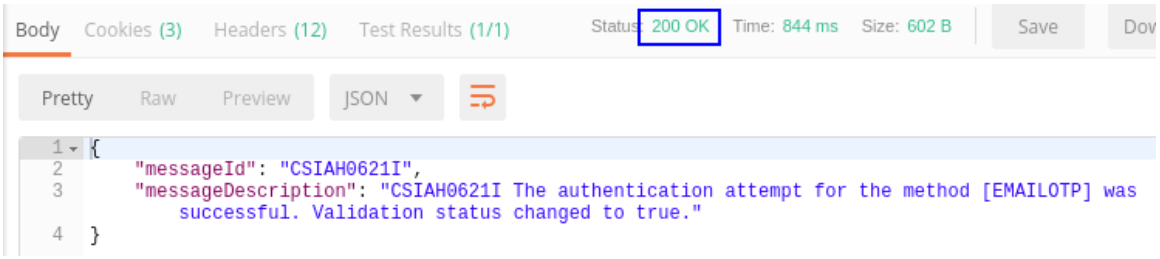
9.2 Validate Enrollment

You should have received a One Time Password to the registered e-mail address. You will now use this to validate the OTP enrollment.



Select the **Validate enrolled Email OTP against validation txn** request and select the **Body** tab.

Check your e-mail and retrieve the OTP code. Enter this in the request body and then click **Send**. Assuming the validation is successful, you will receive a *200 OK* response:



If the validation fails, it may be because you took too long to perform the validation. By default, OTPs are valid for 300 seconds (5 minutes).

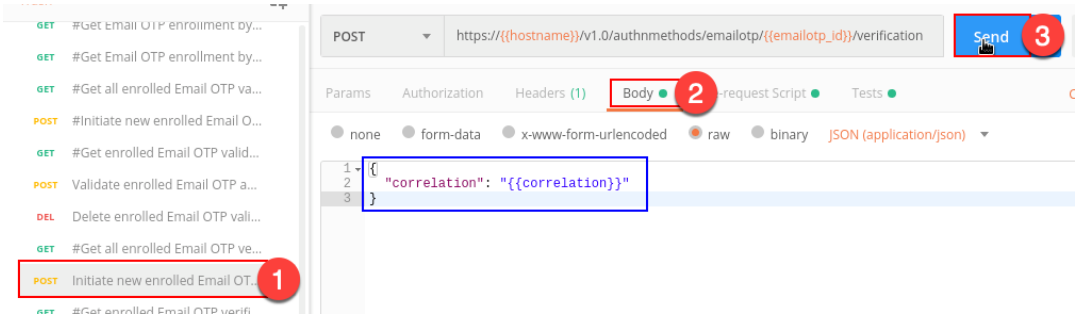
> The e-mail OTP configuration can be modified by using GET and PUT calls against the properties endpoint. See **Get Email OTP Configuration** request. Configuration can modify character set for OTP, OTP length, max retries, OTP lifetime, and whether validation on enroll is required.

Your e-mail OTP enrollment is now complete and can be used for OTP verification transactions. You can use the **Delete enrolled Email OTP validation transaction** to clean up this transaction.

9.2.1 Initiate transaction

You will now run an e-mail One Time Password flow using the enrolled e-mail OTP method.

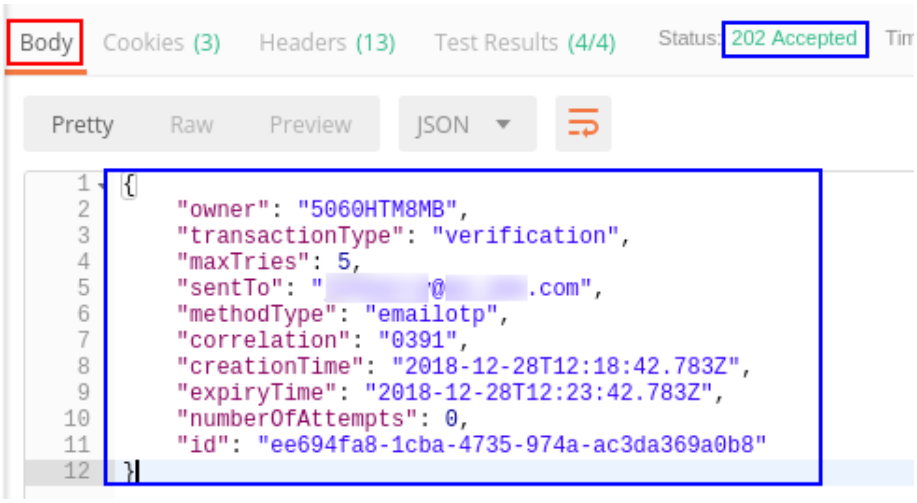
In a normal 2FA situation, the application would first have to look up the available OTP Enrollment(s) for the user and, optionally, ask them to choose one. The **Get Email OTP enrollment by userId** request would be used for this purpose; It stores the ID of the first returned enrollment in the *emailotp_id* variable. You don't need to do that here because the variable is already populated.



Select the **Initiate new enrolled Email OTP verification txn** request.

Select the **Body** tab and notice that no e-mail address is being sent this time. The e-mail address is part of the enrollment. You can see a *correlation* parameter being sent. This isn't required; it just shows that the caller can specify a specific correlation ID if desired. If this wasn't used the request body would have to be just *{ }*.

Click **Send**. The response is loaded below the request in the Postman window:



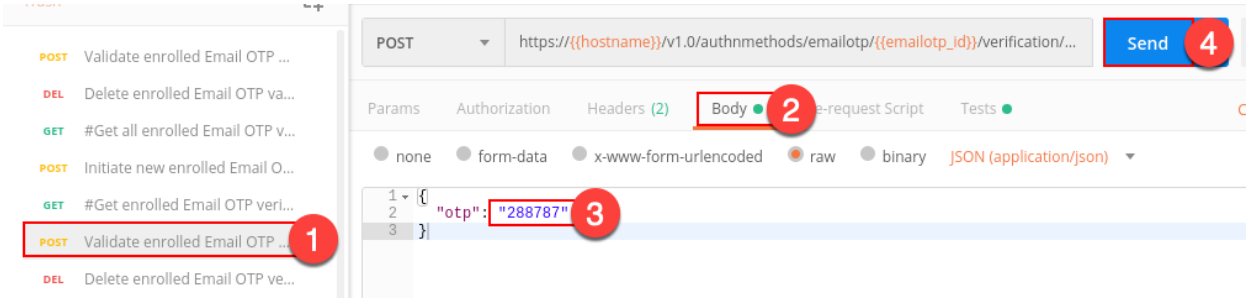
The status code of *202 Accepted* is returned. This indicates that the request was accepted. An e-mail containing a random OTP will be sent to the e-mail address associated with the enrollment. The *id* in this response identifies the transaction and must be sent in the validation request. This ID has been stored in the *emailotp_vid* variable.

At this point the application would show the user a challenge page telling them to check their e-mail and enter the received OTP into a form.

You now need to wait for the e-mail to arrive.

9.2.2 Validate transaction

When the end user receives the One Time Password message, they will provide it to the application. The application now needs to check whether the OTP is correct. This is done using a validation request.



Select the **Validate enrolled Email OTP verification transaction** request and select the **Body** tab.

Edit the body to send the OTP code from the e-mail you received. Then click **Send**.

Assuming the OTP is validated successfully, you will receive a *200 OK* response. The application can now continue with knowledge that 2FA is complete.

