Utiliser les API Catalyst Center avec Python

Table des matières

Introduction

Ce document décrit comment utiliser les différentes API disponibles sur Cisco Catalyst Center à l'aide de Python.

Conditions préalables

Exigences

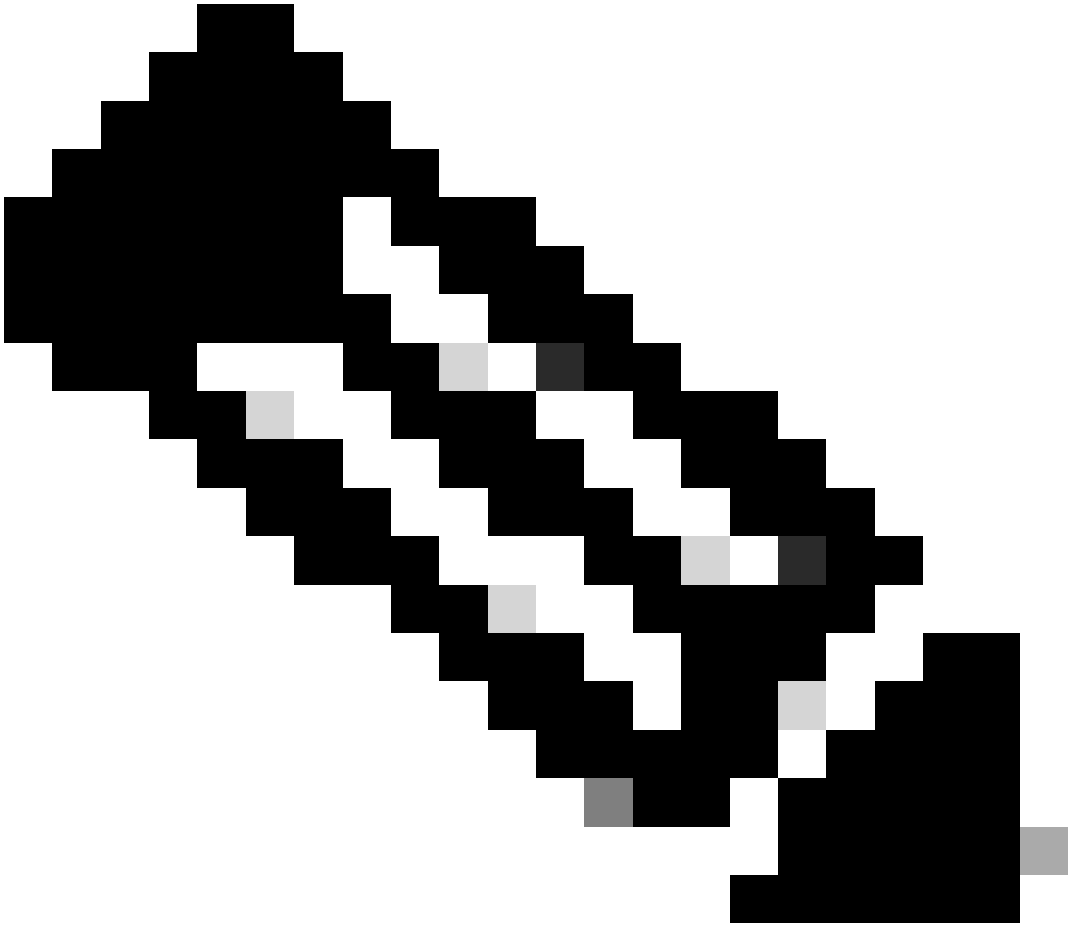
Connaissances de base sur :

- Cisco Catalyst Center
- API
- Python

Composants utilisés

- Cisco Catalyst Center 2.3.5.x
- Python 3.x.x

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. Si votre réseau est en ligne, assurez-vous de bien comprendre l’incidence possible des commandes.

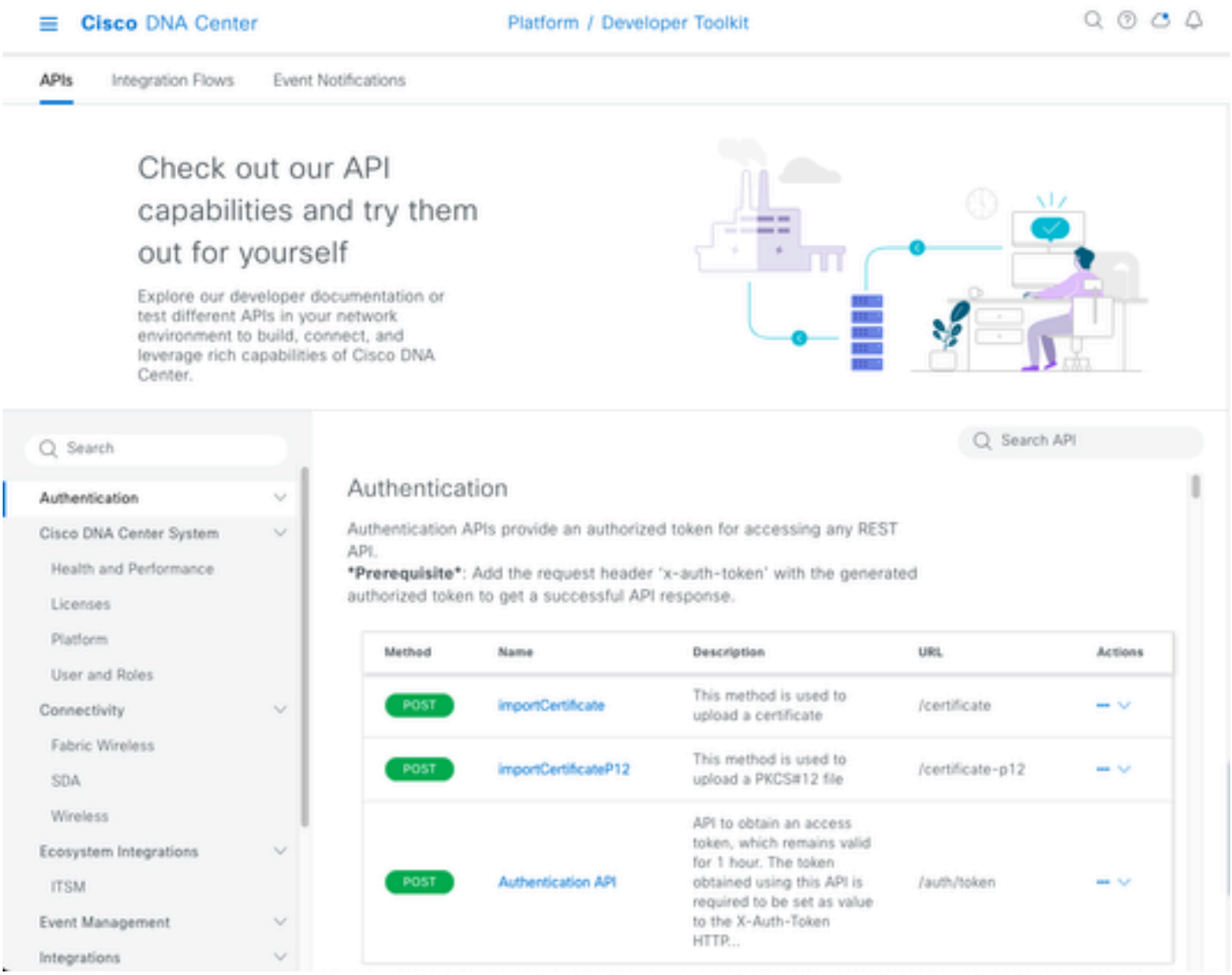


Remarque : le centre d'assistance technique Cisco (TAC) ne fournit pas d'assistance technique pour Python. Si vous rencontrez des problèmes avec Python, veuillez contacter l'assistance Python pour une assistance technique.

