Usar APIs do Catalyst Center com Python

Contents

Introdução

Este documento descreve como usar as diferentes APIs disponíveis no Cisco Catalyst Center usando Python.

Pré-requisitos

Requisitos

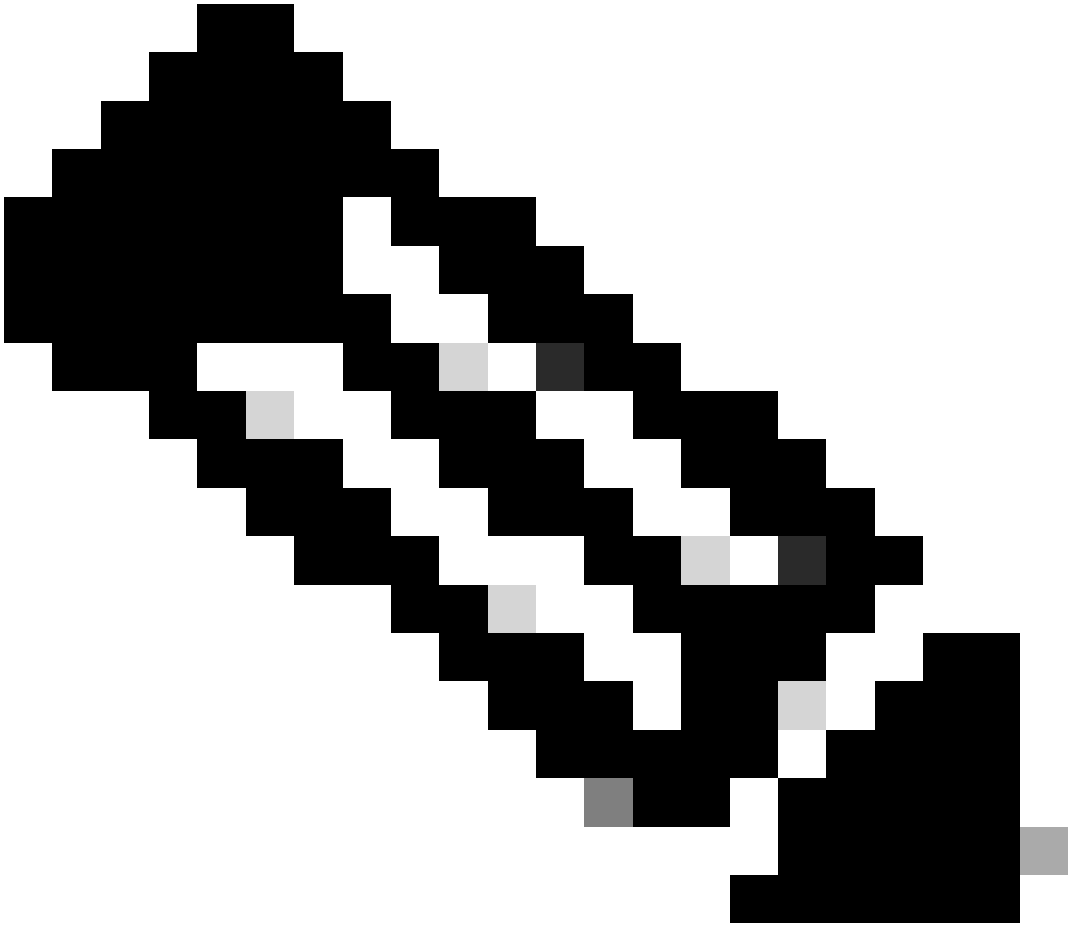
Conhecimento básico sobre:

- Cisco Catalyst Center
- APIs
- Python

Componentes Utilizados

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

As informações neste documento foram criadas a partir de dispositivos em um ambiente de laboratório específico. Todos os dispositivos utilizados neste documento foram iniciados com uma configuração (padrão) inicial. Se a rede estiver ativa, certifique-se de que você entenda o impacto potencial de qualquer comando.

