# Troubleshooting API Calls

4.8.1

## Troubleshooting REST API Requests

You have learned about REST APIs and explored them in a lab. In the provided scenarios, everything was working properly and was always successful. Of course, in real life, that is not always the way things work. At some point, you will find yourself making REST API requests and not getting the response that you expected. In this part, you will learn how to troubleshoot the most common REST API issues.

Note that when you apply these concepts and troubleshooting tips to a particular REST API, make sure you have the API reference guide and API authentication information handy; it will make your job a lot easier.

4.8.2

## No Response and HTTP Status Code from the API server

Let's start with a simple question:

True or false: every request that is sent to the REST API server will return a response with a status code.

The answer is false. While one might expect to always receive an HTTP status code such as 2xx, 4xx and 5xx, there are many cases where the API server cannot be reached or fails to respond. In those scenarios, you will not receive an HTTP status code because the API server is unable to send a response.

While a non-responsive server can be a big issue, the root cause of the unresponsiveness can be simple to identify. You can usually identify what went wrong from the error messages received as a result of the request.

Let's look at some troubleshooting tips to help you debug why the API server didn't send a response and HTTP status code, as well as some potential ways to fix the problem.

### Client side error

The first thing to check is whether you have a client-side error; it is here that you have more control in terms of fixing the issue.

### Is there a user error?

Some questions to ask to isolate errors from the user include:

- Is the URI correct?
- Has this request worked before?

When using a REST API for the first time, mistyping the URI is common. Check the API reference guide for that particular API to verify that the request URI is correct.

### Invalid URI example

To test the invalid URI condition, run a script such as this one, which simply makes the request to a URI that is missing the scheme. You can create a Python file or run it directly in a Python interpreter.

```
import requests
uri = "sandboxdnac.cisco.com/dna/intent/api/v1/network-device"
resp = requests.get(uri, verify = False)
```

The traceback will look like this:

```
....requests.exceptions.MissingSchema: Invalid URL
'sandboxdnac.cisco.com/dna/intent/api/v1/network-device': No schema supplied.
Perhaps you meant http://sandboxdnac.cisco.com/dna/intent/api/v1/network-
device?....
```

### Wrong domain name example

To test the wrong domain name condition, run a script such as this one, which simply makes the request to a URI that has the wrong domain name.

```
import requests
url = "https://sandboxdnac123.cisco.com/dna/intent/api/v1/network-device"
resp = requests.get(url, verify = False)
```

The traceback will look like this:

```
....requests.exceptions.ConnectionError:
HTTPSConnectionPool(host='sandboxdnac123.cisco.com', port=443): Max retries
exceeded with url: /dna/intent/api/v1/network-device (Caused by
NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at
0x109541080>: Failed to establish a new connection: [Errno 8] nodename nor servname
provided, or not known')).....
```

**Is there a connectivity issue?**

If the URI is correct, it may still be inaccessible. Ask yourself these questions:

- Are there any Proxy, Firewall or VPN issues?
- Is there an SSL error?

**Invalid certificate example**

This issue can only occur if the REST API URI uses a secure connection (HTTPS).

When the scheme of the URI is HTTPS, the connection will perform an SSL handshake between the client and the server in order to authenticate one another. This handshake needs to be successful before the REST API request is even sent to the API server.

When using the `requests` library in Python to make the REST API request, if the SSL handshake fails, the traceback will contain `requests.exceptions.SSLError`.

For example, if the SSL handshake fails due to the certificate verification failing, the traceback will look like this:

```
requests.exceptions.SSLError: HTTPSConnectionPool(host='xxxx', port=443): Max
retries exceeded with url: /dna/intent/api/v1/network-device (Caused by
SSLError(SSLCertVerificationError(1, '[SSL: CERTIFICATE_VERIFY_FAILED] certificate
verify failed: self signed certificate (_ssl.c:1108)')))
```

In this situation, you must fix the invalid certificate. But, if you are working in a lab environment where the certificates aren't valid yet, you can turn off the certificate verification setting.

To turn it off for the requests library in Python, add the verify parameter to the request.

```
resp = requests.get(url, verify = False)
```

**Resolution**: Client errors are usually simple to fix, especially when the error messages in the traceback indicate what may be wrong and the possible solution. Analyze the logs carefully for the root cause.

**Server side error**

After you've verified that there aren't any client-side errors, the next thing to check are server-side errors.

**Is the API server functioning?**

The most obvious place to start is to make sure that the API server itself is functioning properly. There are a few things you will want to ask yourself:

- Is the power off?
- Are there cabling issues?
- Did the domain name change?
- Is the network down?

To test if the IP address is accessible, run a script such as this one, which simply makes the request to the URL and waits for a response.

**Warning**: Expect a long delay when running this script.

```
import requests
url = "https://209.165.209.225/dna/intent/api/v1/network-device"
resp = requests.get(url, verify = False)
```

**Note**: The IP address in this script is bogus, but the script produces the same result as a server that is unreachable.

