PythonでのCatalyst Center APIの使用

内容

はじめに

このドキュメントでは、Pythonを使用してCisco Catalyst Centerで使用可能なさまざまなAPIを使用する方法について説明します。

前提条件

要件

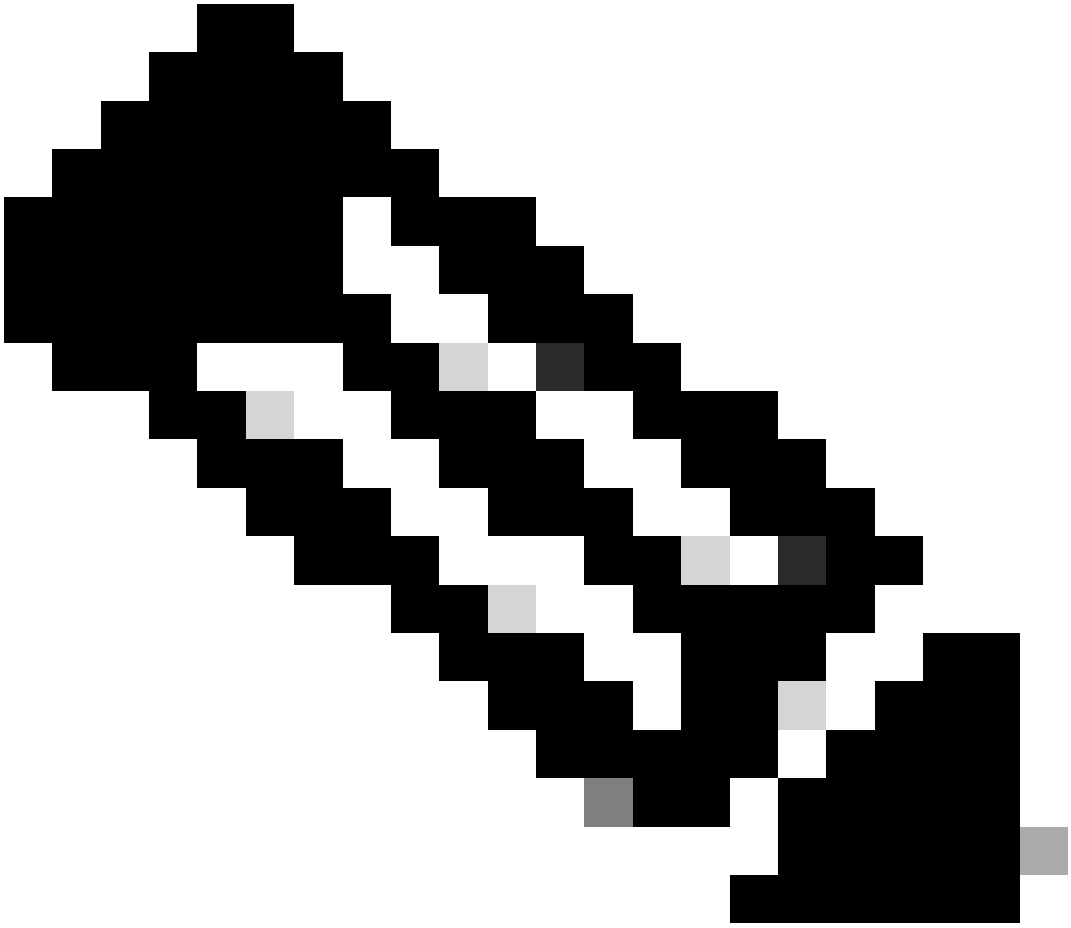
次の項目に関する基礎知識

- Cisco Catalystセンター
- API
- Python

使用するコンポーネント

- Cisco Catalystセンター2.3.5.x
- Python 3.x.x

このドキュメントの情報は、特定のラボ環境にあるデバイスに基づいて作成されました。このドキュメントで使用するすべてのデバイスは、クリアな（デフォルト）設定で作業を開始しています。本稼働中のネットワークでは、各コマンドによって起こる可能性がある影響を十分確認してください。

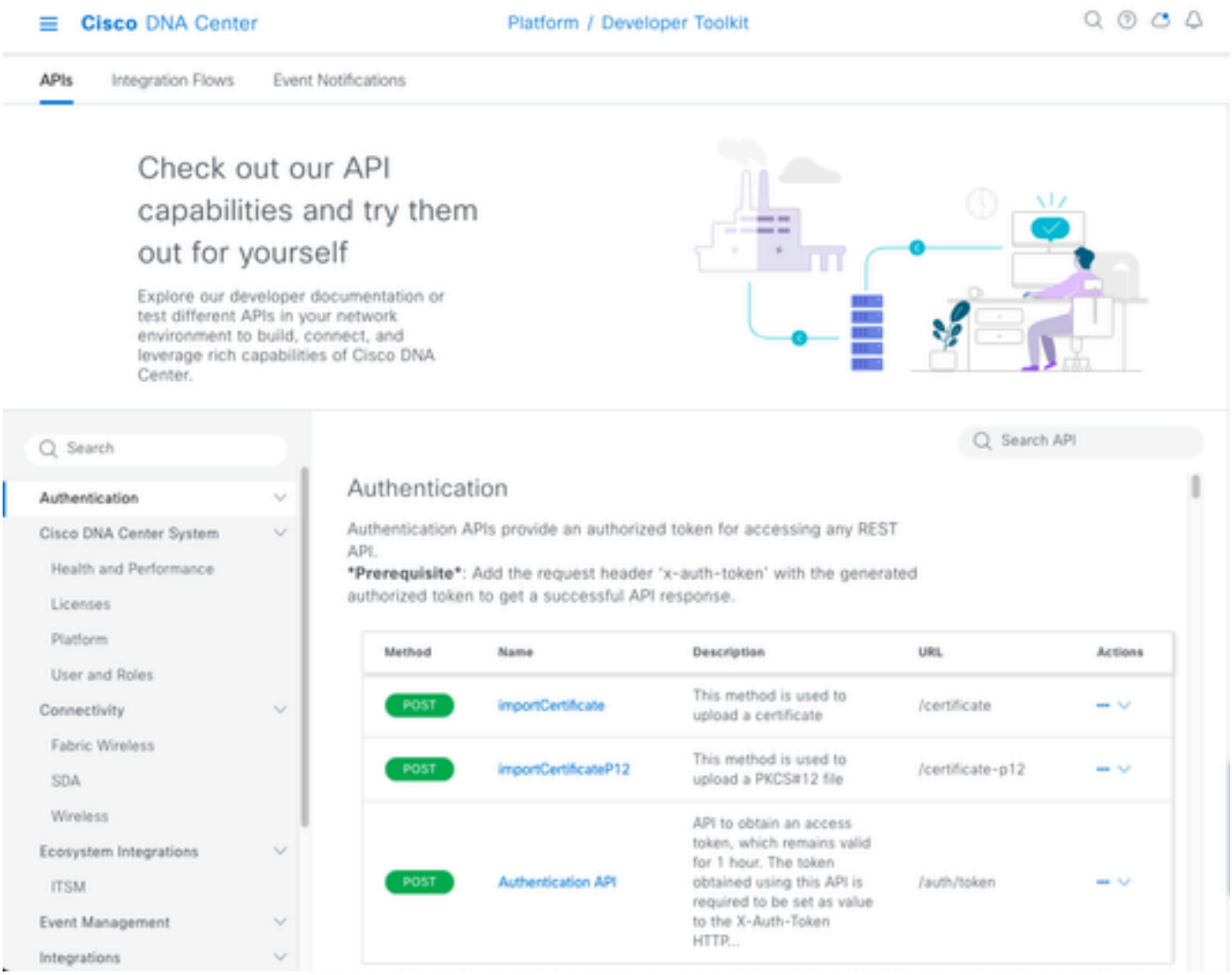


注:Cisco Technical Assistance Center(TAC)は、Pythonのテクニカルサポートを提供していません。Pythonに関する問題が発生した場合は、Pythonサポートに連絡してテクニカルサポートを受けてください。