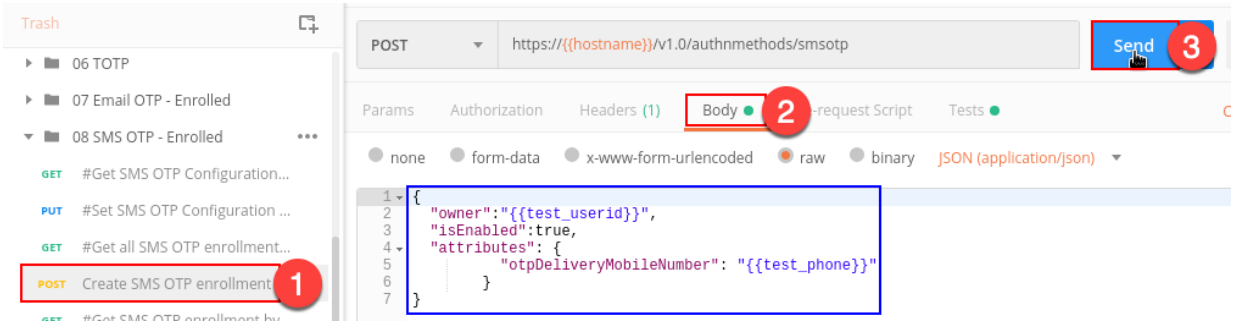
9.2.3 Manage Transactions

At this point the E-mail OTP transaction is complete but it is still held in Cloud Identity Verify. You can view the specific transaction by making a **Get enrolled Email OTP verification txn by id** request. If you want to clean out the completed transaction, you can make the **Delete enrolled Email OTP verification txn by id** request. There is also a **Get all enrolled Email OTP verification txns** request which returns all transactions (both pending and complete).

9.3 SMS OTP

9.3.1 Initiate Enrollment

Before a user can use an enrolled SMS One Time Password authentication, they must register their phone number with Cloud Identity Verify. This is done using an enrollment REST call.



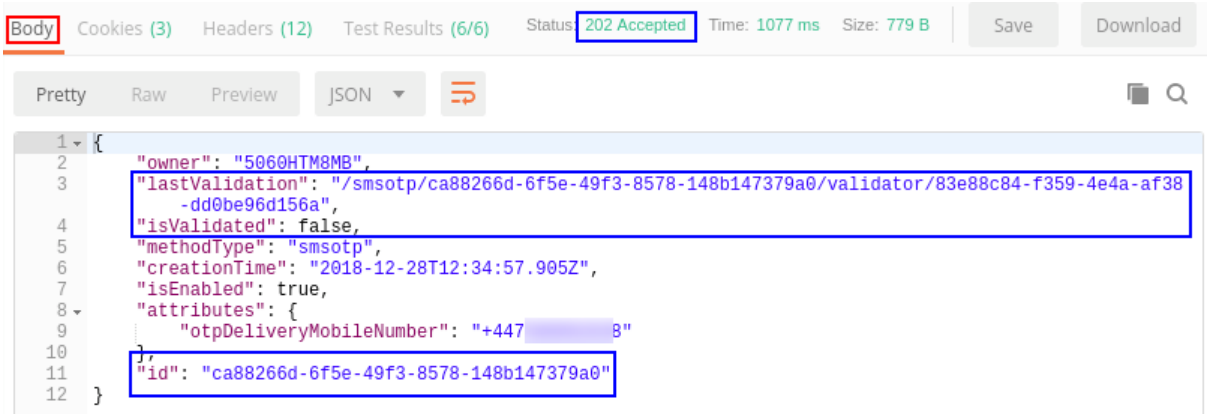
Expand the **08 SMS OTP – Enrolled** folder and select the **Create SMS OTP enrollment** request.

Select the **Body** tab and review the parameters being sent. The *owner* parameter is set to the unique ID for the user (which you would get from a user search or following a password authentication). The *otpDeliveryMobileNumber* is set to the phone number you added to the environment. The *isEnabled* flag is *true* which means this enrollment will be enabled on creation.

Click **Send**.

At this point Cloud Identity Verify sends a validation OTP to the phone number provided. This is done to check that the user has access to the phone. Before the new phone number can be used for One Time Password verification, the validation must be completed.

The response is shown below the request:

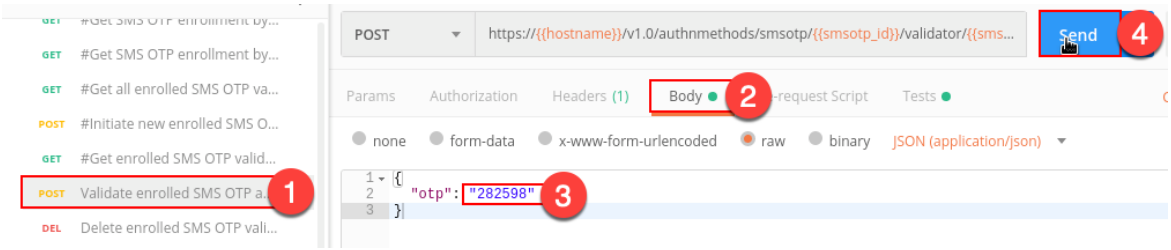


In the response you can see the *id* of the new enrollment. This has been saved to variable sms*otp_id*.

You can also see that the *isValidated* flag is *false*. This indicates that this enrollment has not yet been validated. Finally, you can see the *lastValidation* URL. This is the URL that should be used to validate this One Time Password enrollment.

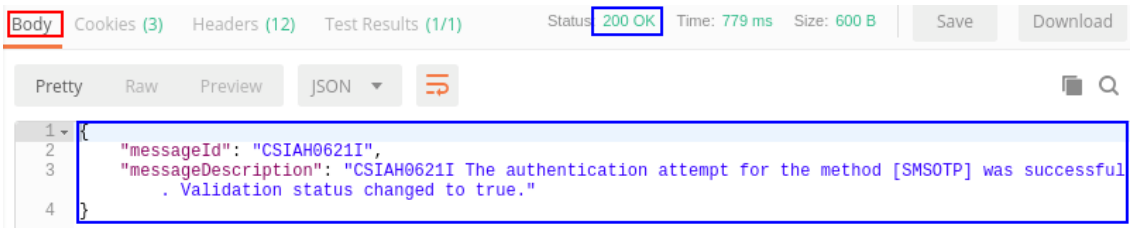
9.4 Validate Enrollment

You should have received a One Time Password via SMS. You will now use this to validate the OTP enrollment.



Select the **Validate enrolled SMS OTP against validation txn** request and select the **Body** tab.

Check your phone and retrieve the OTP code. Enter this in the request body and then click **Send**. Assuming the validation is successful, you will receive a *200 OK* response:



If the validation fails, it may be because you took too long to perform the validation. By default, OTPs are valid for 300 seconds (5 minutes).

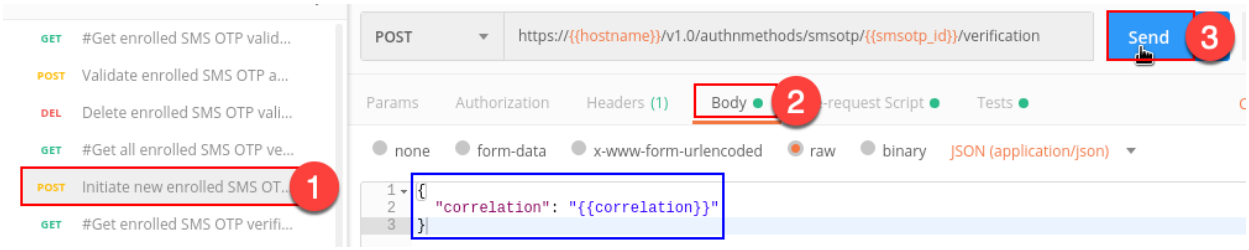
> The SMS OTP configuration can be modified by using GET and PUT calls against the properties endpoint. See **Get SMS OTP Configuration** request. Configuration can modify character set for OTP, OTP length, max retries, OTP lifetime, and whether validation on enroll is required.

Your SMS OTP enrollment is now complete and can be used for OTP verification transactions. You can use the **Delete enrolled SMS OTP validation transaction** to clean up this transaction.

9.4.1 Initiate transaction

You will now run an SMS One Time Password flow using the enrolled SMS OTP method.