Configurer

Aperçu

Cisco Catalyst Center dispose de nombreuses API. Pour vérifier quelles API peuvent être utilisées, sur Catalyst Center, accédez à Platform > Developer Toolkit > APIs.

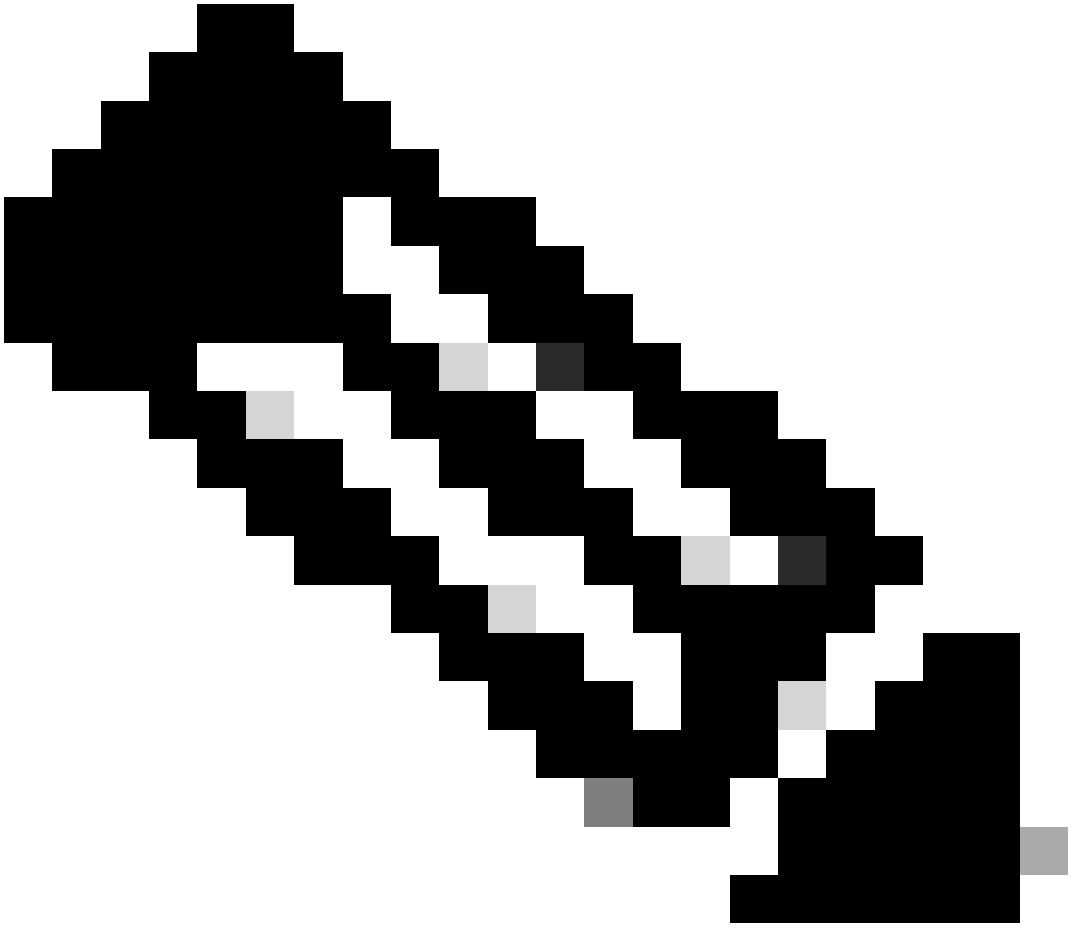


Page Catalyst Center API

Chaque API a sa propre fonction, en fonction des informations ou de l'action à effectuer sur Catalyst Center. Pour que les API fonctionnent, un jeton doit être utilisé comme condition préalable pour s'authentifier correctement auprès de Catalyst Center et obtenir une réponse API réussie. Le jeton identifie les privilèges de l'appelant REST en conséquence.

Il est également important d'identifier les composants qui composent une API, à savoir :

- URL : point de terminaison permettant d'accéder à une ressource spécifique.
- Méthode : toutes les API doivent inclure une méthode. Il définit l'action/l'opération que le client souhaite effectuer sur le point d'extrémité spécifique. Exemples : POST, GET, PUT, DELETE.
- En-tête : fournit des informations supplémentaires sur la demande au format des paires clé-valeur. Par exemple, l'en-tête d'autorisation fournit une méthode d'authentification utilisant des informations d'identification.
- Paramètres : variables qui fournissent des instructions spécifiques au point de terminaison à l'aide de l'API. Les paramètres peuvent faire partie de l'URL du point de terminaison.
- Données utiles : données qui doivent être envoyées au point d'extrémité pendant l'appel de l'API.