設定

概要

Cisco Catalyst Centerでは、多数のAPIを使用できます。使用可能なAPIを確認するには、Catalyst CenterでPlatform > Developer Toolkit > APIsの順に選択します。

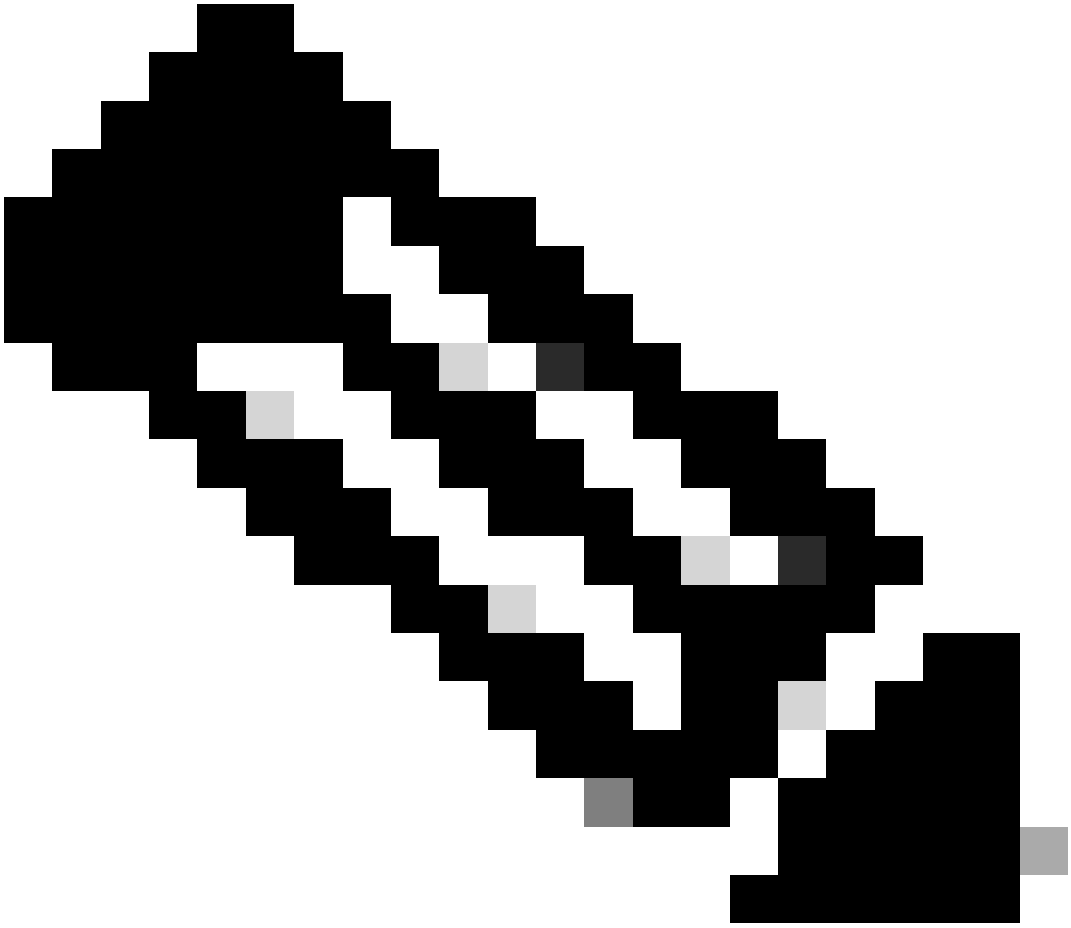


Catalyst Center APIページ

Catalyst Centerで実行する必要がある情報やアクションに応じて、各APIには独自の目的があります。APIが機能するためには、前提条件として、トークンを使用してCatalyst Centerに対して適切に認証し、正常なAPI応答を取得する必要があります。トークンは、それに応じてREST呼び出し元の特権を識別します。

また、APIを構成するコンポーネントを特定することも重要です。コンポーネントは次のとおりです。

- URL:特定のリソースにアクセスできるエンドポイント。
- メソッド：すべてのAPIにメソッドを含める必要があります。特定のエンドポイントに対してクライアントが実行するアクション/操作を定義する例：POST、GET、PUT、DELETE。
- ヘッダー：キーと値のペアの形式で要求に関する追加情報を提供します。たとえば、認可ヘッダーはクレデンシャルを使用した認証方式を提供します。
- パラメータ：APIを使用してエンドポイントに特定の命令を提供する変数。パラメータは、エンドポイントのURLの一部にすることができます。
- ペイロード：APIコール中にエンドポイントに送信する必要があるデータ。