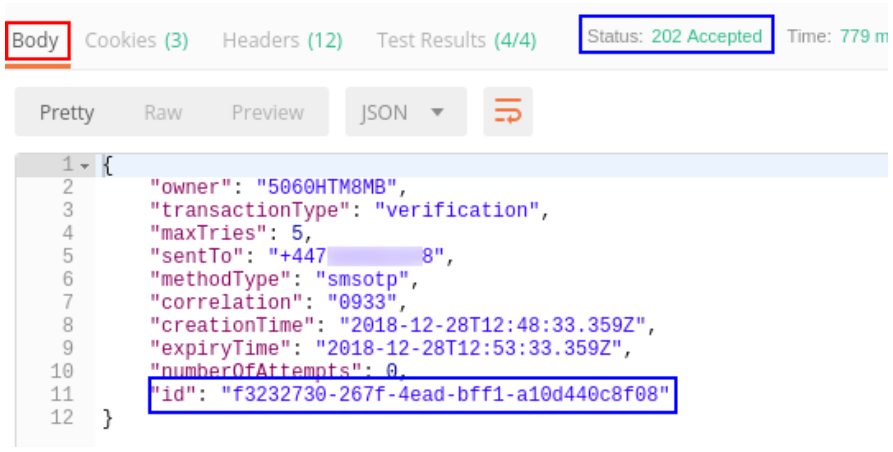
In a normal 2FA situation, the application would first have to look up the available OTP Enrollment(s) for the user and, optionally, ask them to choose one. The **Get SMS OTP enrollment by userId** request would be used for this purpose; It stores the ID of the first returned enrollment in the *smsotp_id* variable. You don't need to do that here because the variable is already populated.



Select the **Initiate new enrolled SMS OTP verification txn** request.

Select the **Body** tab and notice that no phone number is being sent this time. The phone number is part of the enrollment. You can see a *correlation* parameter being sent. This isn't required; it just shows that the caller can specify a specific correlation ID if desired. If this wasn't used the request body would have to be just *{ }*.

Click **Send**. The response is loaded below the request in the Postman window:



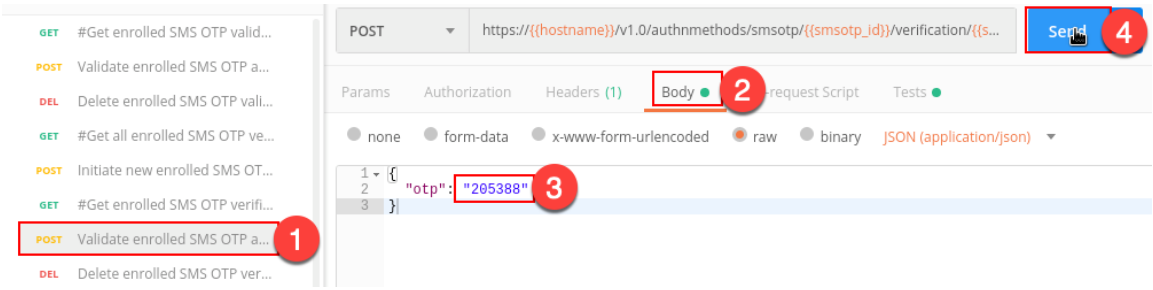
The status code of *202 Accepted* is returned. This indicates that the request was accepted. An SMS containing a random OTP will be sent to the phone number associated with the enrollment. The *id* in this response identifies the transaction and must be sent in the validation request. This ID has been stored in the *smsotp_vid* variable.

At this point the application would show the user a challenge page telling them to check their phone and enter the received OTP into a form.

You now need to wait for the SMS to arrive.

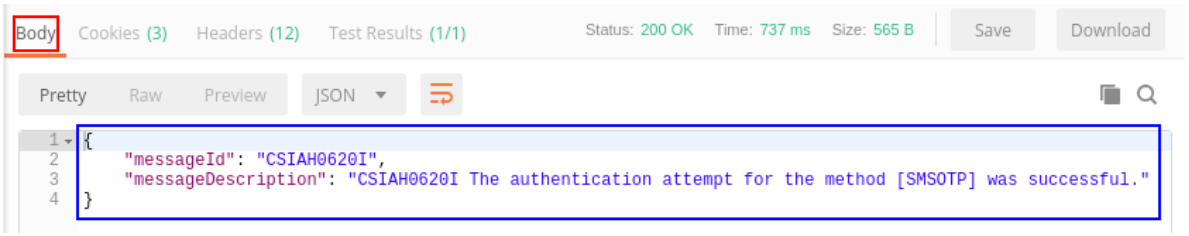
9.4.2 Validate transaction

When the end user receives the One Time Password SMS, they will provide it to the application. The application now needs to check whether the OTP is correct. This is done using a validation request.



Select the **Validate enrolled SMS OTP verification transaction** request and select the **Body** tab.

Edit the body to send the OTP code from the SMS you received. Then click **Send**.



Assuming the OTP is validated successfully, you will receive a *200 OK* response. The application can now continue with knowledge that 2FA is complete.

9.4.3 Manage Transactions

At this point the SMS OTP transaction is complete but it is still held in Cloud Identity Verify. You can view the specific transaction by making a **Get enrolled SMS OTP verification txn by id** request. If you want to clean out the completed transaction, you can make the **Delete enrolled SMS OTP verification txn by id** request. There is also a **Get all enrolled SMS OTP verification txns** request which returns all transactions (both pending and complete).

10 Mobile PUSH with IBM Verify

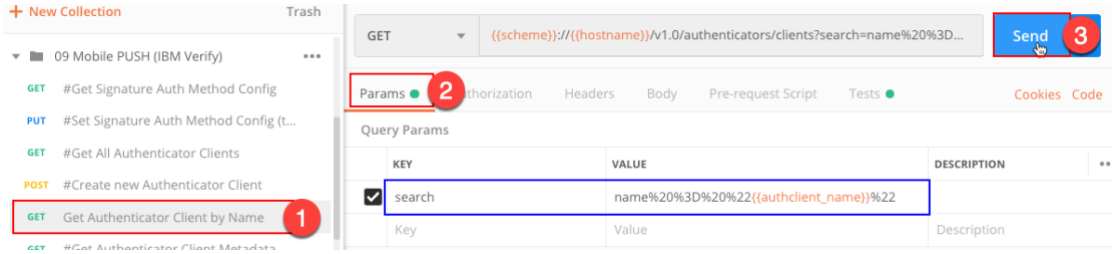
In this section you will enroll your test user for Mobile PUSH transaction verification, using the IBM Verify mobile application, and then test this capability. This functionality can be used for out-of-band transaction verification or for 2nd Factor Authentication.

For this section you will need the IBM Verify client. You can download this for iOS and Android:
iOS - https://itunes.apple.com/gb/app/ibm-verify/id1162190392
Android - https://play.google.com/store/apps/details?id=com.ibm.security.verifyapp

10.1 Look up Authenticator Client ID

A Cloud Identity Verify tenant can support multiple "Registration Profiles" for Mobile PUSH applications. In the APIs a "Registration Profile" is known as an "Authenticator Client". For this exercise you will use the default profile, *Verify Profile.* You will now lookup this Authenticator Client by name to get the ID that is needed to identify the authenticator client during enrollment. It will be stored in the *authclient_id* variable.

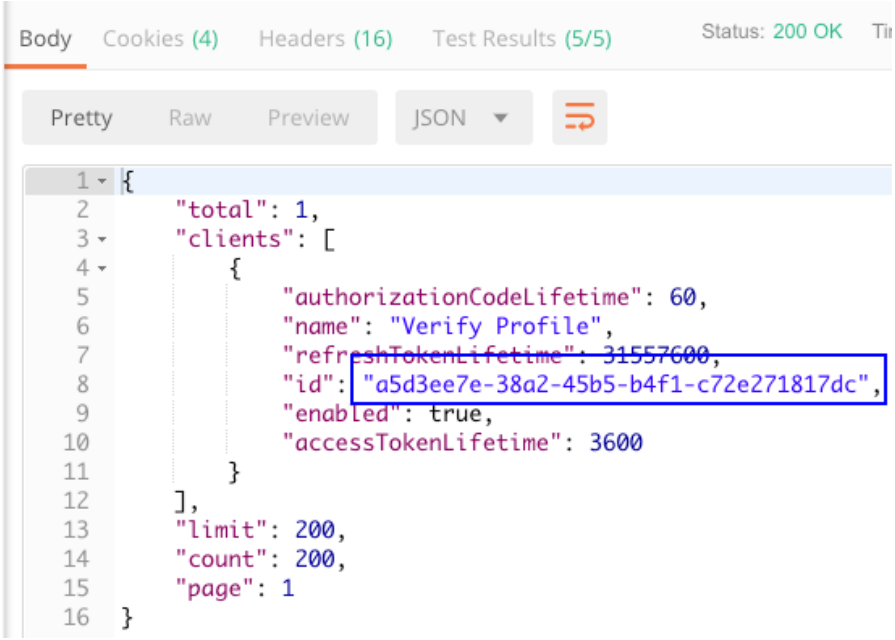
> The "Verify Registration Profile" ID is available directly in the Cloud Identity UI. If you prefer you could skip this lookup step and directly configure the ID as variable *authclient_id* in the Postman environment.