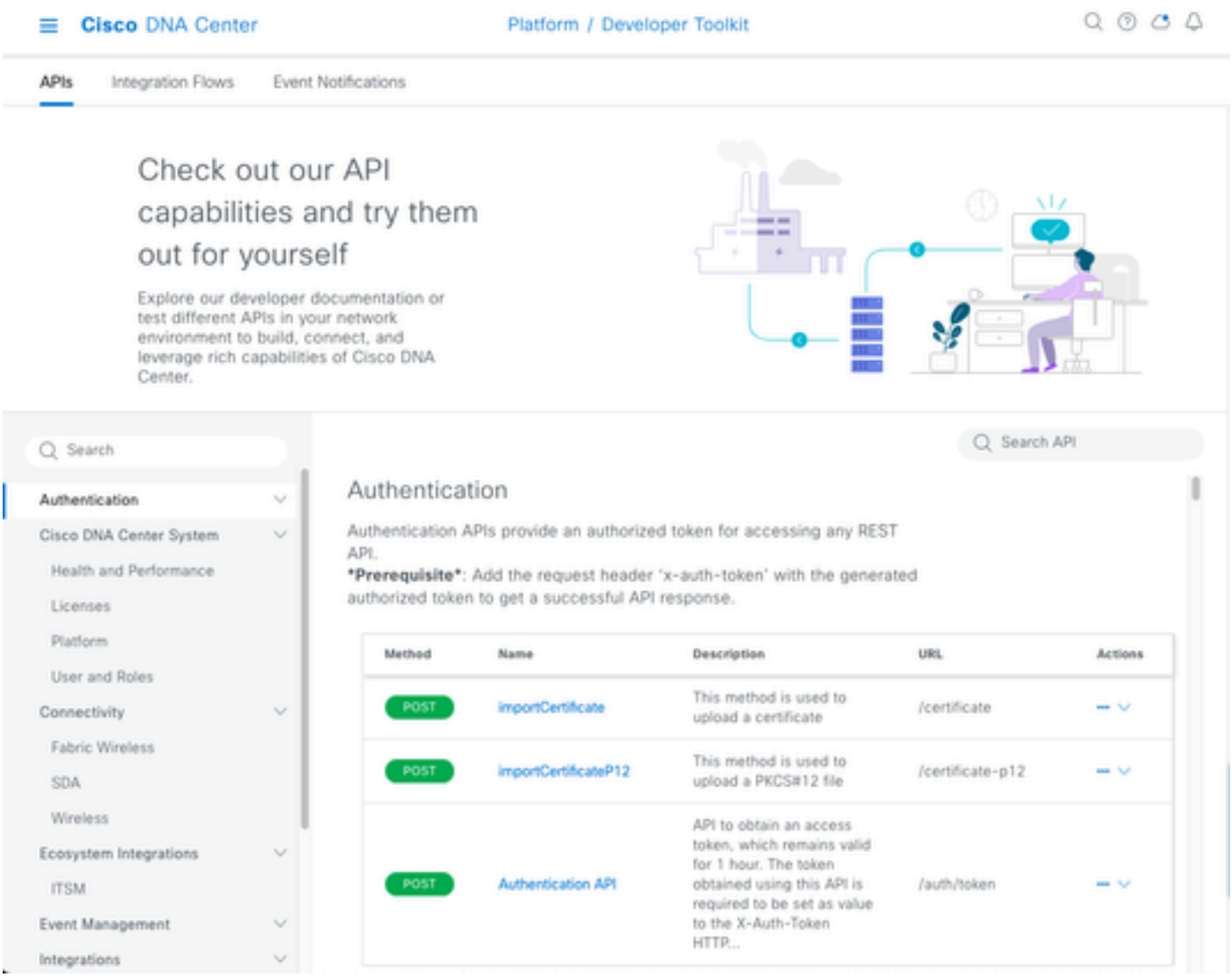


Observação: o Cisco Technical Assistance Center (TAC) não fornece suporte técnico para Python. Se você tiver problemas com o Python, entre em contato com o Suporte do Python para obter assistência técnica.

Configurar

Overview

O Cisco Catalyst Center tem muitas APIs disponíveis. Para verificar quais APIs podem ser usadas, no Catalyst Center, navegue para Plataforma > Developer Toolkit > APIs.