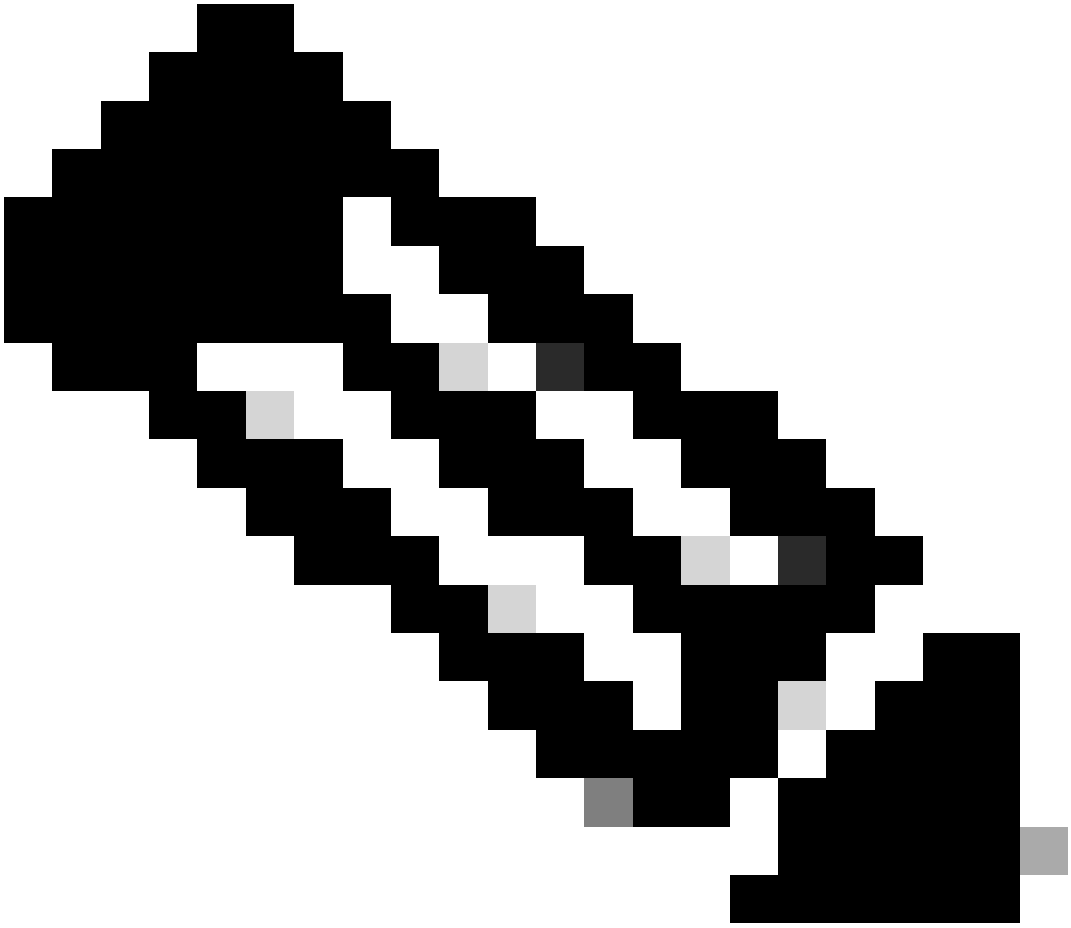


注:Catalyst Centerで使用可能な各APIについての詳細は、『APIリファレンス』ガイドを参照してください。

Modules

使用するPythonモジュール：

- requests:このモジュールでは、HTTP/1.1要求を特定のURLに送信できます。モジュールの詳細については、リクエストモジュールガイドを参照してください。
- base64:エンコードおよびデコード機能を提供します。 モジュールの詳細については、base64モジュールガイドを参照してください。
- json:このモジュールでは、API応答から特定のデータを取得できます。 モジュールの詳細については、jsonモジュールガイドを参照してください。



注：Pythonモジュールのインストール方法の詳細については、『Pythonモジュールのインストール』ドキュメントを参照してください。

トークンの生成

新しいトークンを生成するには、Authentication APIというAPIを使用する必要があります。

認証API:

```
POST      https://<CatalystCenterIP>/dna/system/api/v1/auth/token
```

生成されるトークンは1時間有効であることを明記することが重要です。1時間後、上記と同じAPIを使用して新しいトークンを生成する必要があります。

新しいPythonファイルで、モジュール(requests, base64 and json)をインポートし、4つの変数を

作成します。

```
import requests
import base64
import json

user = 'user'    # User to login to Catalyst Center
password = 'password'    # Password to login to Catalyst Center
token = ''    # Variable to store the token string
authorizationBase64 = ''   # Variable that stores Base64 encoded string of "username:password"
```

認証APIは、ヘッダー内の認証トークンとして基本認証をサポートします。基本認証(Basic Auth)は、エンドポイントの認証に使用できる方式で、コロンで区切られたユーザ名とパスワード(ユーザ名：パスワード)を指定します。両方の値はbase64でエンコードされ、エンドポイントはログインクレデンシャルをデコードして、ユーザがアクセスできるかどうかを確認します。

ユーザ名とパスワードに対応するBase64ベースの文字列を作成するために、base64モジュールが使用されます。これを行うには、b64encode関数を使用します。

```
byte_string = (f'{user}:{password}').encode("ascii")
authorizationBase64 = base64.b64encode(byte_string).decode()
```

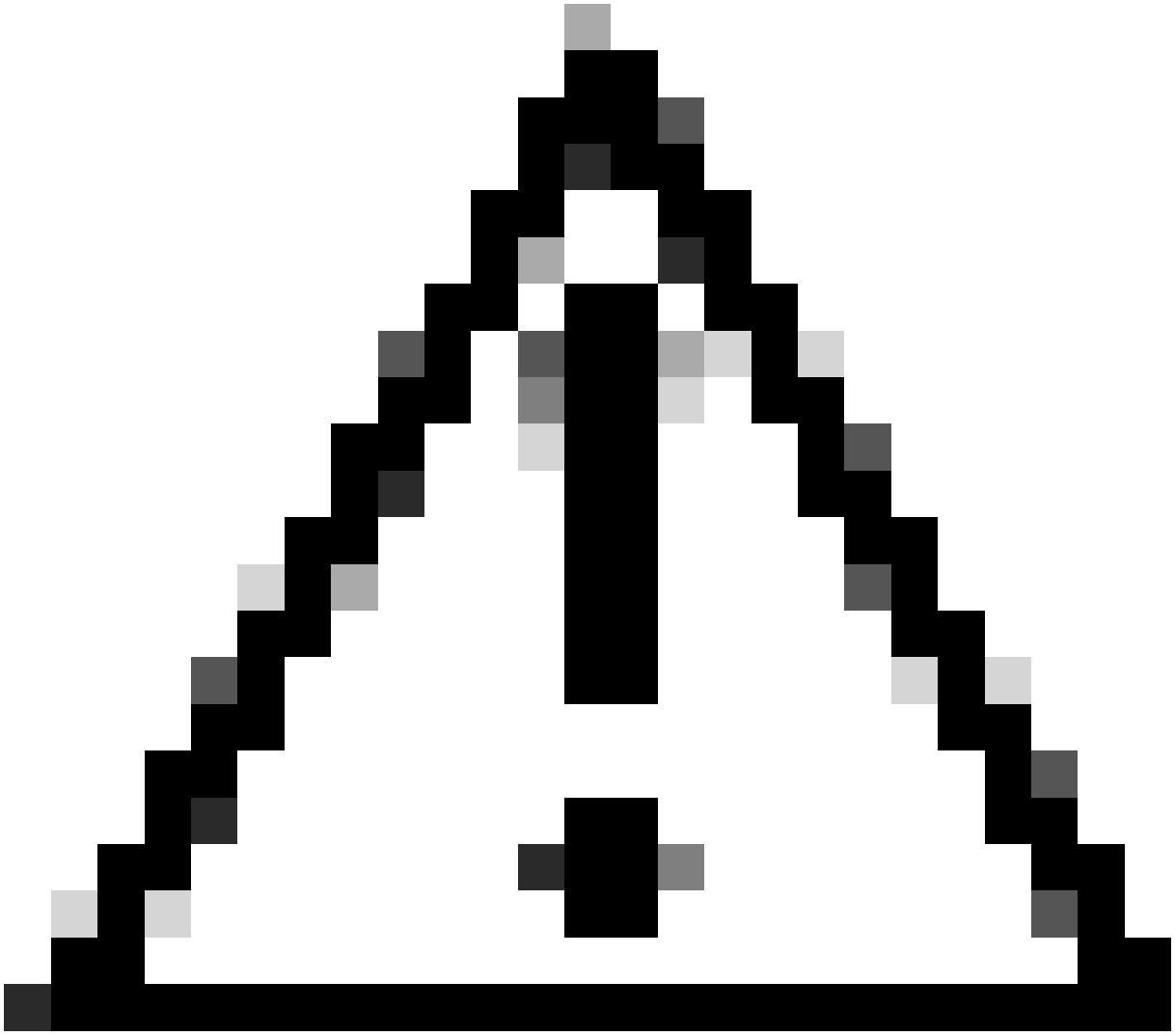
上記のコードでは、「.encode("ascii")」関数を使用してbyte_string変数が作成されています。これは、base64.b64encode関数がバイトのようなオブジェクトを必要とするためです。また、文字列形式'user:password'を保持するためにuser変数とpassword変数が使用されていたことにも注意してください。最後に、base64エンコードされたバイト文字列がユーザとパスワードで作成されました。'decode()'メソッドを使用して、値はstrオブジェクトに変換されました。