Expand the **09 Mobile PUSH (IBM Verify)** folder and select the **Get Authenticator Client by Name** request.

Select the **Params** tab and note the search filter of *name = "<authclient_name>"* being used for the lookup.

Click **Send**. The response is loaded below the request:



The *id* returned here is loaded to the *authclient_id* variable. Notice the other configuration information shown here. This can be updated in the Cloud Identity Admin UI or using REST calls.

10.2 Enroll Authenticator

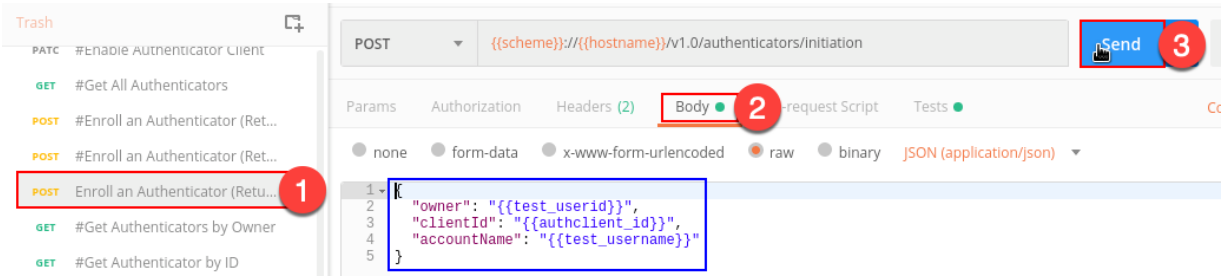
Enrollment of an IBM Verify Authenticator is initiated by calling a REST API on Cloud Identity Verify which starts an OAuth 2.0 Authorization Code flow. The authorization code, together with other enrollment details, are returned to the caller. This information needs to be provided to the authenticator. The IBM Verify App expects to receive this information in the form of a QR Code.

There are several ways that the API caller can receive the enrollment details. This is determined by the request and the accept header:

URI	Accept Header	Response
…/initiation	application/json	URI Text in JSON
…/initiation?qrcodeInResponse=true	application/json	Base-64 encoded QR Code image in JSON
…/initiation	image/png	QRCode image in PNG format

You will use the option to request a QRCode image here. This will be displayed by Postman and allows easy registration using the IBM Verify app.

> Note that TOTP enrollment is completed as part of enrollment of the IBM Verify application. This part will fail if the user already has a TOTP enrollment. If your test user has an active TOTP enrollment you should delete it now.



Select the **Enroll an Authenticator (Return QR code image)** request.

Select the **Body** tab and review the request parameters. You can see that the *test_userid* is provided as the *owner* of the enrollment and *authclient_id* is provided as the *clientId*. The *accountName* specifies how the user will be displayed in the mobile application. It is being set to the test user's username (but this could be changed).

Click **Send**. A QRCode image is returned and is displayed by Postman:

This QR Code contains a JSON object with a number of attributes in it. These include the authorization code, account name, and registration URI.

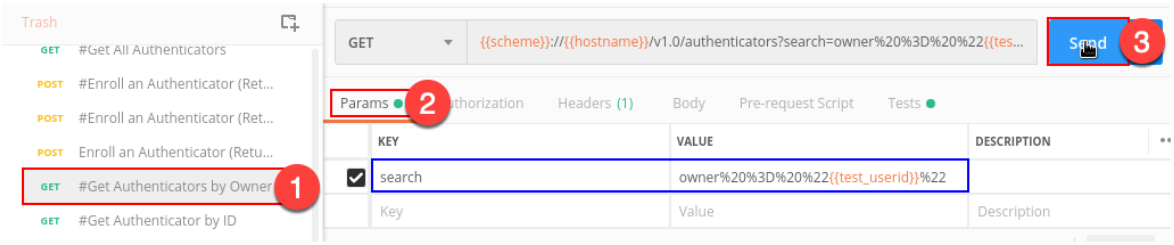
Open the IBM Verify App on your mobile device and scan the QRCode to register. Complete the registration process including registration using Fingerprint/Face ID if available.

At this point an Authenticator has been registered as well as one or more signatures (one for user presence and, optionally, one for fingerprint).

> A TOTP enrollment has also been completed for the user. If you want to, you can look this up and perform validation using the steps in the TOTP section. It is not required for use of Mobile PUSH though.

10.3 Lookup Authenticator (to get ID)

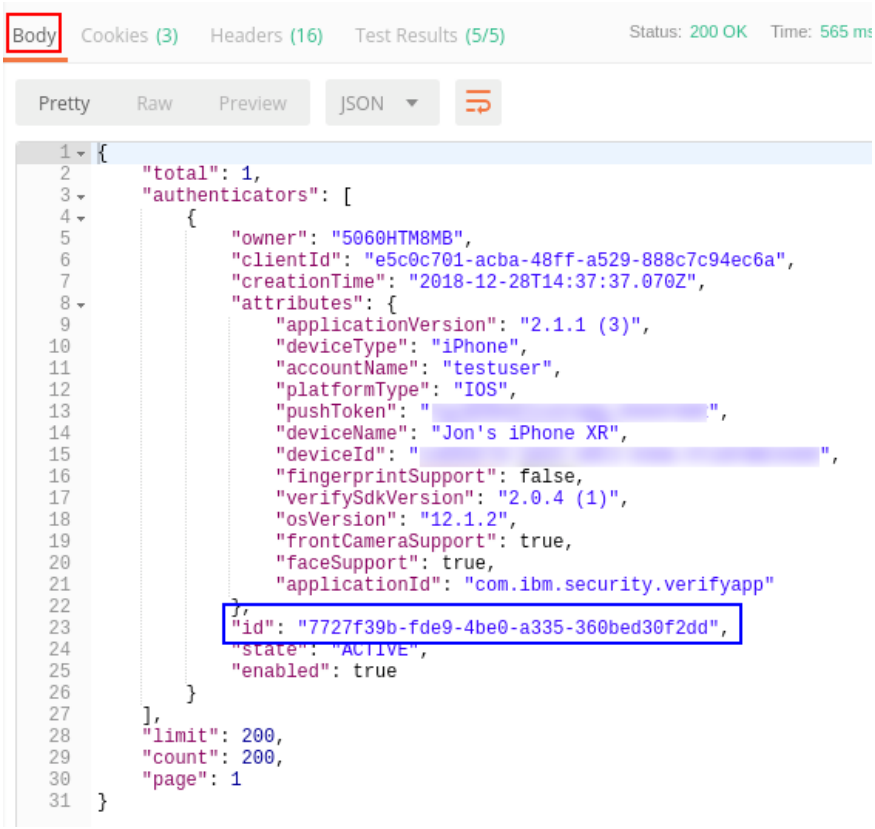
In order to interact with an enrolled authenticator, you need to have the authenticator ID. If the enrollment step had requested a JSON response, this ID would have been included but, because you requested a QR Code image response, you didn't get the ID. In this case a call can be used to lookup authenticator enrollments for a provided User ID.



Select **Get Authenticators by Owner** request. Select **Params** tab to see the search being used to lookup the TOTP enrollments. The search is: *owner = "<userID>"*.

Click **Send**.

The response contains information about the Authenticator including the id:



This id is stored in the *auth_id* variable and is used if you want to delete the authenticator later on.