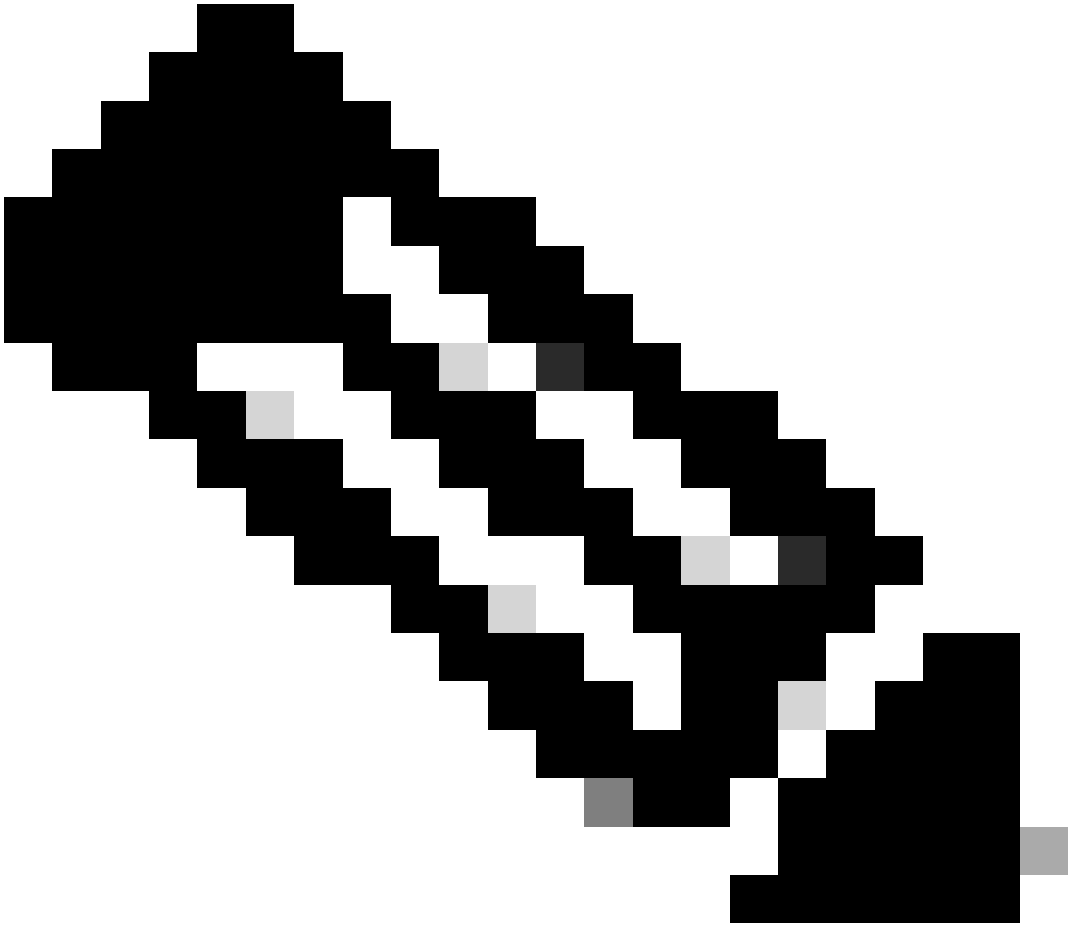


Remarque : pour plus d'informations sur chaque API disponible sur Catalyst Center, reportez-vous au guide API Reference.

Modules

Modules Python utilisés :

- requests : ce module permet d'envoyer des requêtes HTTP/1.1 à des URL spécifiques. Pour plus d'informations sur le module, consultez le guide du module de demande.
- base64 : fournit des fonctions de codage et de décodage. Pour plus d'informations sur le module, consultez le guide du module base64.
- json : ce module permet d'obtenir des données spécifiques à partir de la réponse des API. Pour plus d'informations sur le module, consultez le guide du module json.



Remarque : Pour plus d'informations sur l'installation des modules Python, consultez la documentation Installation des modules Python.

Générer un jeton

L'API appelée API d'authentification doit être utilisée pour générer un nouveau jeton.

API d'authentification :

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

Il est important de mentionner que le jeton qui est généré est valide pendant 1 heure. Au bout d'une heure, un nouveau jeton doit être généré à l'aide de la même API que celle mentionnée ci-dessus.

Dans un nouveau fichier Python, importez les modules (requêtes, base64 et json) puis créez quatre variables :

```
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"
```

L'API d'authentification prend en charge Basic Auth comme jeton d'autorisation dans l'en-tête. L'authentification de base est une méthode qui peut être utilisée pour s'authentifier auprès d'un point de terminaison, en fournissant un nom d'utilisateur et un mot de passe séparés par deux points (username:password). Les deux valeurs sont codées en base64, et le terminal décode les informations d'identification de connexion et vérifie si l'utilisateur peut y accéder ou non.

Le module base64 est utilisé pour créer la chaîne en base64 de notre nom d'utilisateur et de notre mot de passe. Pour ce faire, la fonction b64encode est utilisée.

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

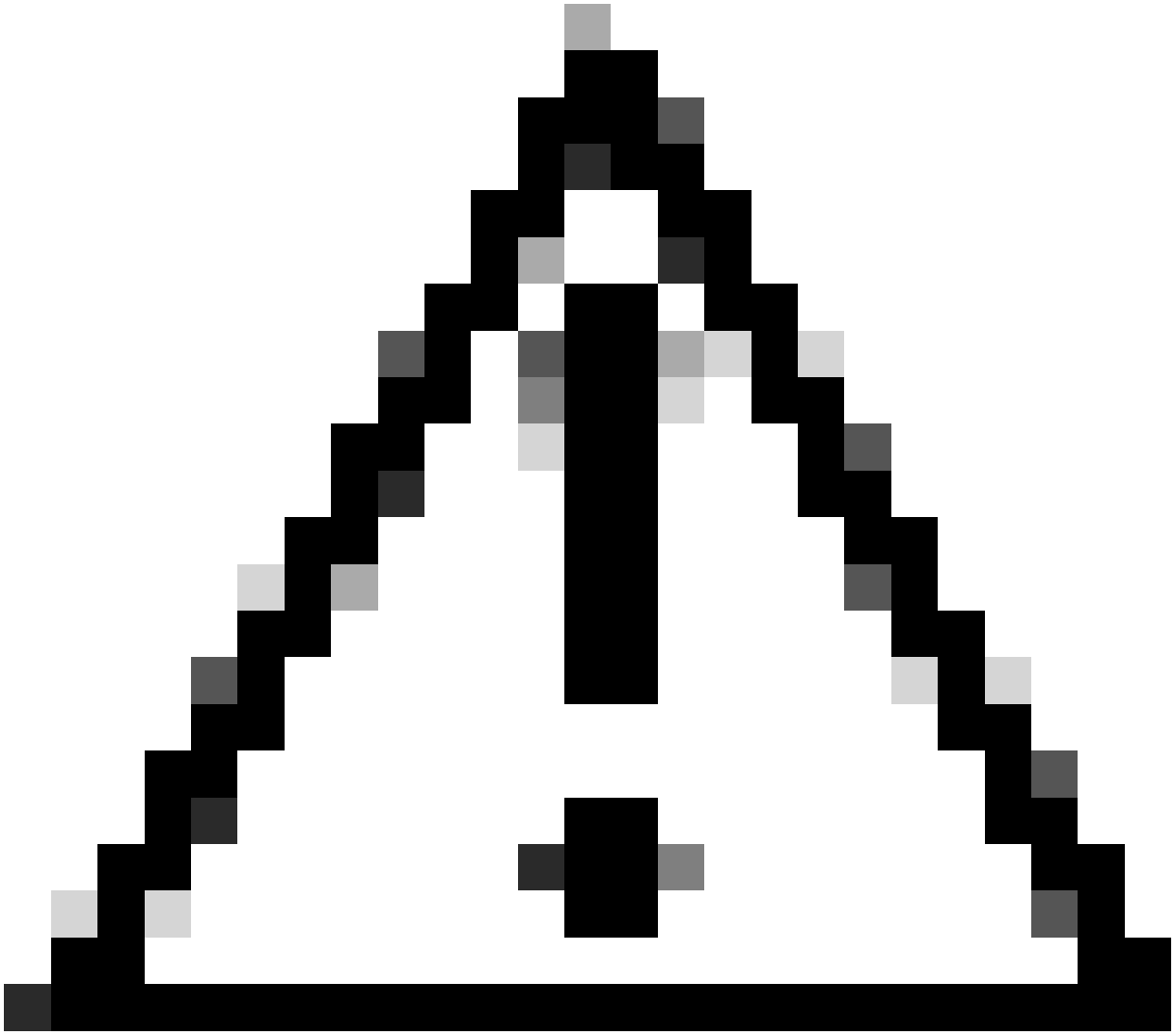
À partir du code ci-dessus, une variable byte_string a été créée à l'aide de la fonction '.encode(«ascii»). En effet, la fonction base64.b64encode nécessite un objet de type octets. Notez également que les variables user et password ont été utilisées pour conserver le format de chaîne 'user:password'. Enfin, une chaîne d'octets codée en base64 a été créée avec l'utilisateur et le mot de passe. En utilisant la méthode « decode() », la valeur a été convertie en objet str.