これを確認するには、authorizationBase64変数の値を出力します。

```
print(authorizationBase64)
```

出力例：

```
am9yZ2QhbDI6Sm9yZ2VhbDXxXxXx
```



注意： base64は暗号化アルゴリズムではありません。セキュリティ上の目的で使用しないでください。また、認証APIでは、ヘッダー内の認証トークンとしてAESキー暗号化がサポートされるため、セキュリティが向上します。

Catalyst Centerでの認証用にユーザとパスワードを使用してBase64エンコードの文字列が作成されました。次に、モジュール要求を使用してAPI認証 APIコールに進みます。また、requestという関数は、要求のテキストを含む応答オブジェクトの取得を許可します。

メソッドの構文：

```
requests.request(“method”, “url”, **kwargs)
```

**kwargsは、要求に渡されるパラメータ（cookie、ユーザエージェント、ペイロード、ヘッダーなど）を意味します。

認証APIでは、方式がPOST、URLが/dna/system/api/v1/auth/tokenと指定されています。ヘッダーで基本認証を指定する必要があります。

これらの変数は、request()関数で使用するために作成します。

```
url = https://<CatalystCenterIP>/api/system/v1/auth/token
headers = {
    'content-type': "application/json",
    'Authorization': 'Basic ' + authorizationBase64
    }
```

headers変数には、2つの項目が指定されています。最初のステートメントはcontent-typeです。これは、エンドポイントに送信されるリソースのメディアタイプを指定します（これにより、エンドポイントはデータを正確に解析して処理できます）。2つ目はAuthorizationです。この場合、変数authorizationBase64（この変数にはbase64が変換された文字列が格納されます）がパラメータとして送信され、Catalyst Centerに対して認証されます。

次に、request()関数を使用してAPIコールを実行します。次のコードは、関数の構文を示します。

```
response = requests.request("POST", url, headers=headers)
```

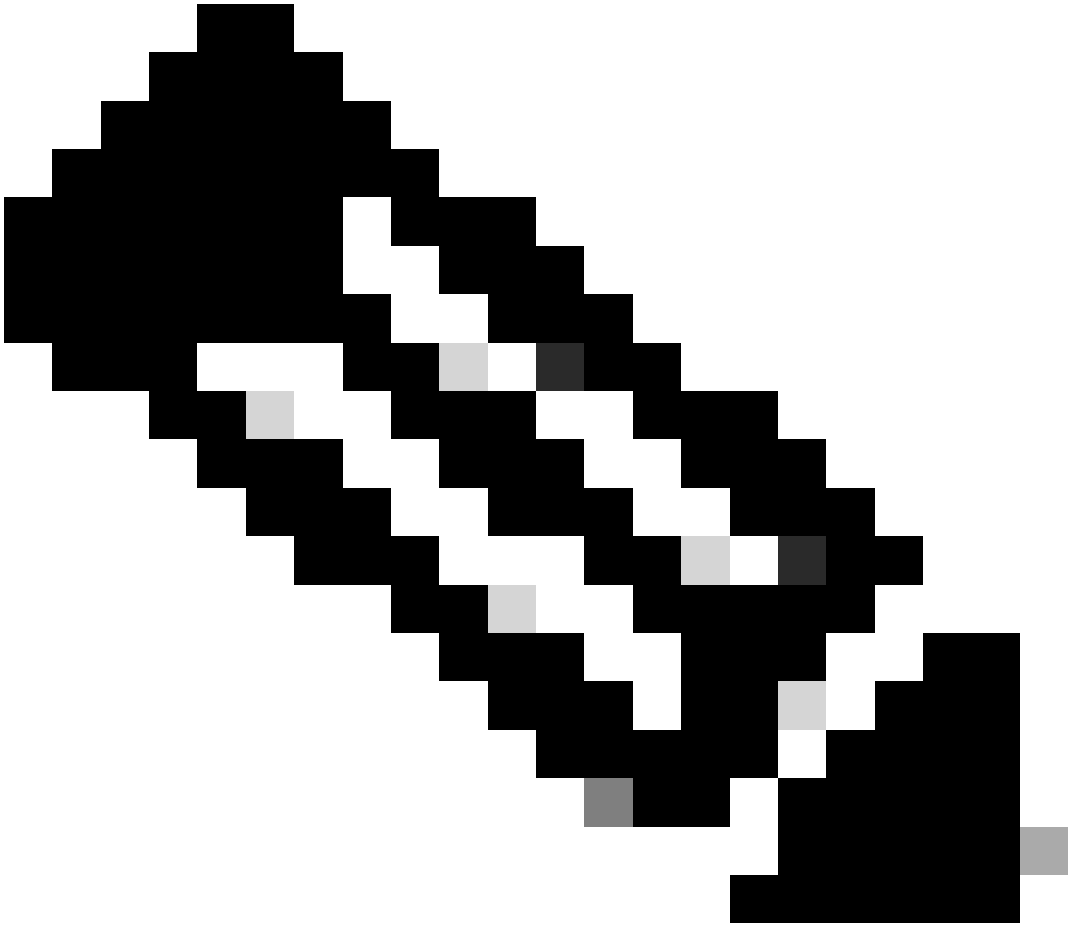
response変数は、実行されたAPIコールのデータを保存するために作成されました。

取得した応答を印刷するには、text()メソッドとともにresponse変数でprint関数を使用します。text()メソッドは、Catalyst Centerから受信した応答を使用してstrオブジェクトを生成します。

```
print(response.text)
```

出力例：

```
{"Token":"eyJhbGciOiJSUzI1NiIsInR5JKO9s1zVmNjkONjhkNTFhNDJ1ZWeLCU291cmNlIjoiaW5OZXJuYWwiLCW2vMPUbUOJNlq
!--- Output is supressed
```



注:Catalyst Centerが自己署名証明書を使用している場合、API要求は次の工ラーで失敗する可能性があります。

```
requests.exceptions.SSLError: HTTPSConnectionPool(host='X.X.X.X', port=443): Max retries exceeded
```

この問題を解決するには、verifyパラメータをFalseとしてrequest関数に追加する必要があります。これにより、エンドポイント(Catalyst Center)からのSSL証明書の確認が無視されます。

```
response = requests.request("POST", url, headers=headers, verify=False)
```

API認証コールから受信した応答から、この構造はPythonのディクショナリに似ていますが、

strオブジェクトであることに注意してください。

オブジェクトのタイプを検証するには、type()関数を使用します。

```
print(type(response.text))
```

次の出力が返されます。

```
<class 'str'>
```

実用的な目的のために、文字列全体ではなく、APIから受信した応答から抽出する必要があるのはトークン値だけです。これは、他のCatalyst Center APIを使用するために、パラメータとして渡される必要があるのはトークンだけであるためです。

APIコールから受信した応答は、Pythonのディクショナリと似た構造を持ちますが、オブジェクトタイプはstrであるため、このオブジェクトはjsonモジュールを使用してディクショナリに変換する必要があります。これにより、APIから受信した文字列全体からトークン値が抽出されます。