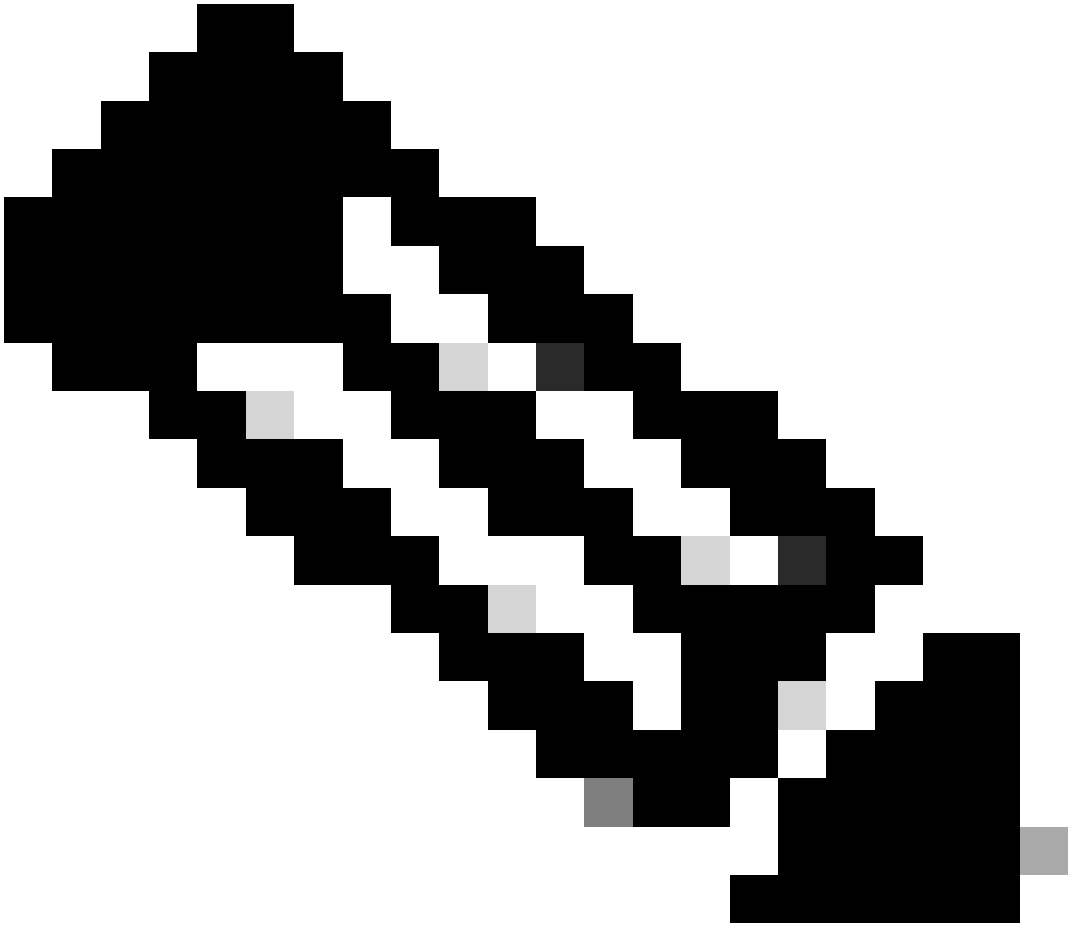


Página de APIs do Catalyst Center

Cada API tem sua própria finalidade, dependendo das informações ou da ação que precisa ser executada no Catalyst Center. Para que as APIs funcionem, como pré-requisito, um token deve ser usado para autenticar-se corretamente no Catalyst Center e obter uma resposta de API bem-sucedida. O token identifica os privilégios para o chamador REST de acordo.

Também é importante identificar os componentes que compõem uma API, que são os seguintes:

- URL: Endpoint que fornece acesso a um recurso específico.
- Método: todas as APIs devem incluir um método. Ele define a ação/operação que o cliente gostaria de executar para o endpoint específico. Exemplos: POST, GET, PUT, DELETE.
- Cabeçalho: Fornece informações adicionais sobre a solicitação no formato de pares chave-valor. Por exemplo, o cabeçalho de autorização fornece um método de autenticação usando credenciais.
- Parâmetros: variáveis que fornecem instruções específicas ao ponto final usando a API. Os parâmetros podem fazer parte da URL do ponto final.
- Carga: Dados que precisam ser enviados ao ponto final durante a chamada à API.

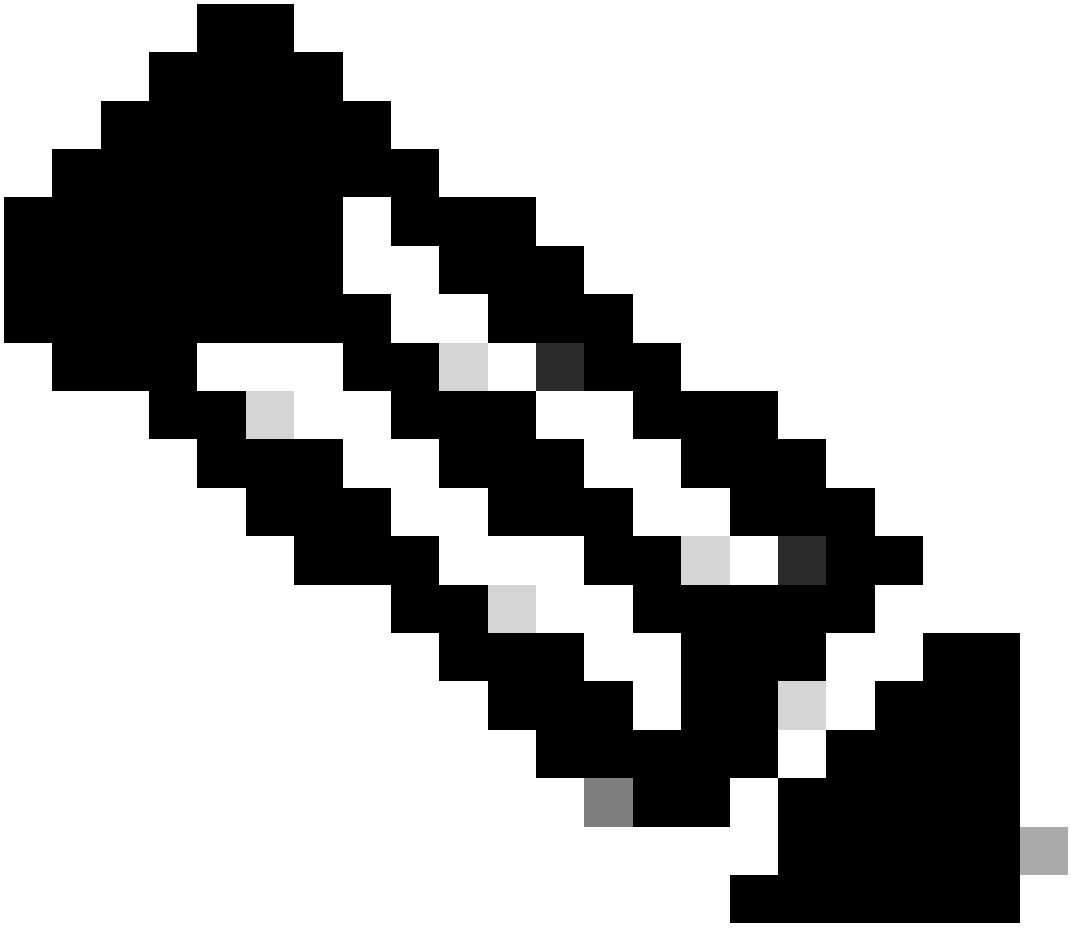


Observação: para obter informações mais detalhadas sobre cada API disponível no Catalyst Center, consulte o guia de Referência de API.

Módulos

Módulos Python usados:

- solicitações:Este módulo permite enviar solicitações HTTP/1.1 para URLs específicos. Para obter mais informações sobre o módulo, consulte o guia do módulo de solicitação.
- base64: Fornece funções de codificação e decodificação. Para obter mais informações sobre o módulo, consulte o guia do módulo base64.
- json: este módulo permite obter dados específicos da resposta de APIs. Para obter mais informações sobre o módulo, consulte o guia do módulo json.