10.4 Lookup Signature (to get ID)

A Mobile PUSH verification is performed against a *Signature*. The signature identifies the owner, the authenticator and the sub type (userPresence or fingerprint). You now need to look up available signatures, pick the one you want to use, and get its ID so you can initiate a Verify transaction.

Both Touch ID and Face ID are registered as sub type “fingerprint”.

Select the **CHOOSE: Get Signatures by Owner (Return 1st)** request. Select the **Params** tab and note the search being used. Search is *owner = “<User ID>”*. This will return all signatures associated with the selected user.

Click **Send**. The response is loaded below the request:

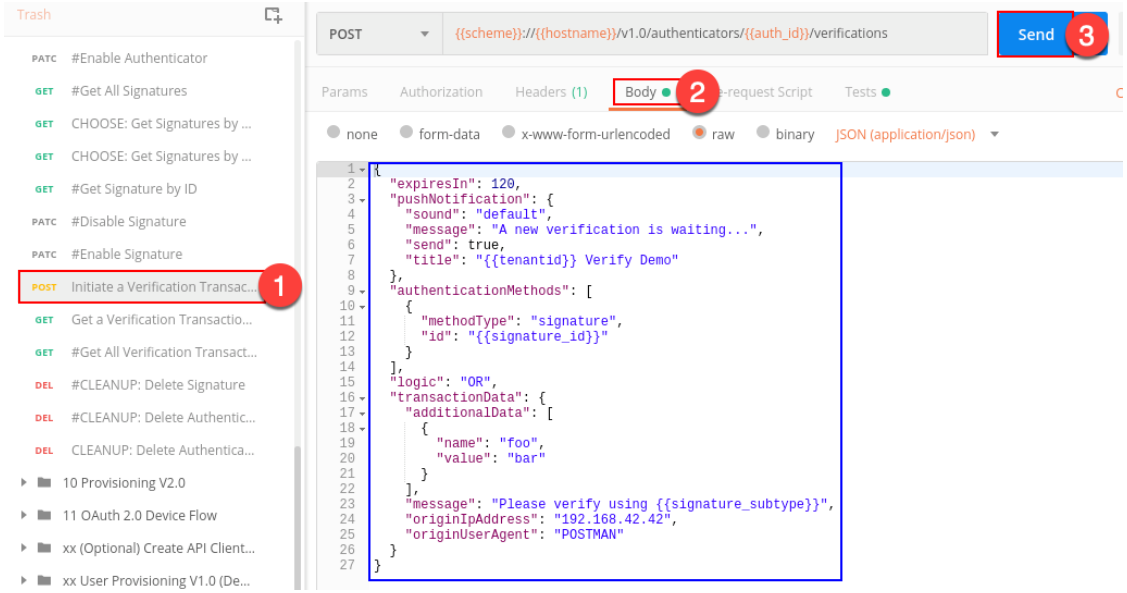
```
{
    "total": 2,
    "limit": 200,
    "count": 200,
    "page": 1,
    "signatures": [
        {
            "owner": "5060HTM8MB",
            "enrollmentUri": "https://yourtenantid.ice.ibmcloud.com/v1.0/authnmethods/signatures/293e1503
                -9b69-4f76-88db-85dbcf8d5afe",
            "methodType": "signature",
            "creationTime": "2018-12-28T14:37:39.228Z",
            "subType": "userPresence",
            "attributes": {
                "deviceSecurity": false,
                "authenticatorUri": "https://yourtenantid.ice.ibmcloud.com/v1.0/authenticators/7727f39b
                    -fde9-4be0-a335-360bed30f2dd",
                "authenticatorId": "7727f39b-fde9-4be0-a335-360bed30f2dd",
                "additionalData": [],
                "algorithm": "RSASHA256"
            },
            "id": "293e1503-9b69-4f76-88db-85dbcf8d5afe",
            "enabled": true
        },
        {
            "owner": "5060HTM8MB",
            "enrollmentUri": "https://yourtenantid.ice.ibmcloud.com/v1.0/authnmethods/signatures/ec2fd677
                -1771-4a27-9d42-34d10da4ce74",
            "methodType": "signature",
            "creationTime": "2018-12-28T14:37:46.593Z",
            "subType": "fingerprint",
            "attributes": {
                "deviceSecurity": true,
                "authenticatorUri": "https://yourtenantid.ice.ibmcloud.com/v1.0/authenticators/7727f39b
                    -fde9-4be0-a335-360bed30f2dd",
                "authenticatorId": "7727f39b-fde9-4be0-a335-360bed30f2dd",
                "additionalData": [],
                "algorithm": "RSASHA256"
            },
            "id": "ec2fd677-1771-4a27-9d42-34d10da4ce74",
            "enabled": true
        }
    ]
}
```


The response should include one or two signatures. If there are two, one will have subtype *userPresence* and the other will have subtype *fingerprint*. The ID for each signature is also returned.

The code associated with this call extracts the ID for the first signature retuned and stores it as *signature_id* variable. It also stores the subtype in *signature_subtype* variable.

10.5 Initiate a transaction and verify in IBM Verify App

You can now initiate a Mobile PUSH transaction. This will use the signature ID extracted in the previous step.



Select the **Initiate a Verification Transaction** request and select the **Body** tab.

Review the request parameters being sent. You can see parameters which control the contents of the PUSH notification sent to the device and the contents of the verification message that the user will see when they access the IBM Verify application.

The Signature Method (which identifies the user, device, and type of verification required) is also specified.

Click **Send**. The response is loaded below the request:


```
{
    "owner": "5060HTM8MB",
    "userActions": [],
    "transactionUri": "https://yourtenantid.ice.ibmcloud.com/v1.0/authenticators/7727f39b-fde9-4be0-a335
        -360bed30f2dd/verifications/62eee5d1-1552-4f7b-9ab1-cb7cfdb1007e",
    "creationTime": "2018-12-28T15:38:43.532Z",
    "authenticationMethods": [
        {
            "methodType": "signature",
            "subType": "userPresence",
            "id": "293e1503-9b69-4f76-88db-85dbcf8d5afe",
            "additionalData": []
        }
    ],
    "expiryTime": "2018-12-28T15:40:43.532Z",
    "transactionData": "{\"originIpAddress\":\"192.168.42.42\",\"originUserAgent\":\"POSTMAN\"
        ,\"additionalData\":[{\"name\":\"foo\",\"value\":\"bar\"}],\"message\":\"Please verify using
        userPresence\",\"timestamp\":\"2018-12-28T15:38:43.532Z\"}",
    "authenticatorId": "7727f39b-fde9-4be0-a335-360bed30f2dd",
    "id": "62eee5d1-1552-4f7b-9ab1-cb7cfdb1007e",
    "state": "PENDING",
    "logic": "OR",
    "pushNotification": {
        "sendState": "SENDING",
        "startTime": "2018-12-28T15:38:43.532Z",
        "message": "A new verification is waiting...",
        "send": true,
        "pushToken": "FgLWV0b0TVsGYqWg:D5HXFO0R"
    }
}
```


You can see that the Verify transaction is in **PENDING** state and the push notification is in **SENDING** state. The transaction ID (which is needed to check the state of the transaction) is stored in the *verification_id* variable.

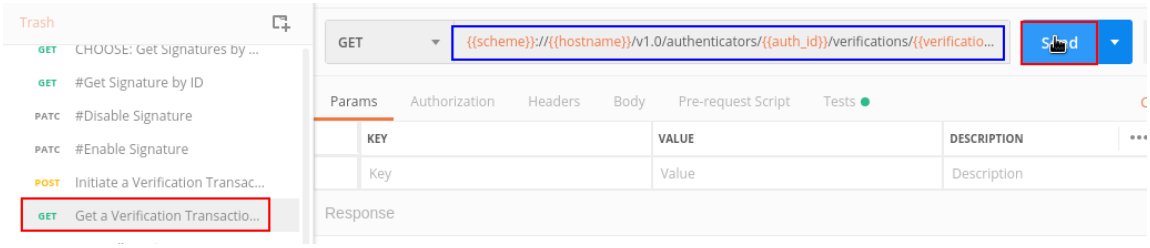
At this point you should receive a notification on your mobile device. If you select this notification, or manually open the IBM Verify application, you should see a prompt to complete a new verification.