これを実現するために、json.loads()関数は文字列をディクショナリに変換し、後でトークン値のみを抽出して、それを直接トークン変数に割り当てます。

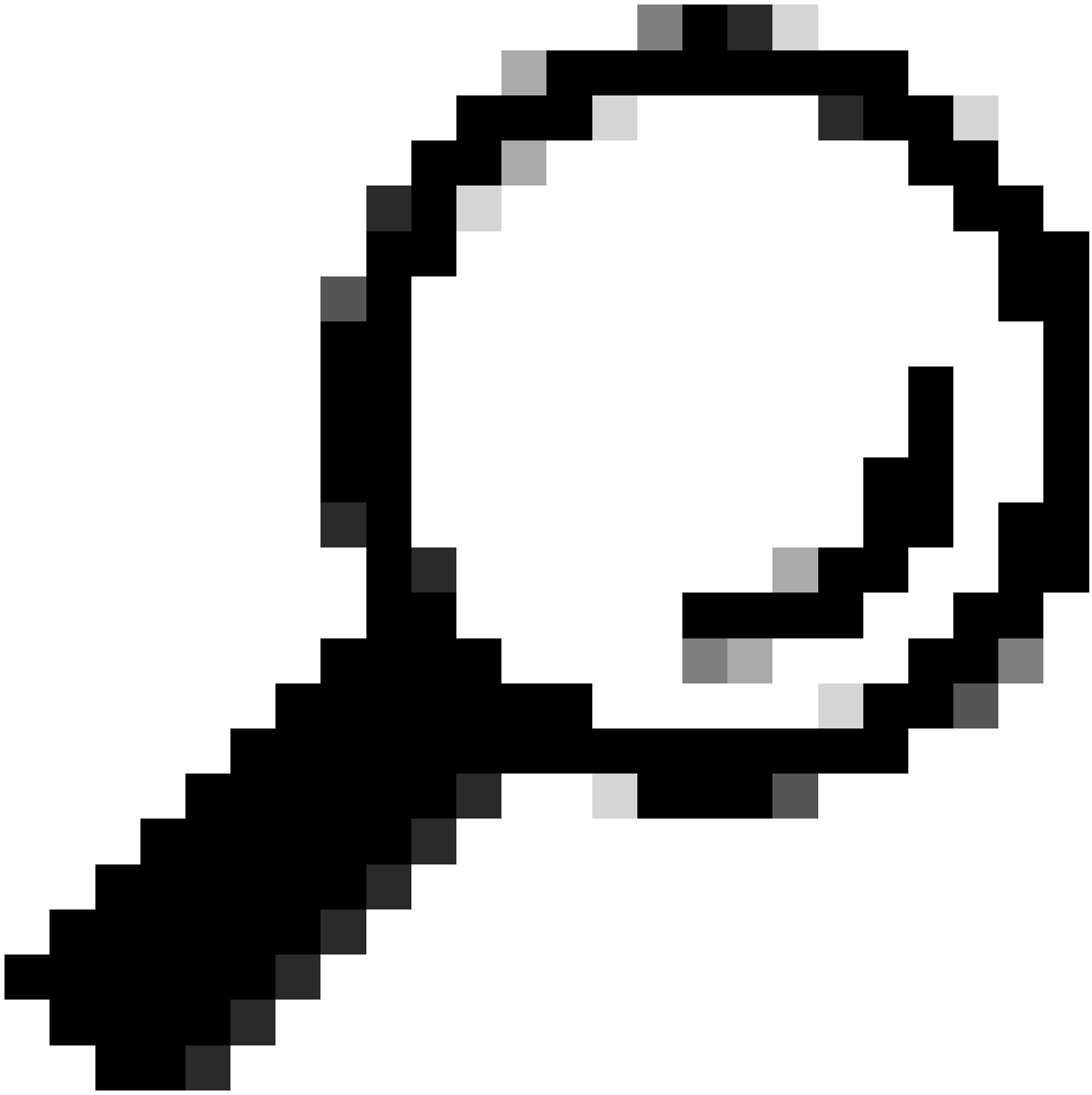
```
token = json.loads(response.text)  # Converting the response.text string value into a dictionary (It is
token = (token["Token"])  # Extracting just the token value by specifying the key as a parameter.
```

token変数の値がトークンのみであることを確認するには、印刷に進みます。

```
print(token)
```

出力例：

```
eyJhbGciOiJSUzI1NiIsInR5JKO9s1zVmNjkONjhkNTFhNDJ1ZWeLCU291cmNlIjoiaW5OZXJuYWwiLCW2vMPUbUOJNlqxOXNe1jMzY
!--- Output is supressed
```



ヒント：生成されたトークンは、デフォルトでは1時間で有効期限が切れるので、トークンを生成するコードを含むPythonメソッドを作成して、トークンが期限切れになるたびに呼び出すことができます。その際、作成したメソッドを呼び出すだけでプログラム全体を実行する必要はありません。

APIのテスト

トークンがtoken変数に正しく割り当てられたので、次に使用可能なCatalyst Center APIを使用できます。

この場合、Cisco DNA Center Nodes Configuration Summary APIがテストされます。

Cisco DNA Centerノードの設定の概要

```
GET     https://<CatalystCenterIP>/dna/intent/api/v1/nodes-config
```

このAPIは、設定されているNTPサーバ、ノード名、クラスタ内リンク、LACPモードなど、Catalyst Centerの現在の設定に関する詳細を提供します。

Cisco DNA Center Nodes Configuration Summary APIでは、使用する方式がGET、URLが"/dna/intent/api/v1/nodes-config"であることを指定しています。また、トークン文字列が抽出されてtoken変数に割り当てられているため、今回はトークンが変数としてAPIコールのヘッダーに「X-Auth-Token」:という形式で渡されます。

これにより、実行される各APIコールに対してCatalyst Centerへの要求が認証されます。各トークンの有効期間は1時間です。1時間後、Catalyst CenterへのAPIコールを続行するには、新しいトークンを生成する必要があります。