Observação: para obter mais informações sobre como instalar módulos Python, consulte a documentação Instalação de módulos Python.

Gerar token

A API chamada Authentication API deve ser usada para gerar um novo token.

API de autenticação:

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

É importante mencionar que o token gerado é válido por 1 hora. Após 1 hora, um novo token deve ser gerado usando a mesma API mencionada acima.

Em um novo arquivo Python, importe os módulos (requests, base64 e json) seguidos da criação

de quatro variáveis:

```
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"
```

A API de autenticação oferece suporte à Autenticação Básica como Token de Autorização no cabeçalho. A Autenticação Básica é um método que pode ser usado para autenticar um endpoint, fornecendo um nome de usuário e uma senha separados por dois-pontos (username:password). Ambos os valores são codificados na base64, e o endpoint decodifica as credenciais de login e verifica se o usuário pode acessar ou não.

Para criar a string codificada na Base64 para seu nome de usuário e senha, o módulo base64 é usado. Para fazer isso, a função b64encode é usada.

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

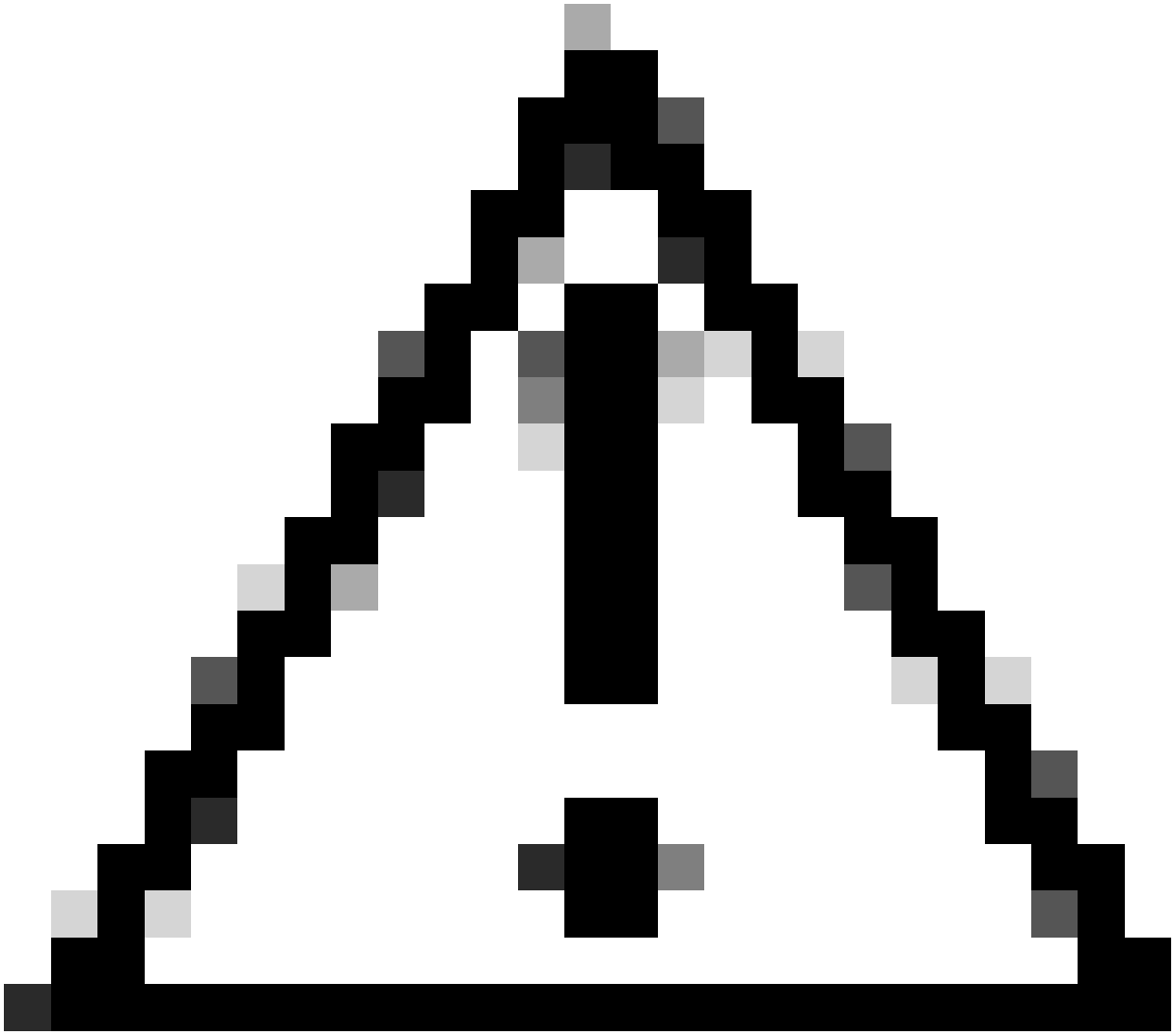
A partir do código acima, uma variável byte_string foi criada usando a função '.encode("ascii")'. Isso ocorre porque a função base64.b64encode requer um objeto semelhante a bytes. Observe também que as variáveis user e password foram usadas para manter o formato de string 'user:password'. Finalmente, uma string de byte codificada na base64 foi criada com o usuário e a senha. Usando o método 'decode()', o valor foi convertido no objeto str.