Notice that the details sent with the transaction are shown if you swipe up on the verification prompt screen. How these attributes are displayed is up to the mobile application implementation.

Complete the verification in the mobile application.

10.6 Check status of transaction

The only way for the API caller to know if an IBM Verify transaction has been completed by the user is to GET the transaction and check its state. A real application would poll this endpoint waiting for the completion state.



Select the **Get a Verification Transaction by Id** request. Notice that the URL includes both the authenticator ID and the verification ID. Click **Send**.

```
{
    "owner": "5060HTM8MB",
    "creationTime": "2018-12-28T15:38:43.532Z",
    "transactionData": "{\"originIpAddress\":\"192.168.42.42\",\"originUserAgent\":\"POSTMAN\"
        ,\"additionalData\":[{\"name\":\"foo\",\"value\":\"bar\"}],\"message\":\"Please verify using
        userPresence\",\"timestamp\":\"2018-12-28T15:38:43.532Z\"}",
    "userActions": [
        {
            "userAction": "VERIFY_ATTEMPT",
            "signedData": "ANEjBrYfbD3Ob++SgwzXfn6EvtnjXzU4FD34i8Wrea+TmhHNWPbNLgjE
                /5fxQRDsofhUdR5kSx1CtgWyFHBDsNYmVYOZs1dQqVeXNLg8RiREAR3AgwcKdJUWCiWe/tv
                +ppA7xPjSu7n7TJpugQWbAYeHmr0sLKuScFjaZI4GFHH+QMzl6wqDrAgvnaSyZ/hpJwlagql8eMDOP6rDlL20
                /Oue1dKZJ/IiIzHMaANircMfS39HXC4IL1io
                /K40MH4pe8FBLaV4eor2UPZkMUwdlcUXOeTsscuQlOvuq6lzR7fMTQRJICMisgpvsPjUqe4oua1QRBUqjnrldtP
                +ei5AYQ==",
            "id": "293e1503-9b69-4f76-88db-85dbcf8d5afe"
        }
    ],
    "completionTime": "2018-12-28T15:40:28.821Z",
    "transactionUri": "https://yourtenantid.ice.ibmcloud.com/v1.0/authenticators/7727f39b-fde9-4be0-a335
        -360bed30f2dd/verifications/62eee5d1-1552-4f7b-9ab1-cb7cfdb1007e",
    "authenticationMethods": [
        {
            "methodType": "signature",
            "subType": "userPresence",
            "id": "293e1503-9b69-4f76-88db-85dbcf8d5afe",
            "additionalData": []
        }
    ],
    "expiryTime": "2018-12-28T15:40:43.532Z",
    "authenticatorId": "7727f39b-fde9-4be0-a335-360bed30f2dd",
    "id": "62eee5d1-1552-4f7b-9ab1-cb7cfdb1007e",
    "state": "VERIFY_SUCCESS",
    "logic": "OR",
    "pushNotification": {
        "completionTime": "2018-12-28T15:38:43.620Z",
        "sendState": "SUCCESS",
        "startTime": "2018-12-28T15:38:43.532Z",
        "message": "A new verification is waiting...",
        "send": true,
        "pushToken": "[redacted]"
    }
}
```


In the response you can see that the *state* is *VERIFY_SUCCESS*. This indicates that the transaction has been successfully verified on the mobile application. Elsewhere in the response you can also see that the state of the push notification is now SUCCESS (because it was successfully sent) and you can see the *transactionData* and *signedData* received from the mobile application.

The application can now continue in the knowledge that Mobile PUSH 2FA is complete.

10.7 Test other signature type (if applicable)

If you have two signatures registered (i.e. user presence and fingerprint) then you can initiate a verification transaction using the other signature by running the **CHOOSE: Get Signatures by owner (return 2nd)** request and then running the **Initiate a Verification Transaction** request again.

10.8 Manage Signatures, Authenticators, and Authenticator Clients

There are REST calls available for management of signatures, authenticators, and Authenticator Clients.

A Signature can be deleted but this is not usually done when working with the IBM Verify application because it has no way to re-register a single signature.

Deleting an Authenticator deletes all the signatures and transactions associated with it. It will also send a push notification to the mobile app to initiate removal of the authenticator registration there too. It is also possible to delete an authenticator from the mobile application but this will NOT delete the authenticator definition in Cloud Identity.

Deleting an Authenticator Client deletes all Authenticators registered against it. This could affect a lot of users so careful thought should be given before doing this.

11 QR Code Login with IBM Verify

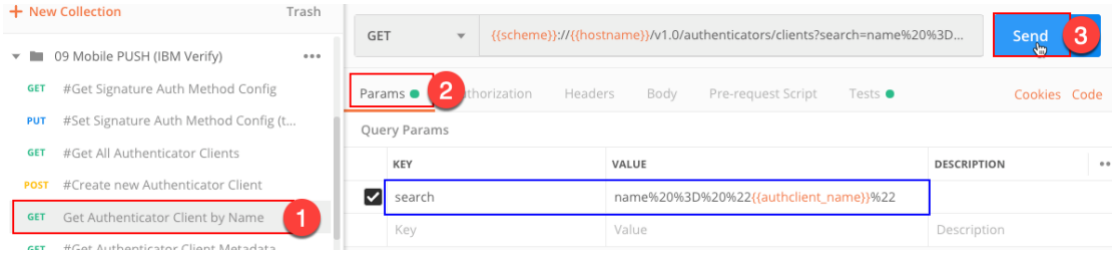
In this section you will see how QR Code Login can be performed using Cloud Identity Verify. QR Code Login allows first factor user authentication by simply scanning a QR Code with a registered authenticator (running on a mobile device).

For this section you must have already registered the IBM Verify application against your test user account. This was done as part of the setup of Mobile PUSH transaction verification in the previous section.

11.1 Look up Authenticator Client ID

A Cloud Identity Verify tenant can support multiple "Registration Profiles" for QR Code Login and Mobile PUSH applications. In the APIs a "Registration Profile" is known as an "Authenticator Client". For this exercise you will use the default profile, *Verify Profile.* You will now lookup this Authenticator Client by name to get the ID that is needed to identify the authenticator client during enrollment. It will be stored in the *authclient_id* variable.

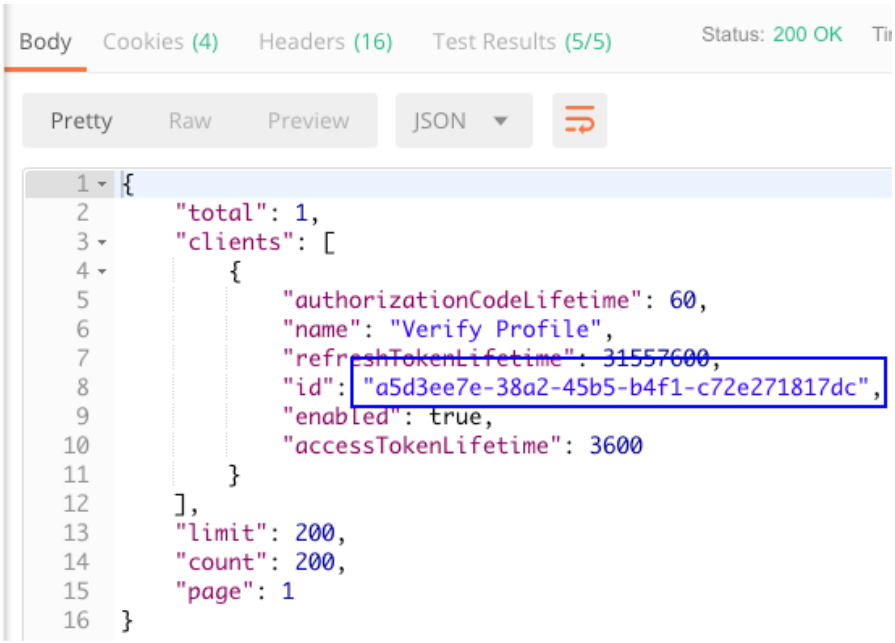
> The "Verify Registration Profile" ID is available directly in the Cloud Identity UI. If you prefer you could skip this lookup step and directly configure the ID as variable *authclient_id* in the Postman environment.