続いて、APIをテストする変数を作成します。

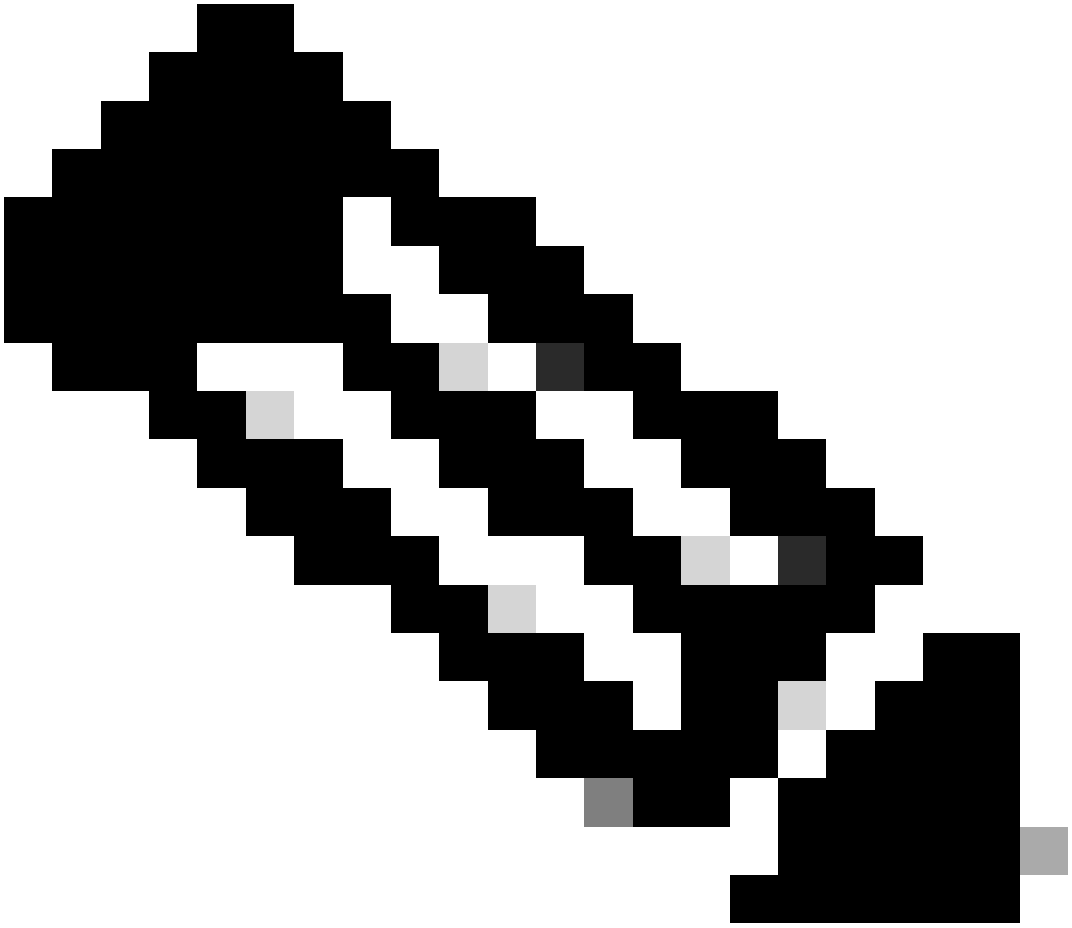
```
nodeInfo_url = "https://<CatalystCenterIP>/dna/intent/api/v1/nodes-config"
nodeInfo_headers = {
    'X-Auth-Token': token
}

nodeInfoResponse = requests.request("GET", nodeInfo_url, headers=nodeInfo_headers)
```

nodeInfo_url変数はAPIのURLを格納するために作成されました。nodeInfo_headers変数はAPIのヘッダーを格納します。この場合、'X-Auth-Token:'とtoken変数が、要求をCatalyst Centerに正常に認証するためのパラメータとして渡されています。最後に、nodeInfoResponse変数はAPIの応答を保存します。

受信した応答を検証するには、print()関数を使用します。

出力例：

```
{"response": {"nodes": [{"name": "Catalyst Center", "id": "ea5dbec1-fbb6-4339-9242-7694eb1cXxXx", "netw
!--- Output is supressed
```



注：自己署名証明書がCatalyst Centerで使用されている場合、API要求は次の工ラーで失敗する可能性があります。

```
requests.exceptions.SSLError: HTTPSConnectionPool(host='X.X.X.X', port=443): Max retries exceeded
```

この問題を修正するには、verifyパラメータをFalseとして要求に追加する必要があります。これにより、エンドポイント(Catalyst Center)からのSSL証明書の検証が抑制されます。

```
nodeInfoResponse = requests.request("GET", nodeInfo_url, headers=nodeInfo_headers, verify=False)
```

APIから受信した応答は読みづらいことがあります。json()モジュールを使用すると、より読みや

すい文字列で応答を出力できます。最初に、json.loads()関数と、その後に続くjson.dumps()関数を使用して、API応答をJSONオブジェクトにロードする必要があります。

```
jsonFormat = (json.loads(nodeInfoResponse.text)) # Creating a JSON object from the string received from
print(json.dumps(jsonFormat, indent=1)) # Printing the response in a more readable string using the dum
```

json.dumps:この関数は、パラメータとして取得したJSONオブジェクトをJSON形式の文字列で返します。

indent:このパラメータは、JSON形式の文字列のインデントレベルを定義します。

出力例：

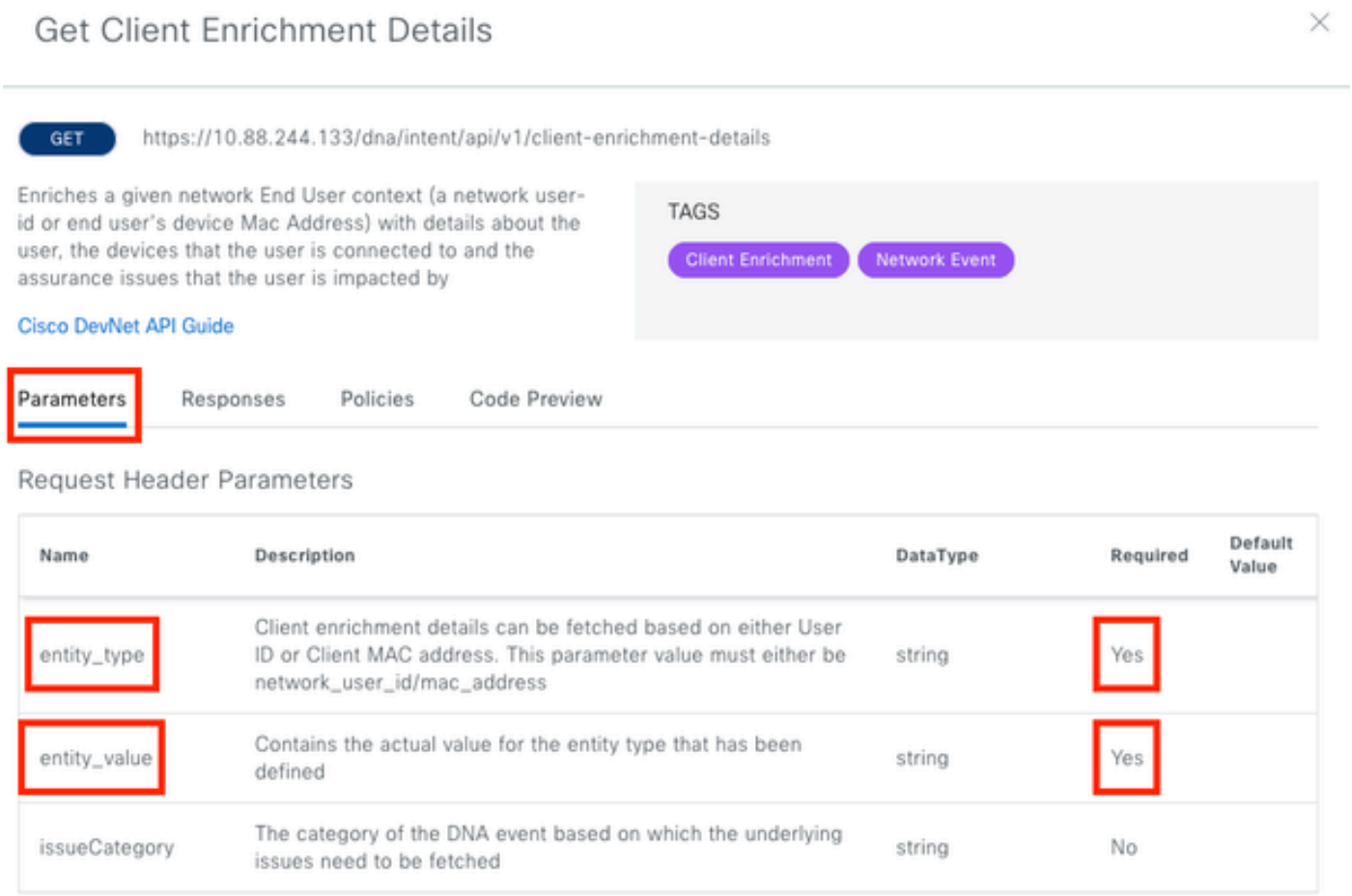
```
{
  "response": {
    "nodes": [
      {
        "name": "X.X.X.X",
        "id": "ea5dbec1-fbb6-4339-9242-7694eb1xXxX",
        "network": [
          {
            "slave": [
              "enp9s0"
            ],
            "lacp_supported": true,
            "intra_cluster_link": false,
!--- Output is supressed
```