Pour le vérifier, vous pouvez imprimer la valeur de la variable authorizationBase64 :

```
print(authorizationBase64)
```

Exemple de résultat :

```
am9yZ2QhbDI6Sm9yZ2VhbDXxXxXx
```



Attention : base64 n'est pas un algorithme de chiffrement. Il ne doit pas être utilisé à des fins de sécurité. L'API d'authentification prend également en charge le chiffrement de clé AES comme jeton d'autorisation dans l'en-tête, ce qui offre plus de sécurité.

Maintenant qu'une chaîne codée en base64 a été créée à l'aide de l'utilisateur et du mot de passe pour s'authentifier auprès de Catalyst Center, il est temps de poursuivre l'appel d'API d'authentification d'API à l'aide des demandes du module. En outre, la fonction appelée request permet d'obtenir un objet de réponse qui contient le texte de la requête.

Syntaxie de la méthode :

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

**kwargs signifie tout paramètre passé dans la requête, par exemple, les cookies, les agents utilisateurs, la charge utile, les en-têtes, etc.

L'API d'authentification spécifie que la méthode est POST, que l'URL est «/dna/system/api/v1/auth/token » et que l'authentification de base doit être spécifiée dans l'en-tête.

Ces variables sont créées pour les utiliser pour la fonction request().

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

Pour la variable headers, deux éléments ont été spécifiés. Le premier est le content-type, qui spécifie le type de support de la ressource envoyée au point d'extrémité (cela aide le point d'extrémité à analyser et traiter les données avec précision). Le second est Authorization, qui, dans ce cas, la variable authorizationBase64 (qui stocke notre chaîne en code base64) est envoyée comme paramètre pour authentifier Catalyst Center.

Maintenant, continuez à utiliser la fonction request() pour effectuer l'appel d'API. Le code suivant montre la syntaxe de la fonction :

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

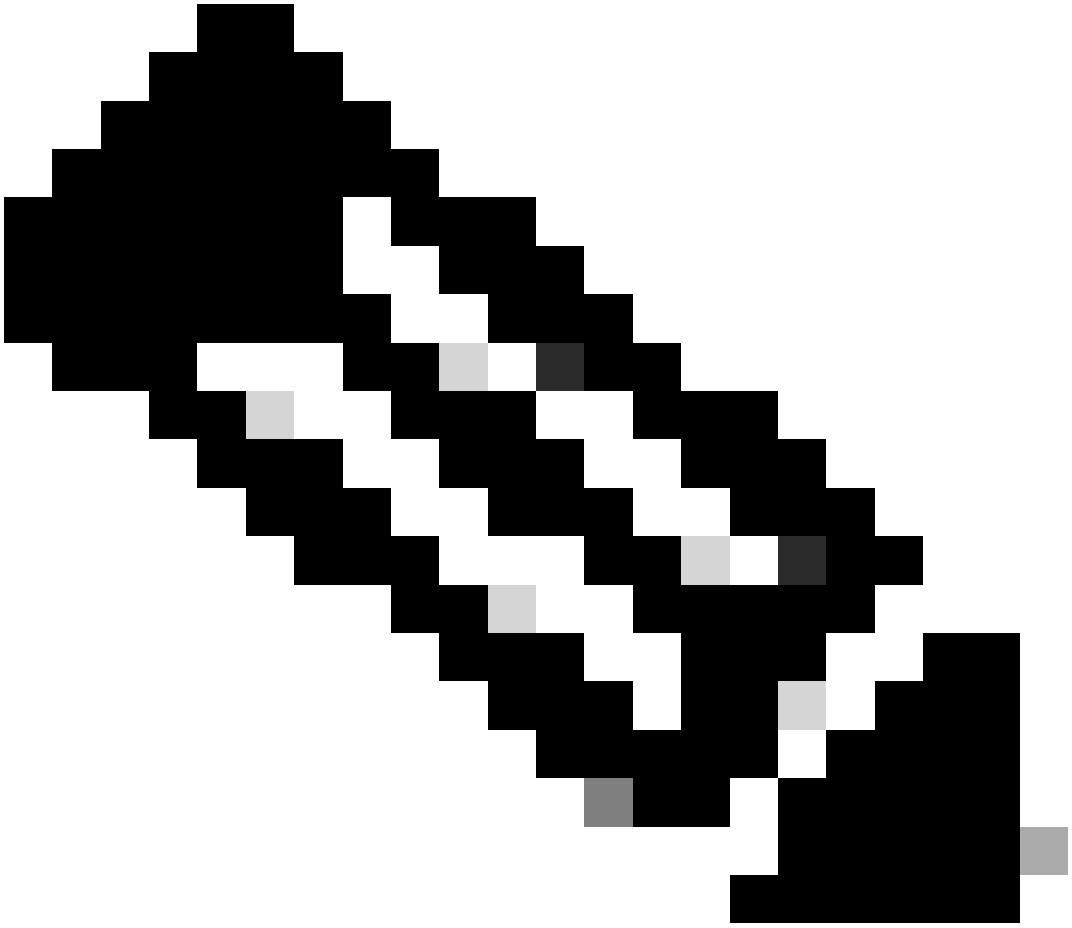
La variable response a été créée pour stocker les données de notre appel API effectué.

Pour imprimer la réponse obtenue, utilisez la fonction print avec la méthode text() dans la variable response. La méthode text() génère un objet str avec la réponse reçue de Catalyst Center.