Para verificá-lo, você pode imprimir o valor para a variável authorizationBase64:

```
print(authorizationBase64)
```

Exemplo de saída:

```
am9yZ2QhbDI6Sm9yZ2VhbDXxXxXx
```



Cuidado: base64 não é um algoritmo de criptografia. Ele não deve ser usado para fins de segurança. A API de autenticação também oferece suporte à criptografia de chave AES como token de autorização no cabeçalho que fornece mais segurança.

Agora que uma string codificada na base64 foi criada usando o usuário e a senha para autenticação no Catalyst Center, é hora de continuar com a chamada da API Authentication usando as solicitações do módulo. Além disso, a função chamada request permite, para obter um objeto de resposta que contém o texto da solicitação.

Sintaxe do método:

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

**kwargs significa qualquer parâmetro transmitido para a solicitação, por exemplo, cookies, agentes de usuário, payload, cabeçalhos, etc.

A API de autenticação especifica que o método é POST, a URL é "/dna/system/api/v1/auth/token" e a autenticação básica precisa ser especificada no cabeçalho.

Essas variáveis são criadas para usá-las para a função request().

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

Para a variável headers, duas coisas foram especificadas. O primeiro é o tipo de conteúdo, que especifica o tipo de mídia do recurso enviado ao ponto final (isso ajuda o ponto final a analisar e processar os dados com precisão). O segundo é Authorization, que, neste caso, a variável authorizationBase64 (que armazena nossa string base64-econded) é enviada como parâmetro para autenticar o Catalyst Center.

Agora, continue a usar a função request() para realizar a chamada à API. O próximo código mostra a sintaxe da função:

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

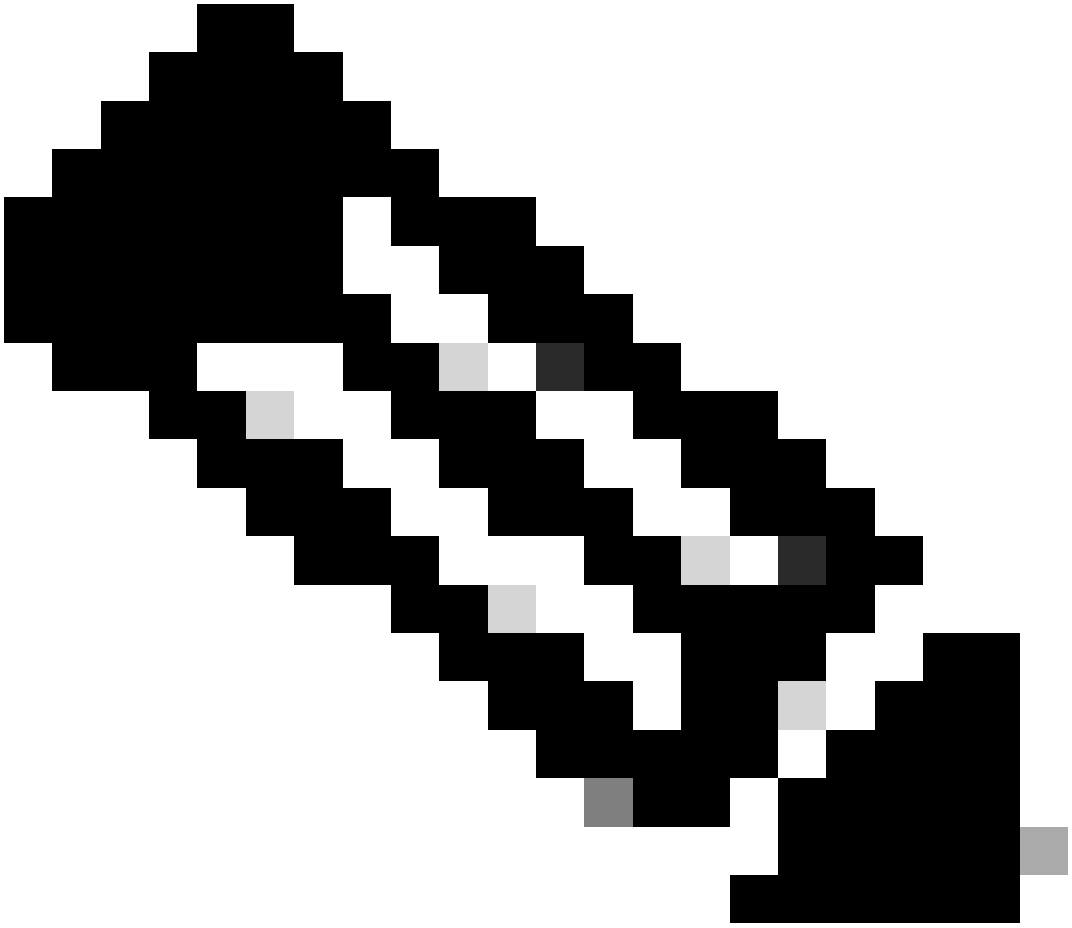
A variável response foi criada para armazenar os dados da chamada API feita.

Para imprimir a resposta obtida, use a função print junto com o método text() na variável response. O método text() gera um objeto str com a resposta recebida do Catalyst Center.

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

Exemplo de saída:

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



Observação: se o Catalyst Center estiver usando um certificado autoassinado, a solicitação de API poderá falhar com o próximo erro:

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

Para corrigir esse problema, você precisa adicionar o parâmetro verify como False à função request. Isso ignora a verificação do certificado SSL do endpoint (Catalyst Center).

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

A partir da resposta recebida da chamada API authentication, observe que a estrutura é semelhante a um dicionário em Python, no entanto, é um objeto str.

Para validar o tipo de um objeto, use a função type().