If the API server is not functioning, you'll get a long silence followed by a traceback that looks like this:

```
....requests.exceptions.ConnectionError:
HTTPSConnectionPool(host='209.165.209.225', port=443): Max retries exceeded with
url: /dna/intent/api/v1/network-device (Caused by
NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at
0x10502fe20>: Failed to establish a new connection: [Errno 60] Operation timed
out')).....
```

**Is there a communication issue between the API server and the client?**

If the server is functioning properly, you'll need to determine whether there's a reason the client isn't receiving a response. Ask yourself:

- Is the IP address and domain name accessible from the client's location on the network?
- Is the API server sending a response, but the client isn't receiving it?

To test this condition, use a network capturing tool to see if the response from the API server is lost in the communication between the API server and the client. If you have access, take a look at the API server logs to determine if the request was received, if it was processed, and if a response was sent.

**Resolution**: Server side issues cannot be resolved from the API client side. Contact the administrator of the API server to resolve this issue.

4.8.3

## Interpreting Status Codes

Because REST APIs are http-based, they also use http status codes to notify API clients of request results.

Unless otherwise stated, the status code is part of the HTTP/1.1 standard (RFC 7231), which means the first digit of the status code defines the class of response. (The last two digits do not have any class or categorization role, but help distinguish between results.) There are 5 of these categories:

- **1xx: Informational**: Request received, continuing to process.
- **2xx: Success**: The action was successfully received, understood, and accepted.
- **3xx: Redirection**: Further action must be taken in order to complete the request.
- **4xx: Client Error**: The request contains bad syntax or cannot be fulfilled.
- **5xx: Server Error**: The server failed to fulfill an apparently valid request.

Typically we see 2xx, 4xx and 5xx from the REST API server. Usually you can find the root cause of an error when you understand the meaning of the response codes. Sometimes the API server also provides additional information in the response body.

For all requests that have a return status code, perform these steps to troubleshoot errors:

**Step 1**: Check the return code. It can help to output the return code in your script during the development phase.

**Step 2**: Check the response body. Output the response body during development; most of the time you can find what went wrong in the response message sent along with the status code.

**Step 3**: If you can't resolve the issue using the above two steps, use the status code reference to understand the definition of the status code.

4.8.4

## 2xx and 4xx Status Codes

Let us look at these codes in more detail:

**2xx - Success**

When the client receives a 2xx response code, it means the client's request was successfully received, understood, and accepted. However, you should always verify that the response indicates success of the right action and that the script is doing what you think it should.

**4xx - Client side error**

A 4xx response means that the error is on the client side. Some servers may include an entity containing an explanation of the error. If not, here are some general guidelines for troubleshooting common 4xx errors:

- **400** - Bad Request

The request could not be understood by the server due to malformed syntax. Check your API syntax.

One cause of a 400 Bad Request error is the resource itself. Double-check the endpoint and resource you are calling, did you misspell one of the resources or forget the "s" to make it plural, such as `/device` versus `/devices` or `/interface` versus `/interfaces`? Is the URI well-formed and complete?

Another cause of a 400 Bad Request error might be a syntax issue in the JSON object that represents your POST request.

Let us look at an imaginary example service:

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/id/"
resp = requests.get(url,auth=("person2","super"),verify = False)
print (resp.status_code)
print (resp.text)
```

Can you guess what is wrong just by reading the code?

This example returns a status code of 400. The server side also tells you "No id field provided", because the `id` is mandatory for this API request. You would have to look that up in the documentation to be sure.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/id/1001001027331"
resp = requests.get(url,auth=("person2","super"),verify = False)
print (resp.status_code)
print (resp.text)
```

- **401** - Unauthorized

This error message means the server could not authenticate the request.

Check your credentials, including username, password, API key, token, and so on. If there are no issues with those items, you may want to check the request URI again, because the server may reject access in the case of an improper request URI.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/all"
resp = requests.get(url,verify = False)
print (resp.status_code)
print (resp.text)
```

This example returns a status code 401, can you guess what is wrong just by reading the code?

The request isn't providing a credential. The authentication `auth=("person1","great")` should be added in the code.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/all"
resp = requests.get(url,auth=("person1","great"),verify = False)
print (resp.status_code)
print (resp.text)
```

- **403** - Forbidden

In this case, the server recognizes the authentication credentials, but the client is not authorized to perform the request. Some APIs, such as Cisco DNA Center, have Role Based Access Control, and require a super-admin role to execute certain APIs. Again, the API reference guide may provide additional information.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp = requests.get(url,auth=("person1","great"),verify = False)
print (resp.status_code)
print (resp.text)
```

Why is this imaginary code snippet not working? We just used the username `person1` and the password `great` and it worked in last example.

The status code 403 is not an authentication issue; the server believes in the user's identity, it is just that the user does not have enough privileges to use that particular API. If we use `person2/super` as username/password for the authentication, it will work, because person2 has the privilege to execute this API. Can you envision how to make it work?

The authentication should be modified to use `person2/super` instead of `person1/great`.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp = requests.get(url,auth=("person2","super"),verify = False)
print (resp.status_code)
print (resp.text)
```

- **404** - Not Found

The server has not found anything matching the request URI; check the request URI to make sure it is correct. If the code used to work, you may want to check the latest API reference guide, as an API's syntax can change over time.