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

Exemple de sortie :

```
{“Token”:“eyJhbGciOiJSUzI1NiIsInR5JKO9s1zVmNjkONjhkNTFhNDJ1ZWeLCU291cmNlIjoiaW5OZXJuYWwiLCW2vMPUbUOJNlq
!--- Output is supressed
```



Remarque : si Catalyst Center utilise un certificat auto-signé, la demande d'API peut échouer avec l'erreur suivante :

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

Pour résoudre ce problème, vous devez ajouter le paramètre verify comme False à la fonction request. Cela ignore la vérification du certificat SSL à partir du point d'extrémité (Catalyst Center).

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

D'après la réponse reçue de l'appel d'authentification de l'API, notez que la structure est similaire

à un dictionnaire en Python, cependant, c'est un objet str.

Pour valider le type d'un objet, utilisez la fonction type().

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

Ce qui renvoie le résultat suivant :

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

Pour des raisons pratiques, seule la valeur de jeton doit être extraite de la réponse reçue de l'API, et non la chaîne entière, puisque, pour utiliser les autres API de Catalyst Center, seul le jeton doit être passé comme paramètre.

Comme la réponse reçue de l'appel API a une structure similaire à un dictionnaire en Python mais que le type d'objet est str, ledit objet doit être converti en un dictionnaire à l'aide du module json. Ceci extrait la valeur de jeton de la chaîne entière reçue de l'API.

Pour ce faire, la fonction json.loads() convertit la chaîne en un dictionnaire pour extraire ultérieurement uniquement la valeur du jeton et l'affecter directement à notre variable jeton.

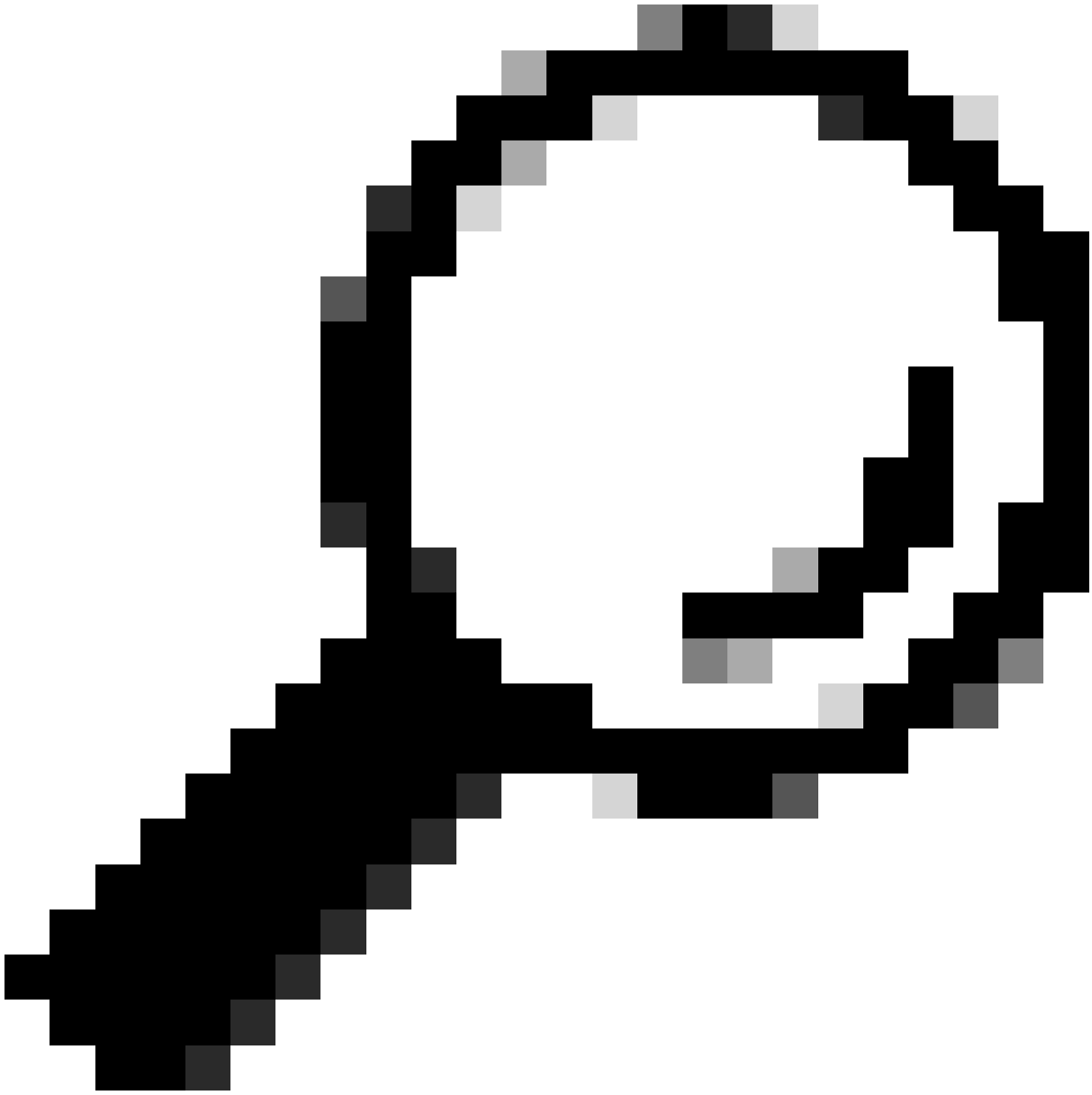
```
token = json.loads(response.text)  # Converting the response.text string value into a dictionary (It is
token = (token["Token"])  # Extracting just the token value by specifying the key as a parameter.
```

Pour vérifier que la variable token a uniquement le jeton comme valeur, continuez à l'imprimer.

```
print(token)
```

Exemple de sortie :

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



Conseil : Comme chaque jeton généré expire dans 1 heure par défaut, une méthode Python qui contient le code pour générer un jeton peut être créée et appelée chaque fois qu'un jeton expire, sans avoir à exécuter le programme entier en appelant simplement la méthode créée.

Test d'une API

Maintenant que le jeton a été assigné avec succès à la variable de jeton, les API Catalyst Center disponibles peuvent être utilisées.

Dans ce cas, l'API Cisco DNA Center Nodes Configuration Summary est testée.

Résumé de la configuration des noeuds Cisco DNA Center

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

Cette API fournit des détails sur la configuration actuelle de Catalyst Center, tels que le serveur NTP configuré, le nom du noeud, la liaison intra-cluster, le mode LACP, etc.

L'API Cisco DNA Center Nodes Configuration Summary spécifie, dans ce cas, que la méthode utilisée est GET, l'URL est « /dna/intent/api/v1/nodes-config » et, comme la chaîne de jeton a été extraite et attribuée à la variable de jeton, cette fois le jeton est passé en tant que variable dans l'en-tête de l'appel d'API comme « X-Auth-Token » : suivi du jeton.

Ceci authentifie la demande à Catalyst Center pour chaque appel d'API qui est effectué. Souvenez-vous que chaque jeton dure 1 heure. Après 1 heure, un nouveau jeton doit être généré pour continuer à passer des appels API à Catalyst Center.