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

O que retorna a próxima saída:

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

Para fins práticos, somente o valor do token precisa ser extraído da resposta recebida da API, não da sequência inteira, pois, para usar as outras APIs do Catalyst Center, somente o token deve ser passado como parâmetro.

Como a resposta recebida da chamada da API tem uma estrutura semelhante a um dicionário em Python, mas o tipo de objeto é str, o objeto precisa ser convertido em um dicionário usando o módulo json. Isso extrai o valor do token de toda a sequência de caracteres recebida da API.

Para fazer isso, a função json.loads() converte a string em um dicionário para extrair posteriormente apenas o valor do token e atribuí-lo diretamente à nossa variável token.

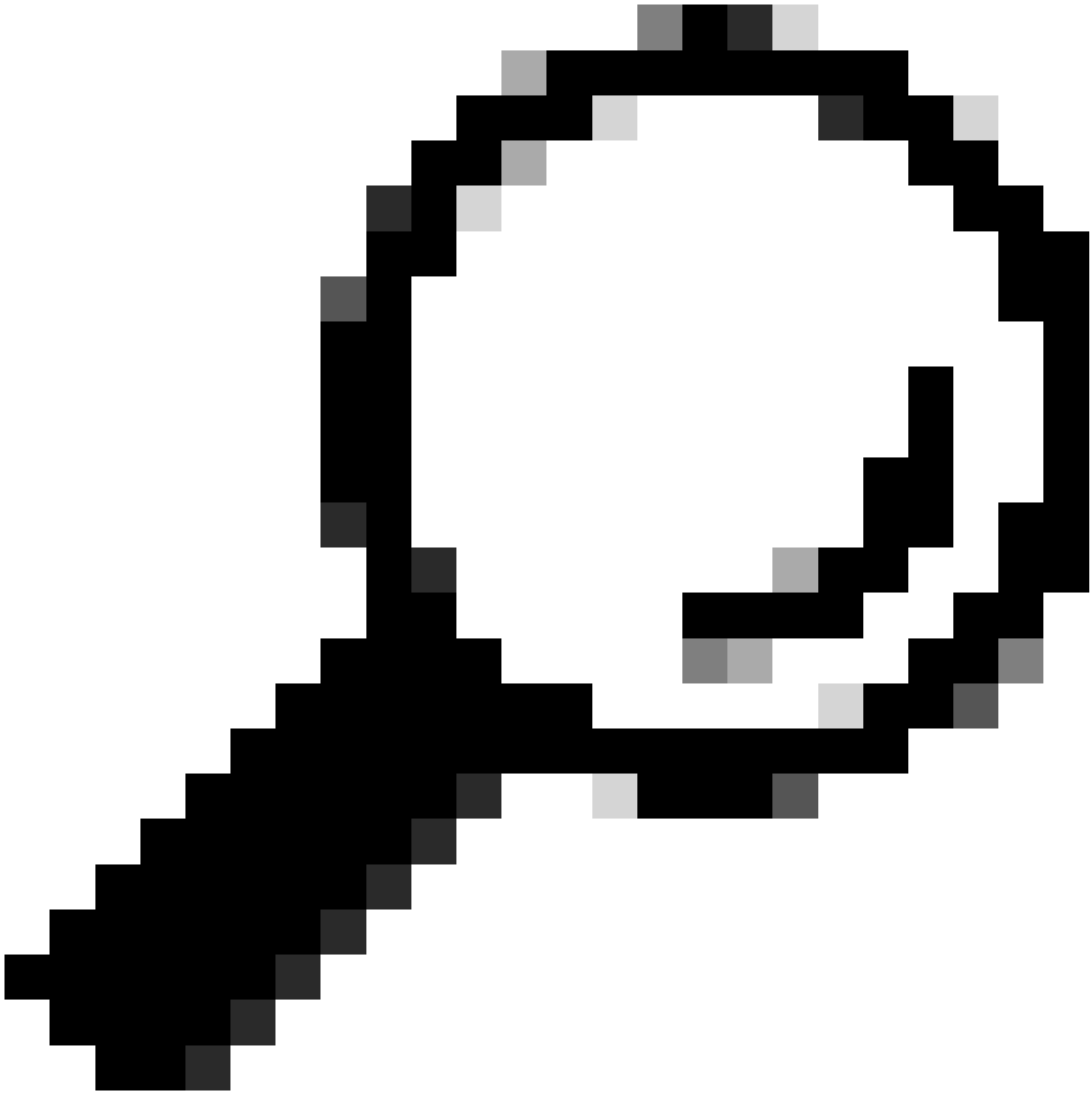
```
token = json.loads(response.text)  # Converting the response.text string value into a dictionary (It is
token = (token["Token"])  # Extracting just the token value by specifying the key as a parameter.
```

Para verificar se a variável token tem apenas o token como seu valor, continue para imprimi-lo.

```
print(token)
```

Exemplo de saída:

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



Dica: como cada token gerado expira em 1 hora por padrão, um método Python que contém o código para gerar um token pode ser criado e chamado toda vez que um token expira, sem ter que executar o programa inteiro apenas chamando o método criado.

Testando uma API

Agora que o token foi atribuído com êxito à variável token, as APIs do Catalyst Center disponíveis podem ser usadas.

Nesse caso, a API Cisco DNA Center Nodes Configuration Summary é testada.

Resumo da configuração de nós do Cisco DNA Center

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

Essa API fornece detalhes sobre a configuração atual do Catalyst Center, como servidor NTP configurado, nome do nó, link dentro do cluster, modo LACP e assim por diante.