Consider the Cisco DNA Center "get all interfaces" API. The title says, "all interfaces", so you try to use `api/v1/interfaces`, but you get a 404 error because the API request is actually `api/v1/interface`.

```
import requests
url = "http://myservice/api/v1/resources/house/room/all"
resp = requests.get(url,auth=("person1","great"),verify = False)
print (resp.status_code)
print (resp.text)
```

Why did this script return a 404 status code, and can you guess how to fix it?

The actual URI is `/api/v1/resources/house/rooms/all`. There is no `room` resource, so the server was unable to find a resource matching the URI of the request.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/all"
resp = requests.get(url,auth=("person1","great"),verify = False)
print (resp.status_code)
print (resp.text)
```

- **405** - Method Not Allowed

In this case, the request was recognized by the server, but the method specified in the request has been rejected by the server. You may want to check the API reference guide to see what methods the server expects. The response from server may also include an `Allow header` containing a list of valid methods for the requested resource.

For example, if you mistakenly use the POST method for an API that expects the GET method you will receive a 405 error.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp = requests.post(url,auth=("person2","super"),verify = False)
print (resp.status_code)
print (resp.text)
```

- **406** - Not Acceptable

This error indicates that the target resource does not have a current representation that would be acceptable to the client. The server has the data, but cannot represent it using any of the options listed in the client's `Accept-` headers.

For example, the client is asking for SVG images: `Accept: image/svg+xml`

```
import requests
headers = {'Accept': 'image/svg+xml'}
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp = requests.get(url,headers=headers,auth=("person2","super"),verify = False)
print (resp.status_code)
print (resp.text)
```

In this case, the server does not support the requested resource, `image/svg+xml`, so it responds with a 406 error.

- **407** - Proxy Authentication Required

This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy. In this scenario, there is a proxy server between the client and server, and the 407 response code indicates that client needs to authenticate with the proxy server first.

- **409** - The request could not be completed due to a conflict with the current state of the target resource.

For example, an edit conflict where a resource is being edited by multiple users would cause a 409 error. Retrying the request later might succeed, as long as the conflict is resolved by the server.

- **415** - Unsupported Media Type

In this case, the client sent a request body in a format that the server does not support. For example, if the client sends XML to a server that only accepts JSON, the server would return a 415 error.

```
import requests
headers = {"content-type":"application/xml"}
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp = requests.get(url,headers=headers,auth=("person2","super"),verify = False)
print (resp.status_code)
print (resp.text)
```

From the error message, can you guess a what could fix the code?

Omitting the header or adding a header `{"content-type":"application/json"}` will work.

```
import requests
headers = {"content-type":"application/json"}

url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp = requests.get(url,headers=headers,auth=("person2","super"),verify = False)
print (resp.status_code)
print (resp.text)
```

These are just the most common 4xx response codes. If you encounter other 4xx response codes, you can refer to either RFC 2616, 6.1, Status-Line or RFC 7231, section 6, Response Status Codes for more information on what they mean.

4.8.5

## 5xx Status Codes

**5xx server side error**

A 5xx response indicates a server side error.

- **500** - Internal Server Error

This error means that the server encountered an unexpected condition that prevented it from fulfilling the request.

- **501** - Not Implemented

This error means that the server does not support the functionality required to fulfill this request. For example, the server will respond with a 501 code when it does not recognize the request method and is therefore incapable of supporting it for any resource.

- **502** - Bad Gateway

This error means that the server, while acting as a gateway or proxy, received an invalid response from an inbound server it accessed while attempting to fulfill the request.

- **503** - Service Unavailable

This code indicates that the server is currently unable to handle the request due to a temporary overload or scheduled maintenance, which will likely be resolved after a delay.

- **504** - Gateway Timeout

This error means that the server, while acting as a gateway or proxy, did not receive a timely response from an upstream server it needed to access in order to complete the request.

If you get a 500 or 501 error, check the API reference guide to make sure the request is valid. For other 5xx errors, check with your API server administrator to resolve the issue.

Before you start troubleshooting your API, it is crucial to have your API reference guide and status code references in hand. If you are unable to receive a response code, you will likely be able to identify the cause from the error log in your script. If you are able to receive a response code, you will be able to identify the root cause of the error, including whether the error is on the client or server side, by understanding the response code.

4.7
Working with Webhooks

4.9
Understanding and Using APIs Summary