Créez les variables pour tester l'API :

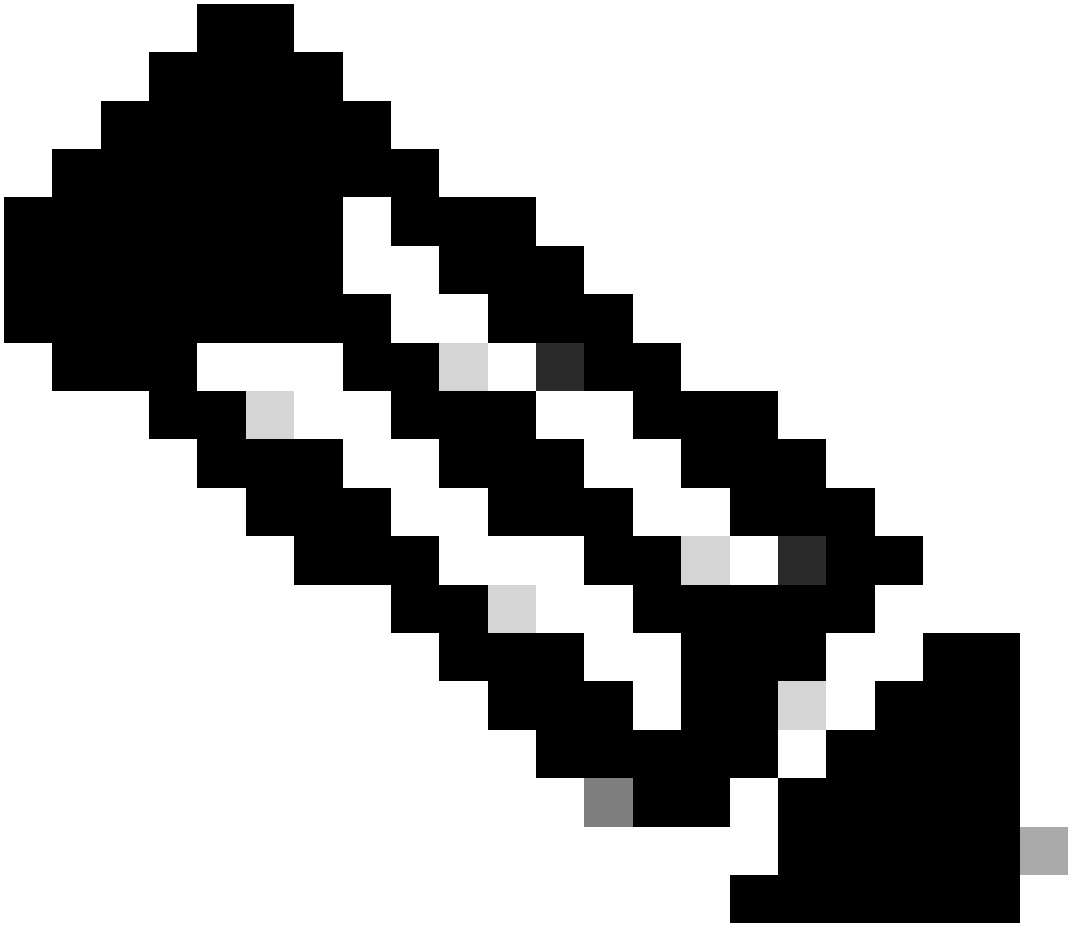
```
nodeInfo_url = "https://<CatalystCenterIP>/dna/intent/api/v1/nodes-config"
nodeInfo_headers = {
    'X-Auth-Token': token
}

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

La variable nodeInfo_url a été créée pour stocker l'URL de notre API. La variable nodeInfo_headers stocke les en-têtes de notre API. Dans ce cas, « X-Auth-Token : » et la variable de jeton ont été passés en tant que paramètres pour authentifier la demande avec succès à Catalyst Center. Enfin, la variable nodeInfoResponse stocke la réponse de l'API.

Pour valider la réponse reçue, vous pouvez utiliser la fonction print().

Exemple de sortie :

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



Remarque : si un certificat auto-signé est utilisé dans Catalyst Center, la requête API peut échouer avec l'erreur suivante :

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

Pour résoudre ce problème, vous devez ajouter le paramètre verify comme False à la requête. Cela supprime la vérification du certificat SSL à partir du terminal (Catalyst Center).

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

La réponse reçue de l'API peut être difficile à lire. Avec le module json(), la réponse peut être

imprimée dans une chaîne plus lisible. Tout d'abord, la réponse de l'API doit être chargée dans un objet JSON à l'aide de la fonction json.loads() suivie de la fonction json.dumps() :

```
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 : cette fonction renvoie l'objet JSON pris comme paramètre dans une chaîne au format JSON.

indent : ce paramètre définit le niveau d'indentation de la chaîne au format JSON.

Exemple de sortie :