Expand the **10 QR Code Login** folder and select the **Get Authenticator Client by Name** request.

Select the **Params** tab and note the search filter of *name = "<authclient_name>"* being used for the lookup.

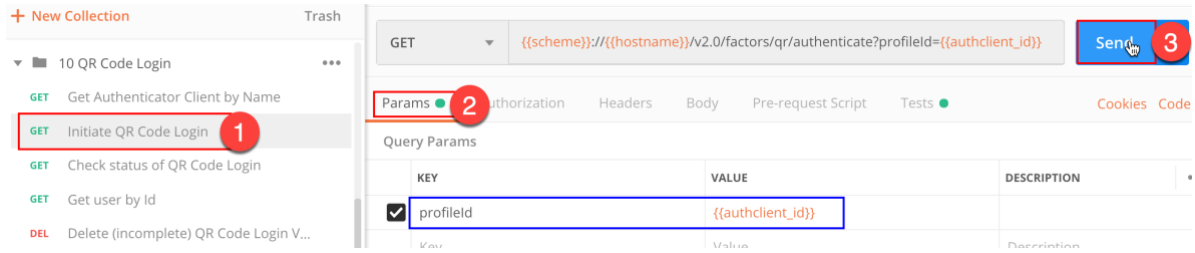
Click **Send**. The response is loaded below the request:



The *id* returned here is loaded to the *authclient_id* variable. Notice the other configuration information shown here. This can be updated in the Cloud Identity Admin UI or using REST calls.

11.2 Initiate QR Code Login Authentication

You can now initiate a QR Code Login authentication.



Select the **Initiate QR Code Login** request and select the **Params** tab.

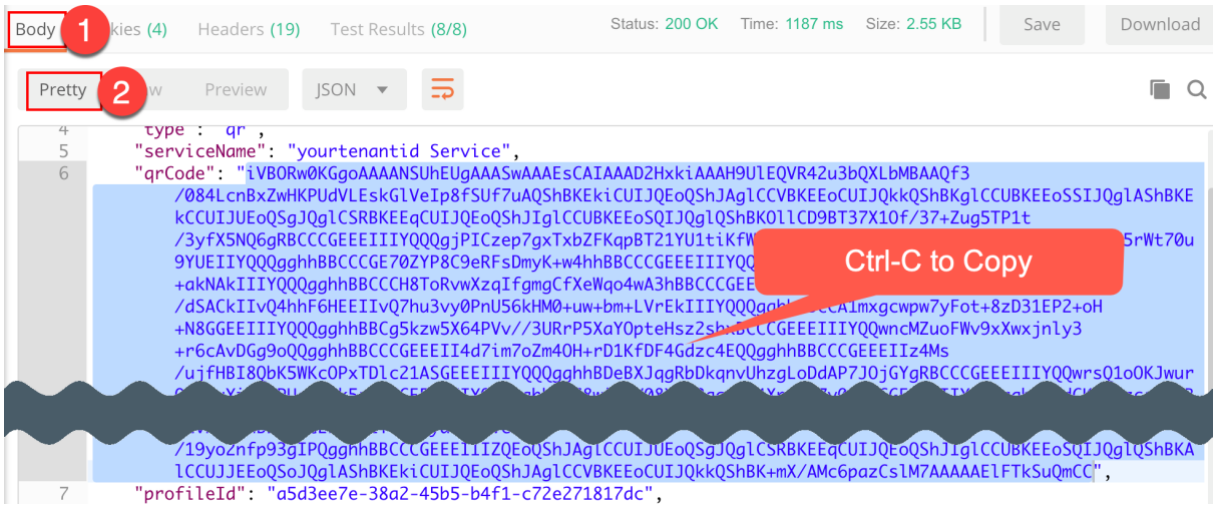
Review the request parameters being sent. The only parameter that is required I the *profileId* which identifies which Registration Profile this authentication applies to. This is necessary because a Cloud Identity tenant may have multiple Registration Profiles defined for different purposes.

Click **Send**. The response is loaded below the request.

The request includes an id (which identifies the authentication), a Login Session Index (lsi) which identifies the session to be authenticated, and a Device Session Index (dsi) which identifies the application when it polls for status.

11.3 Convert qrCode response object to an image

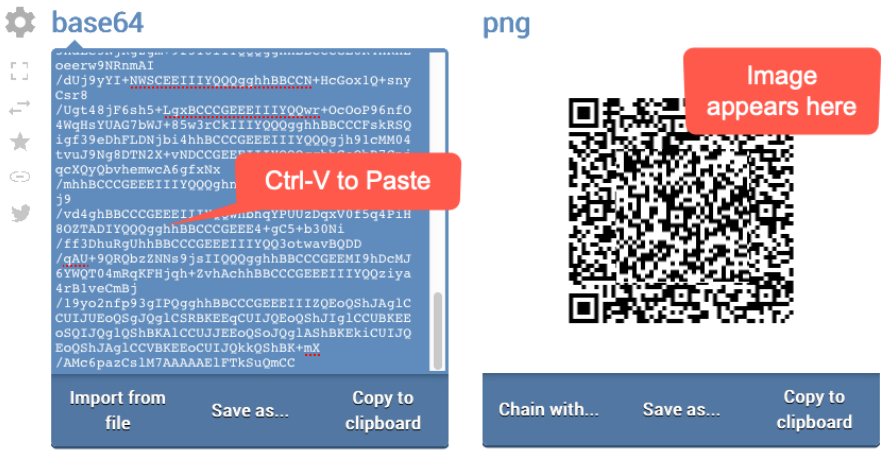
The QR Code to be displayed to the end user is returned as a Base64 encoded PNG image. The application using Cloud Identity Verify would need to convert this to an image and display to the user. For this exercise, you will use an on-line tool for the conversion.



Select the **Body** tab of the response and then select **Pretty** as the display option.

Select the value of the *qrCode* parameter in the response and copy it. You can either use Ctrl-c or right-click and select Copy.

Open a browser and navigate to: **https://onlinepngtools.com/convert-base64-to-png**



Paste the Base64 text you copied above into the *base64* box. The image is displayed in the png box.

11.4 Scan QR Code image with IBM Verify application

To complete the login, open the IBM Verify application on your mobile device and touch the bar code icon in the top-left of the display.

Scan the QR Code image that you created from the Base64 encoded text.

The IBM Verify application receives a “Login Session Identifier” in the QR Code and information on the server endpoints it needs to interact with. It matches the endpoints against the user accounts it has registered locally.

> If you have multiple users of your Cloud Identity tenant registered in your IBM Verify application you will see a chooser at this point so you can select which one is authenticating.

The IBM Verify App connects to the QR Code Login endpoint of Cloud Identity and authenticates itself using OAuth (using Refresh/Access Token acquired during registration). It then acts on behalf of the user and marks the QR Login session as authenticated for that user.

11.5 Check status of QR Code Login

The only way for the API caller to know if a QR Code Login has been completed is to GET the status. A real application would poll this endpoint waiting for the completion state.

Select the **Check status of QR Code Login** request. Notice that the URL includes both the QR Code ID and the Device Session ID. Click **Send**.