ヘッダーパラメータを使用したAPI

一部のAPIでは、予期したとおりに動作させるために、いくつかのパラメータをヘッダーで送信する必要があります。この場合、Get Client Enrichment Details APIがテストされます。

```
GET        https://<CatalystCenterIP>/dna/intent/api/v1/client-enrichment-details
```

APIが予期したとおりに動作するために必要なヘッダーパラメータを確認するには、Platform > Developer Toolkit > APIs > Get Client Enrichment Detailsの順に移動し、APIの名前をクリックします。新しいウィンドウが開き、Parametersオプションの下に、APIが動作するために必要なHeaders Parametersが表示されます。



この場合、entity_typeパラメータについては、説明に従ってnetwork_user_idまたはmac_addressのいずれかの値にすることができ、entity_valueパラメータには定義済みのエンティティタイプの値を含める必要があります。

続行するには、2つの新しい変数entity_typeとentity_valueを対応する値とともに定義します。

```
entity_type = 'mac_address'   #This value could be either 'network_user_id' or 'mac_address'.
entity_value = 'e4:5f:02:ff:xx:xx'    #Depending of the 'entity_type' used, need to add the correspondi
```

APIコールを実行するための新しい変数も作成されます。APIコールのURLはuserEnrichment_url変数に保存されます。ヘッダーはuserEnrichmentHeaders変数に保存されます。受信した応答はuserEnrichmentResponse変数に保存されます。

```
userEnrichment_url = "https://<CatalystCenterIP>/dna/intent/api/v1/user-enrichment-details"

userEnrichmentHeaders = {
'X-Auth-Token': token,
'entity_type': entity_type,
'entity_value': entity_value,
}
```

```
userEnrichmentResponse = requests.request("GET", userEnrichment_url, headers=userEnrichmentHeaders)
```

userEnrichmentHeadersからわかるように、entity_type変数とentity_value変数が、token変数とともにAPIコール用のヘッダーパラメータとして渡されています。

受信した応答を検証するには、print()関数を使用します。

```
print(userEnrichmentResponse.text)
```

出力例：

```
[ {
  "userDetails" : {
    "id" : "E4:5F:02:FF:xx:xx",
    "connectionStatus" : "CONNECTED",
    "tracked" : "No",
    "hostType" : "WIRELESS",
    "userId" : null,
    "duid" : "",
    "identifier" : "jonberrypi-1",
    "hostName" : "jonberrypi-1",
    "hostOs" : null,
    "hostVersion" : null,
    "subType" : "RaspberryPi-Device",
    "firmwareVersion" : null,
    "deviceVendor" : null,
    "deviceForm" : null,
    "salesCode" : null,
    "countryCode" : null,
    "lastUpdated" : 1721225220000,
    "healthScore" : [ {
      "healthType" : "OVERALL",
      "reason" : "",
      "score" : 10
    }, {
      "healthType" : "ONBOARDED",
      "reason" : "",
      "score" : 4
!--- Output is suppressed
```

クエリパラメータを使用したAPI

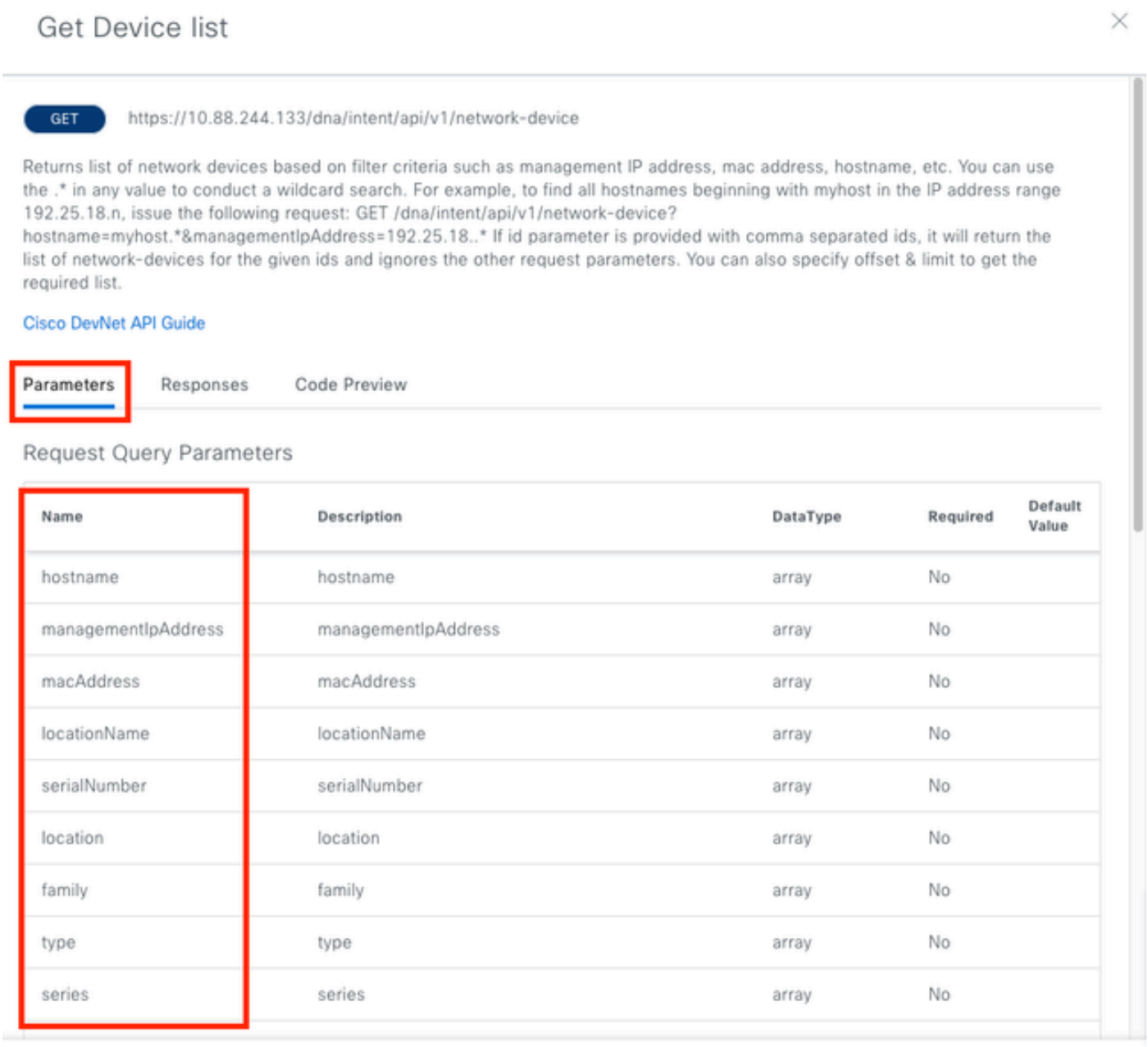
クエリー・パラメータを使用して、APIから返される特定の数の結果をフィルタできます。これらのパラメータは、APIのURLに追加されます。