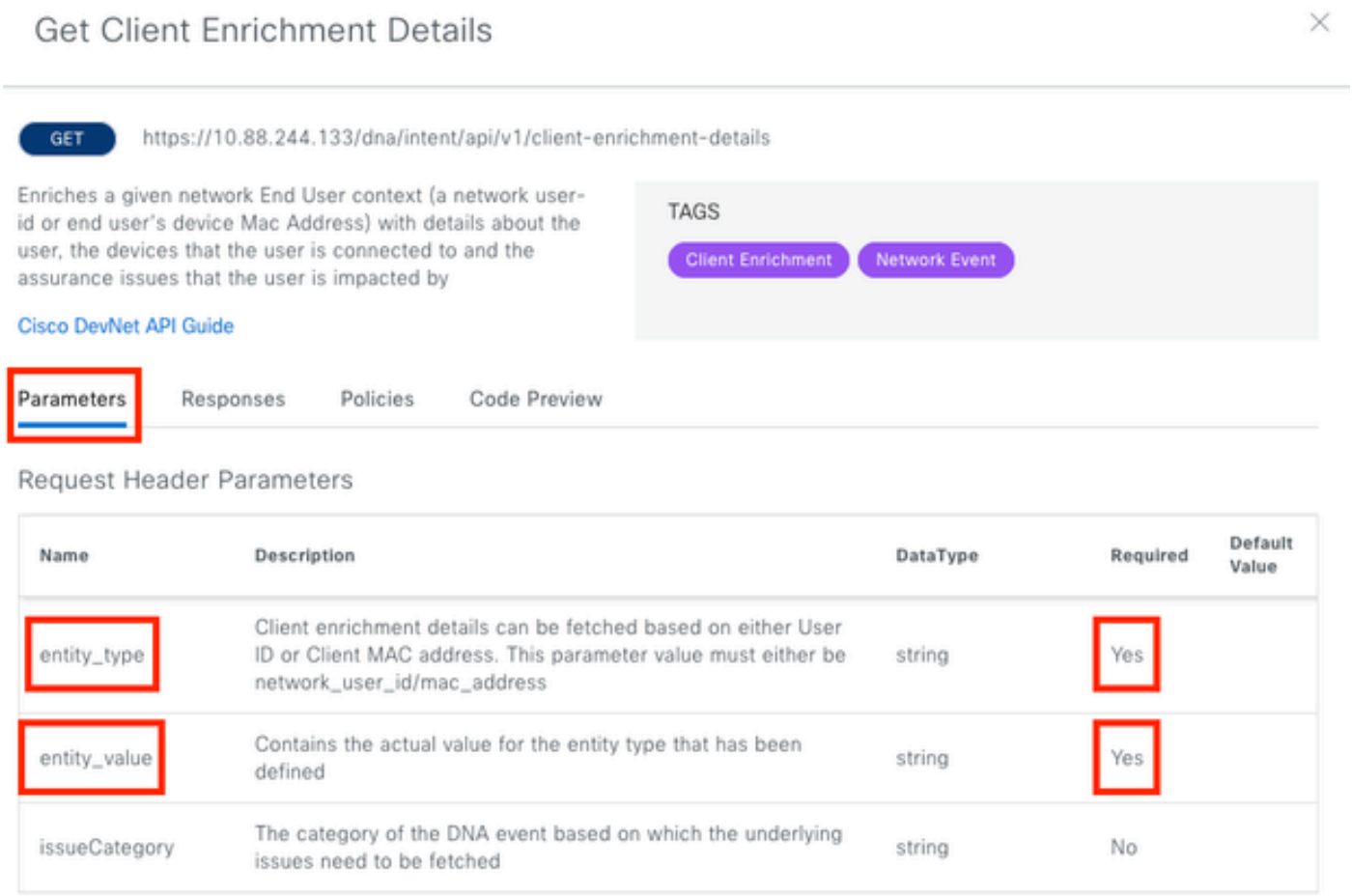
```
{
  "response": {
    "nodes": [
      {
        "name": "X.X.X.X",
        "id": "ea5dbec1-fbb6-4339-9242-7694eb1xXxX",
        "network": [
          {
            "slave": [
              "enp9s0"
            ],
            "lacp_supported": true,
            "intra_cluster_link": false,
!---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

API avec paramètres d'en-tête

Certaines API nécessitent l'envoi de certains paramètres dans l'en-tête pour fonctionner comme prévu. Dans ce cas, l'API Get Client Enrichment Details est testée.

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

Pour vérifier quels paramètres d'en-tête sont requis pour que l'API fonctionne comme prévu, accédez à Platform > Developer Toolkit > APIs > Get Client Enrichment Details et cliquez sur le nom de l'API. Une nouvelle fenêtre s'ouvre et sous l'option Parameters, Headers Parameters, les paramètres requis pour que l'API fonctionne sont affichés.



Dans ce cas, pour le paramètre entity_type, selon la description, la valeur peut être soit network_user_id soit mac_address et le paramètre entity_value doit contenir la valeur pour le type d'entité qui a été défini.

Pour continuer, deux nouvelles variables sont définies, entity_type et entity_value avec leurs valeurs correspondantes :

```
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
```

De nouvelles variables sont également créées pour exécuter l'appel d'API. L'URL de l'appel d'API est stockée dans la variable userEnrichment_url. Les en-têtes sont stockés dans la variable userEnrichmentHeaders. La réponse reçue est stockée dans la variable userEnrichmentResponse.

```
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)
```

Comme vous pouvez le voir, à partir de userEnrichmentHeaders, les variables entity_type et entity_value ont été passées en tant que paramètres d'en-tête pour l'appel d'API, avec la variable de jeton.

Pour valider la réponse reçue, utilisez la fonction print().

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

Exemple de sortie :