A API de resumo da configuração de nós do Cisco DNA Center especifica, neste caso, que o método usado é GET, a URL é "/dna/intent/api/v1/nodes-config" e, como a sequência de token foi extraída e atribuída à variável token, desta vez o token é passado como uma variável no cabeçalho da chamada de API como ‘X-Auth-Token’: seguido pelo token.

Isso autentica a solicitação ao Catalyst Center para cada chamada de API que é executada. Lembre-se de que cada token dura 1 hora. Após 1 hora, um novo token deve ser gerado para continuar fazendo chamadas de API para o Catalyst Center.

Continue para criar as variáveis para testar a API:

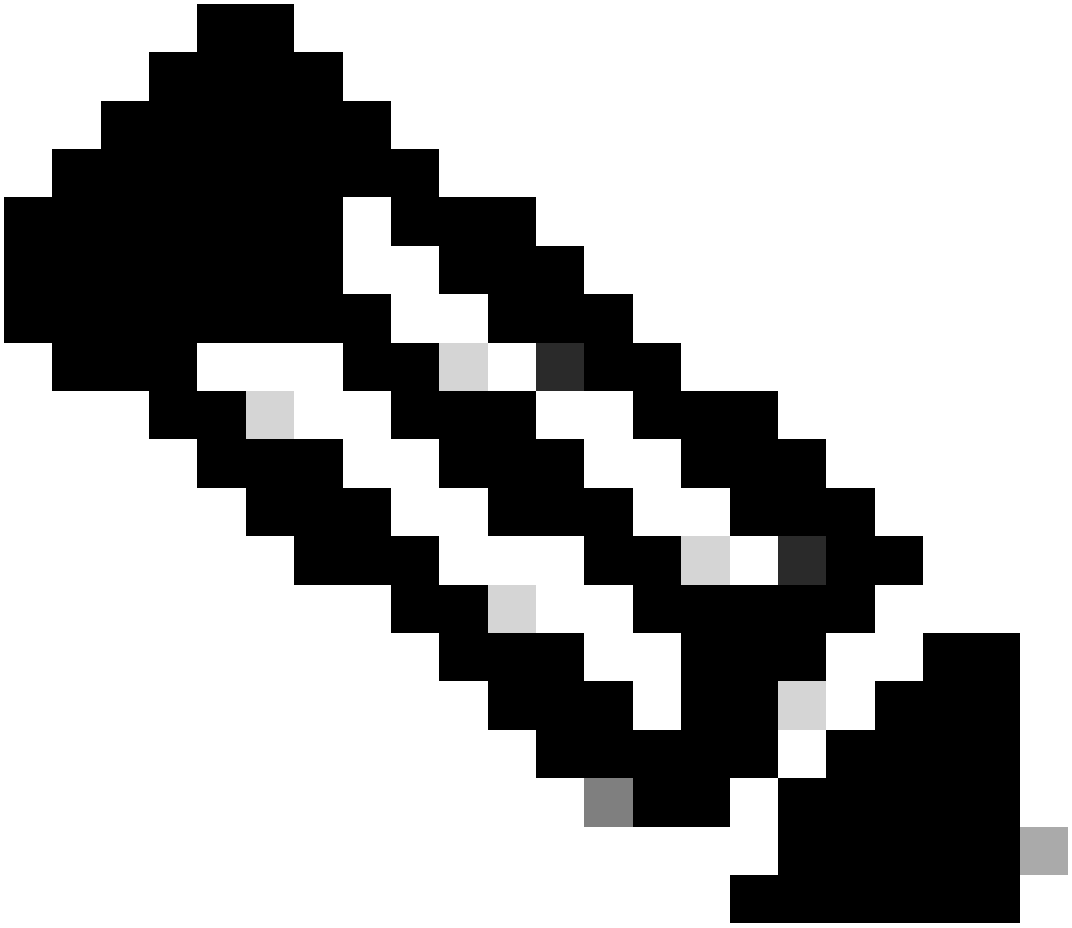
```
nodeInfo_url = "https://<CatalystCenterIP>/dna/intent/api/v1/nodes-config"
nodeInfo_headers = {
    'X-Auth-Token': token
}

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

A variável nodeInfo_url foi criada para armazenar a URL da nossa API. A variável nodeInfo_headers armazena os cabeçalhos da nossa API. Nesse caso, ‘X-Auth-Token:’ e a variável token foram passados como parâmetros para autenticar a solicitação com êxito ao Catalyst Center. Finalmente, a variável nodeInfoResponse armazena a resposta da API.

Para validar a resposta recebida, você pode usar a função print().

Exemplo de saída:

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



Observação: caso um certificado autoassinado esteja sendo usado no Catalyst Center, a solicitação de API poderá falhar com o próximo erro:

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

Para corrigir esse problema, você precisa adicionar o parâmetro verify como False à solicitação. Isso suprime a verificação do certificado SSL do endpoint (Catalyst Center).

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

A resposta recebida da API pode ser difícil de ler. Usando o módulo json(), a resposta pode ser impressa em uma string mais legível. Primeiro, a resposta da API deve ser carregada em um

objeto JSON usando a função json.loads() seguida pela função 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: Esta função retorna o objeto JSON tomado como um parâmetro em uma string formatada JSON.

Indentação: este parâmetro define o nível de indentação para a string formatada JSON.

Exemplo de saída:

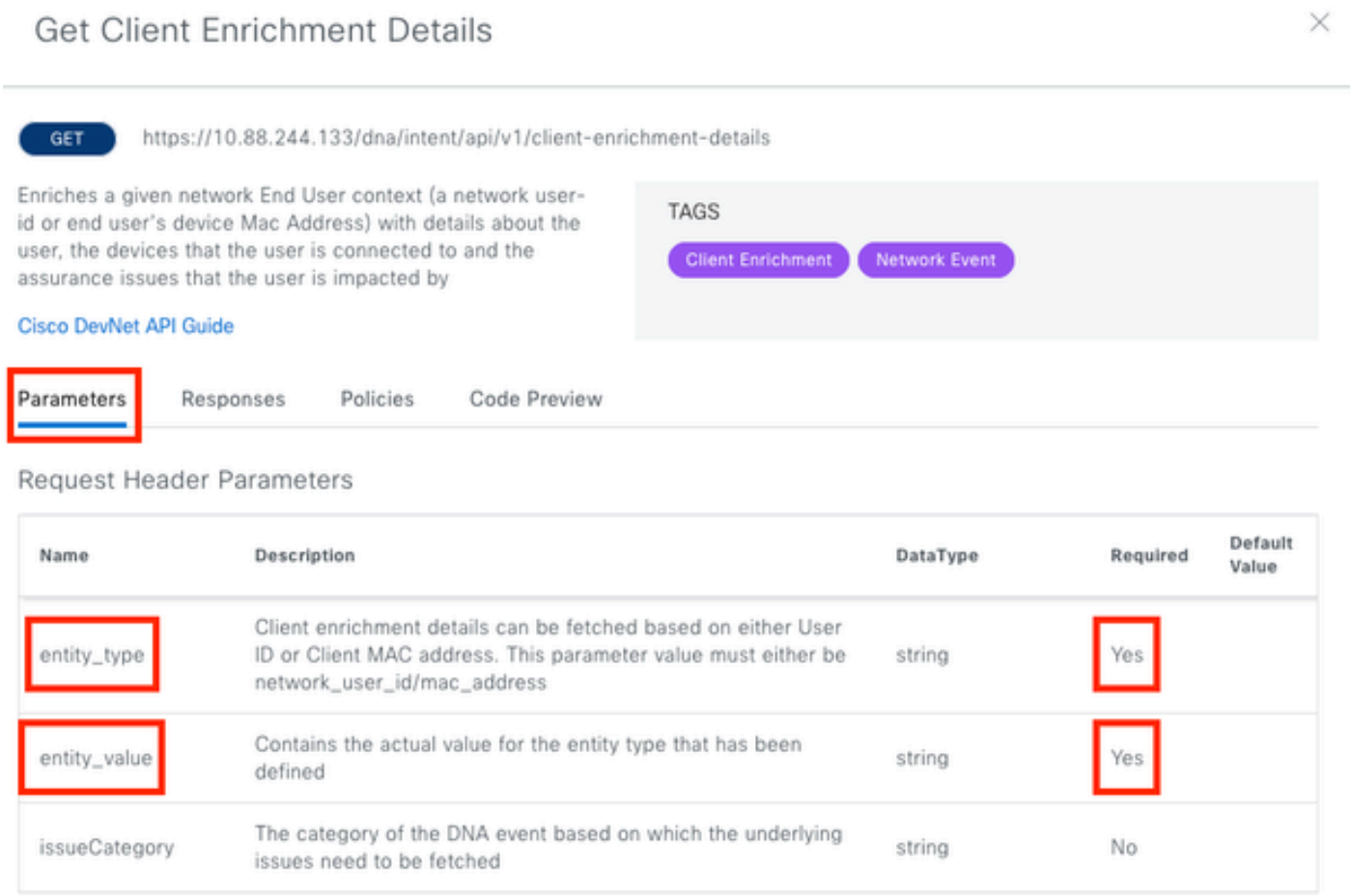
```
{
  "response": {
    "nodes": [
      {
        "name": "X.X.X.X",
        "id": "ea5dbec1-fbb6-4339-9242-7694eb1xXxX",
        "network": [
          {
            "slave": [
              "enp9s0"
            ],
            "lacp_supported": true,
            "intra_cluster_link": false,
!---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