Get Device List APIコールがテストされます。

```
GET          https://10.88.244.133/dna/intent/api/v1/network-device
```

Get Device List APIは、Catalyst Centerに追加されたすべてのデバイスのリストを返します。特定のデバイスの詳細が要求された場合は、クエリパラメータを使用して特定の情報をフィルタできます。

APIで使用可能なクエリパラメータを確認するには、Platform > Developer Toolkit > APIs > Get Device Listの順に選択し、APIの名前をクリックします。新しいウィンドウが開き、Parametersオプションの下に、APIで使用可能なクエリーパラメータが表示されます。



この例では、managementIpAddressとserialNumberのクエリパラメータが使用されています（APIコールですべてのクエリパラメータを使用する必要がないことを考慮してください）。続いて、両方のクエリーパラメータに対応する値を作成して割り当てます。

```
managementIpAddress = '10.82.143.250'
serialNumber = 'FDO25160X9L'
```

前述のように、クエリパラメータはAPIのURLに追加されます。具体的には、クエリパラメータの後に「?」を使用して、APIの最後に追加されます。

複数のクエリーパラメータを使用する場合は、それらの間に&'記号を配置して、いわゆるクエリ文字列を形成します。

次の例は、APIコールのURLを保存するdeviceListUrl変数にクエリパラメータを追加する方法を示しています。

```
deviceListUrl = "https://<CatalystCenterIP>/dna/intent/api/v1/network-device?managementIpAddresss=" + ma
```

先に作成した変数がURL文字列に追加されていることに注意してください。つまり、URLの文字列全体は次のようになります。

```
deviceListUrl = "https://<CatalystCenterIP>/dna/intent/api/v1/network-device?managementIpAddresss=10.82
```

APIコールを続行すると、APIヘッダーを格納するdeviceListHeaders変数がパラメータとして渡されるtoken変数とともに作成され、deviceListResponse変数にAPI応答が格納されます。

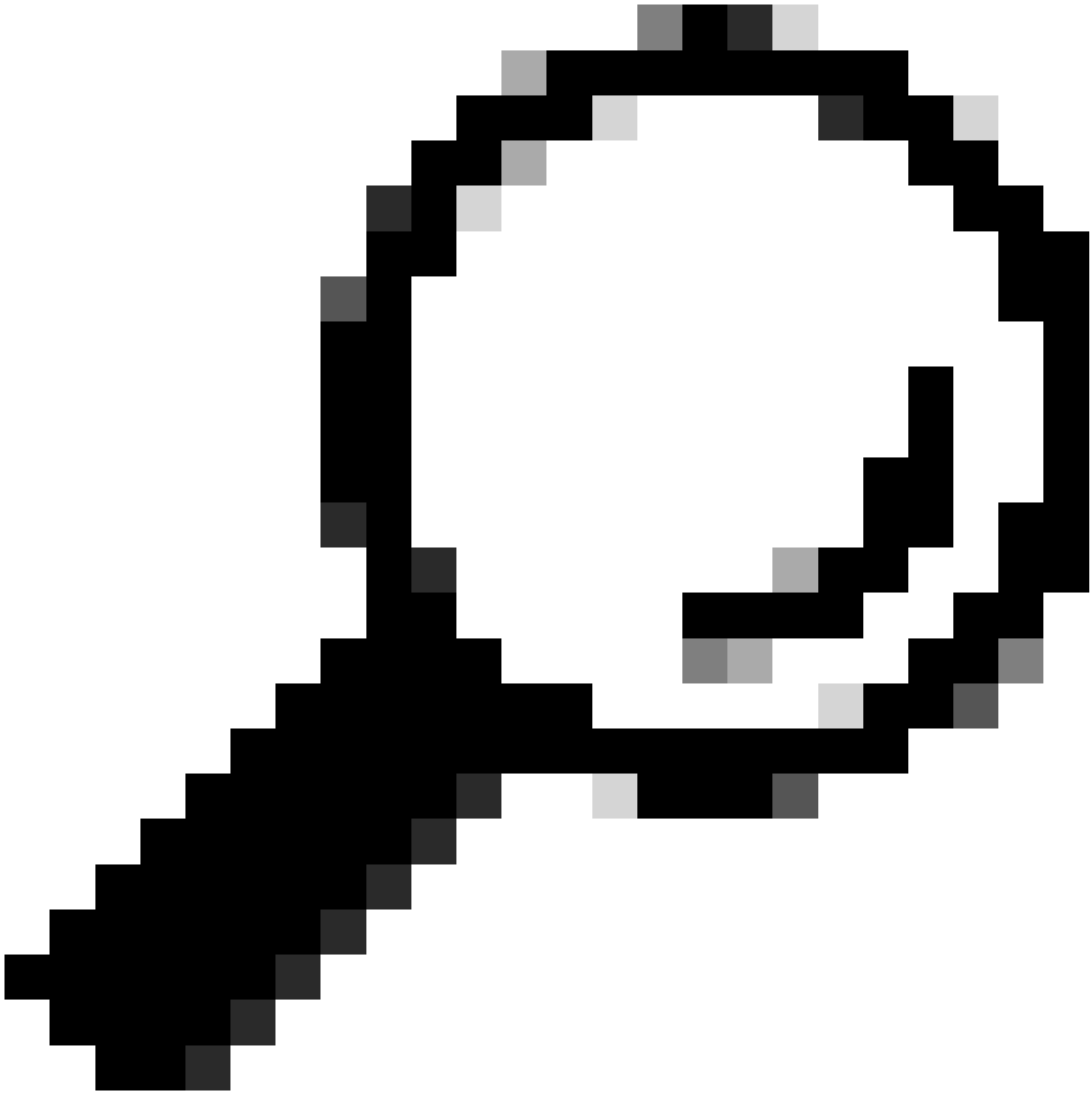
```
deviceListHeaders = {
            'X-Auth-Token': token,
        }

deviceListResponse = requests.request("GET", deviceListUrl, headers=deviceListHeaders)
```

受信した応答を検証するには、print()関数を使用します。

```
print(deviceListResponse.text)
```

出力例：

```
{"response":[{"family":"Switches and Hubs","description":"Cisco IOS Software [Cupertino], Catalyst L3 Sw
!--- Output is suppressed
```



ヒント：より読みやすい方法で応答を出力するには、「APIのテスト」セクションで説明しているjson.loads()およびjson.dumps()関数を使用できます。

翻訳について
シスコは世界中のユーザにそれぞれの言語でサポート コンテンツを提供するために、機械と人による翻訳を組み合わせて、本ドキュメントを翻訳しています。ただし、最高度の機械翻訳であっても、専門家による翻訳のような正確性は確保されません。シスコは、これら翻訳の正確性について法的責任を負いません。原典である英語版（リンクからアクセス可能）もあわせて参照することを推奨します。