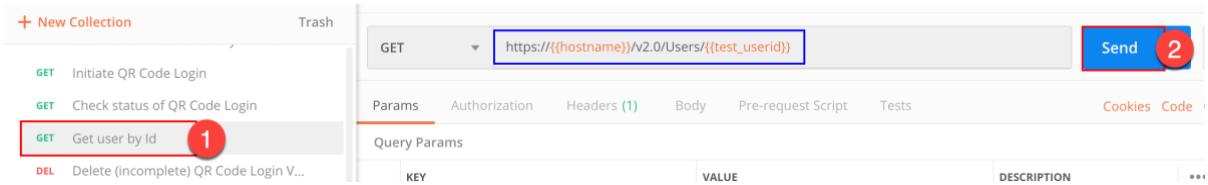

```
{
    "updatedBy": "50TU9QWFTQ",
    "created": "2019-07-10T14:15:10.651Z",
    "profileId": "a5d3ee7e-38a2-45b5-b4f1-c72e271817dc",
    "location": "https://yourtenantid.ice.ibmcloud.com/v2.0/factors/qr/9387ea81-9d0d-49c4-8041-e32188fc56d5",
    "id": "9387ea81-9d0d-49c4-8041-e32188fc56d5",
    "expiry": "2019-07-10T14:17:10.651Z",
    "state": "SUCCESS",
    "type": "qr",
    "serviceName": "yourtenantid Service",
    "userId": "50TU9QWFTQ",
    "updated": "2019-07-10T14:15:37.153Z",
    "tenant": "yourtenantid.ice.ibmcloud.com"
}
```

In the response you can see that the *state* is *SUCCESS*. This indicates that the QR Code Login was successfully completed. You can also see the *userId* of the user that is associated with the mobile device that scanned the QR Code.

The application can now treat the identified user as authenticated.

11.6 Lookup user information

Now that the application has the internal Cloud Identity userId of the authenticated user, it can make a standard Get User call to get user information (preferred username, group membership etc.).



Select the **Get user by Id** request and review the request URL. The *test_userid* was populated from the userId returned by the QR Code status call.

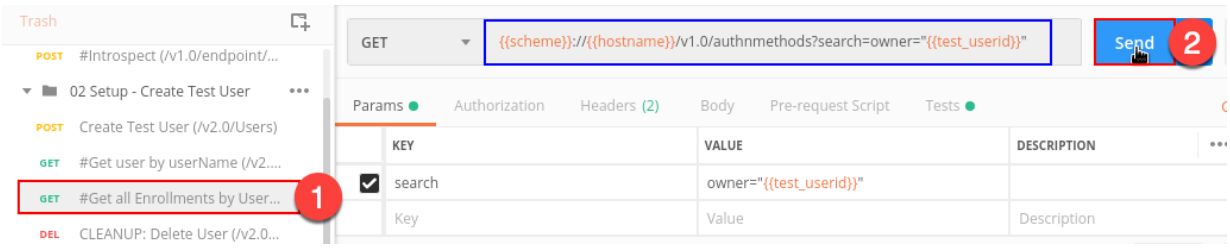
Click **Send**. The response is loaded below the request.


```
{
    "addresses": [
        {
            "country": "US",
            "streetAddress": "11101 Burnet Rd. Austin, TX",
            "postalCode": "78759",
            "locality": "Austin",
            "type": "work",
            "region": "TX"
        }
    ],
    "preferredLanguage": "en-US",
    "displayName": "Test User",
    "urn:ietf:params:scim:schemas:extension:ibm:2.0:User": {
        "userCategory": "regular",
        "twoFactorAuthentication": false,
        "realm": "cloudIdentityRealm",
        "pwdChangedTime": "2019-01-09T12:25:19Z"
    },
    "active": true,
    "userName": "testuser",
    "title": "End User",
    "phoneNumbers": [
```

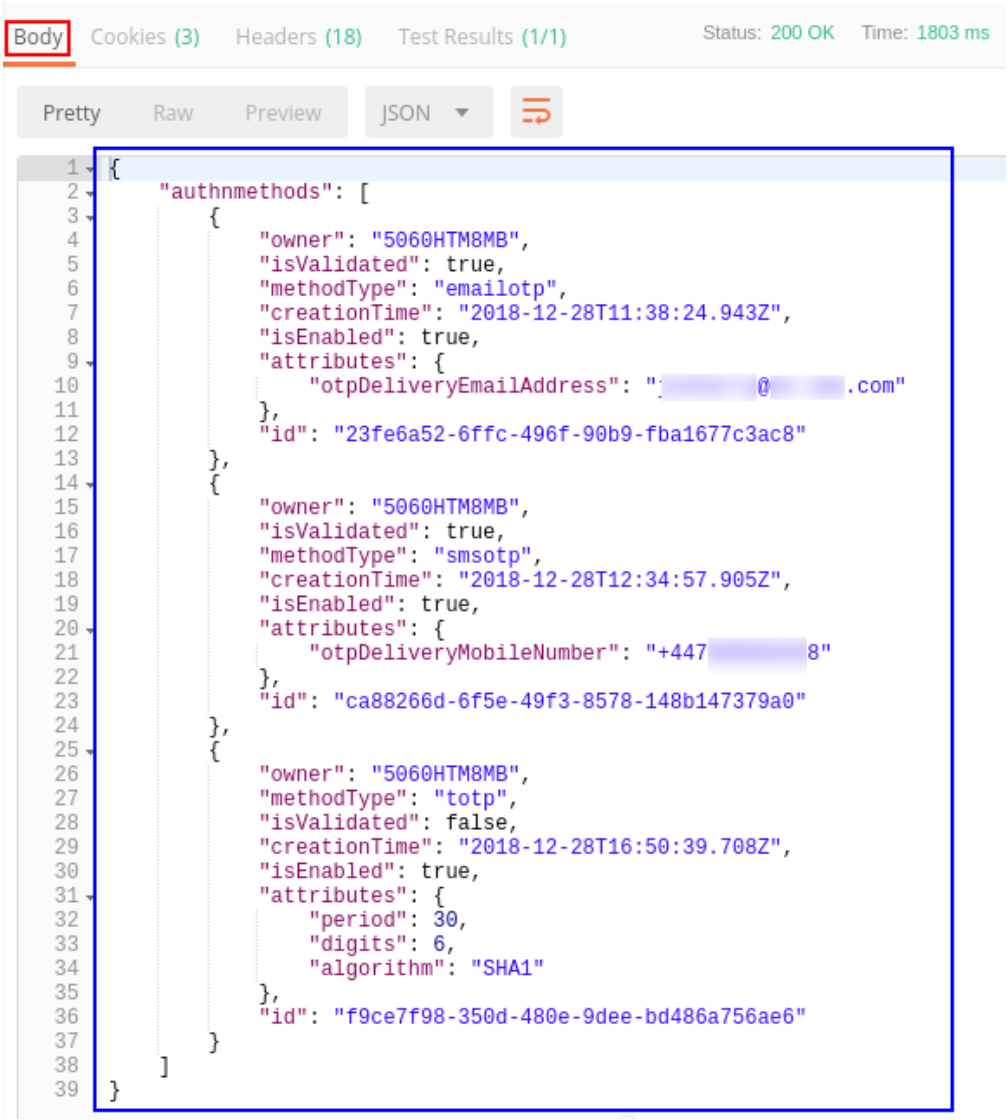

The application now has the details of the authenticated user.

12 Lookup OTP Enrollments for a user

When an application wants to perform a 2FA for a user, it might want to look up all available OTP enrollments. There is a REST API available for this. It is a search for all enrollments but with a filter on user ID.



Select the **Get all Enrollments by User** in the *02 Setup – Create Test User* folder. Notice the search filter in the URI; this filters responses by User ID. Click **Send**.



The response includes an array of *authmethods*. The type, enabled state, and validation state of each is shown along with attributes and the enrollment ID. The application could parse these results to pick the most appropriate 2FA method to use or to present a choice to the user.

This concludes the exercises in this cookbook.

13 Notices

Statement of Good Security Practices

IT system security involves protecting systems and information through prevention, detection and response to improper access from within and outside your enterprise. Improper access can result in information being altered, destroyed, misappropriated or misused or can result in damage to or misuse of your systems, including for use in attacks on others. No IT system or product should be considered completely secure and no single product, service or security measure can be completely effective in preventing improper use or access. IBM systems, products and services are designed to be part of a comprehensive security approach, which will necessarily involve additional operational procedures, and may require other systems, products or services to be most effective. IBM DOES NOT WARRANT THAT ANY SYSTEMS, PRODUCTS OR SERVICES ARE IMMUNE FROM, OR WILL MAKE YOUR ENTERPRISE IMMUNE FROM, THE MALICIOUS OR ILLEGAL CONDUCT OF ANY PARTY.

International Business Machines Corporation
New Orchard Road Armonk, NY 10504
Produced in the United States of America 01-2016