```
[ {
  "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
```


API avec paramètres de requête

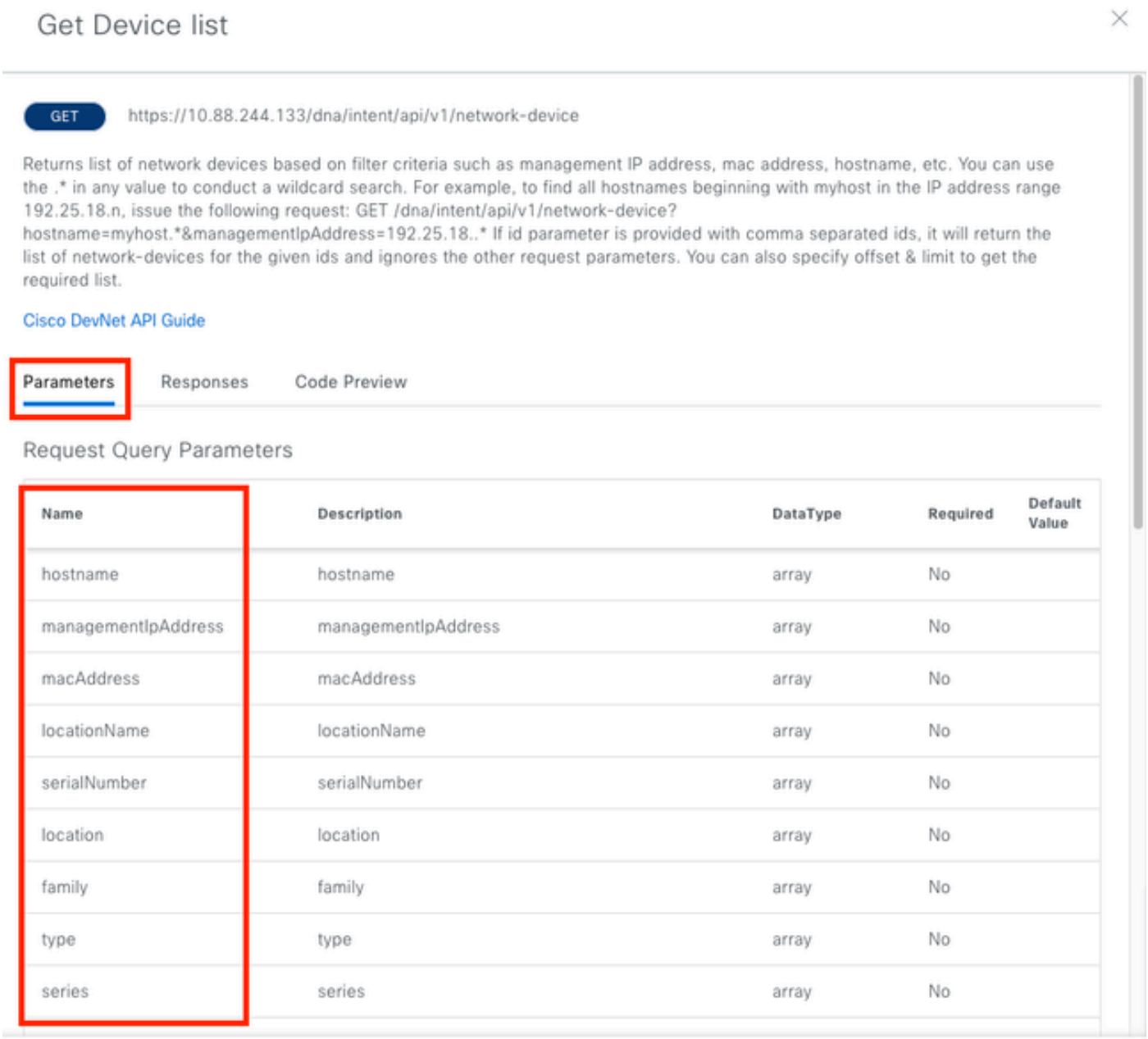
Les paramètres de requête peuvent être utilisés pour filtrer un nombre spécifique de résultats renvoyés par une API. Ces paramètres sont ajoutés à l'URL de l'API.

L'appel de l'API Get Device List est testé.

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

L'API Get Device List retourne une liste de tous les périphériques qui sont ajoutés dans Catalyst Center. Si des détails sont demandés pour un périphérique spécifique, les paramètres de requête peuvent aider à filtrer des informations spécifiques.

Pour vérifier quels paramètres de requête sont disponibles pour l'API, accédez à Platform > Developer Toolkit > APIs > Get Device List et cliquez sur le nom de l'API. Une nouvelle fenêtre est ouverte et sous l'option Parameters, les paramètres de requête disponibles pour l'API sont affichés.



Dans cet exemple, les paramètres de requête managementIpAddress et serialNumber sont utilisés (tenez compte du fait qu'il n'est pas nécessaire d'utiliser tous les paramètres de requête pour l'appel API). Créez et attribuez les valeurs correspondantes pour les deux paramètres de requête.

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

Comme mentionné ci-dessus, les paramètres de requête sont ajoutés dans l'URL de l'API, en particulier à la fin de celle-ci, à l'aide d'un « ? » suivi des paramètres de requête.

Si plusieurs paramètres de requête sont utilisés, un signe '&' est placé entre eux pour former ce qui est appelé une chaîne de requête.

L'exemple suivant montre comment ajouter les paramètres de requête à la variable deviceListUrl qui stocke l'URL de l'appel API.

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

Notez que les variables précédemment créées ont été ajoutées à la chaîne d'URL. En d'autres termes, la chaîne entière de l'URL ressemble à ceci :

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

Poursuivez avec l'appel d'API, la variable deviceListHeaders est créée pour stocker les en-têtes d'API avec la variable de jeton passée comme paramètre et la variable deviceListResponse stocke la réponse d'API.

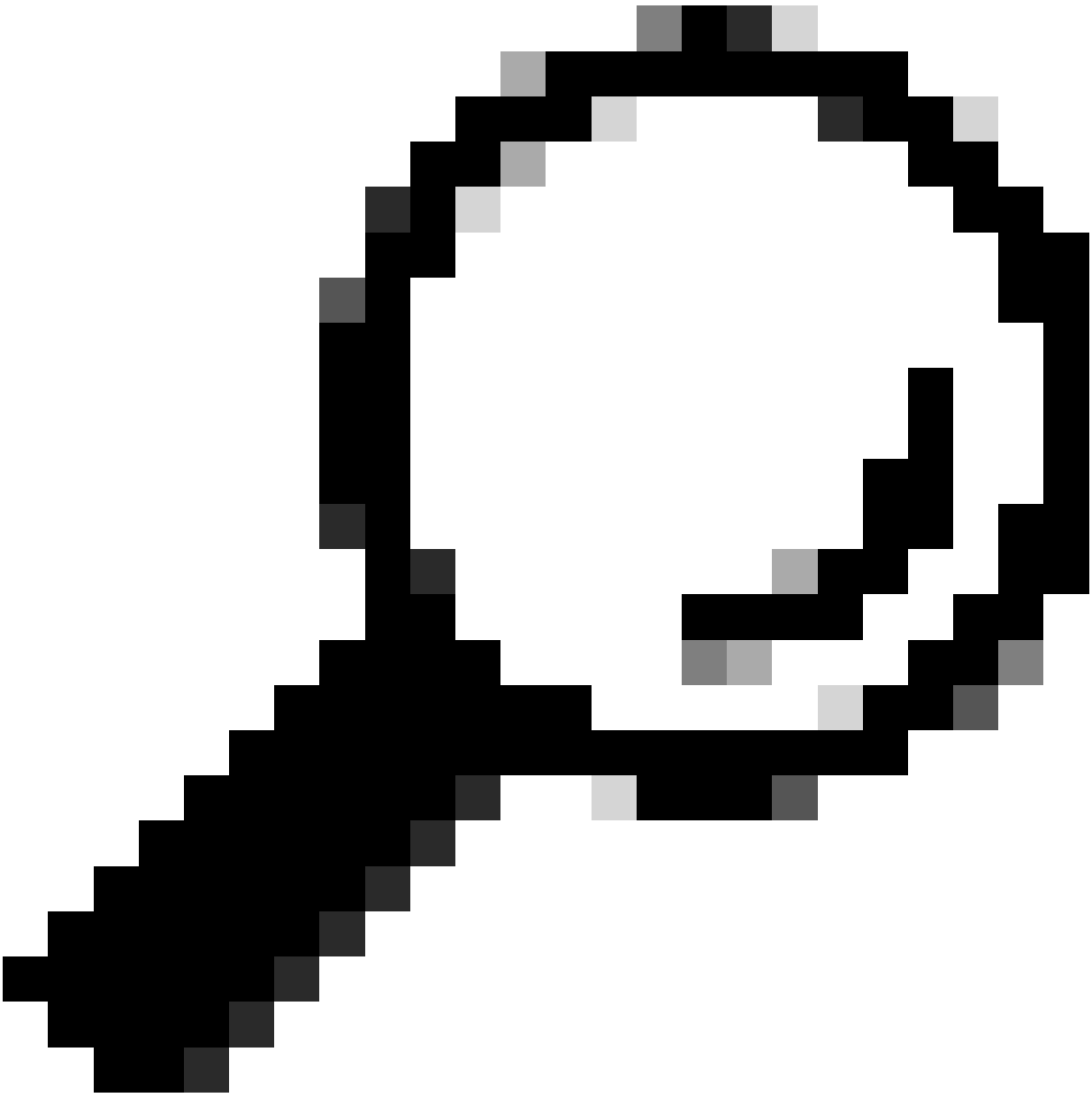
```
deviceListHeaders = {
            'X-Auth-Token': token,
        }

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

Pour valider la réponse reçue, vous pouvez utiliser la fonction print().

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

Exemple de sortie :

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



Conseil : pour imprimer la réponse de manière plus lisible, vous pouvez utiliser les fonctions json.loads() et json.dumps() décrites dans la section Test de l'API.

À propos de cette traduction
Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.