APIs com parâmetros de cabeçalho

Há algumas APIs que exigem que alguns parâmetros sejam enviados no Cabeçalho para funcionar como esperado. Nesse caso, a API Get Client Enrichment Details é testada.

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

Para verificar quais Parâmetros de Cabeçalhos são necessários para que a API funcione como esperado, navegue para Plataforma > Developer Toolkit > APIs > Obter Detalhes de Enriquecimento do Cliente e clique no nome da API. Uma nova janela é aberta e, na opção Parameters, os parâmetros de cabeçalhos necessários para que a API funcione são exibidos.



Nesse caso, para o parâmetro entity_type, de acordo com a descrição, o valor pode ser network_user_id ou mac_address e o parâmetro entity_value deve conter o valor para o tipo de entidade que foi definido.

Para continuar com ele, duas novas variáveis são definidas, entity_type e entity_value com seus valores correspondentes:

```
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
```

Novas variáveis também são criadas para executar a chamada à API. A URL da chamada API é armazenada na variável userEnrichment_url. Os cabeçalhos são armazenados na variável userEnrichmentHeaders. A resposta recebida está armazenada na variável 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)
```

Como você pode ver, a partir de userEnrichmentHeaders, as variáveis entity_type e entity_value foram passadas como parâmetros Header para a chamada API, junto com a variável token.

Para validar a resposta recebida, use a função print().

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

Exemplo de saída:


```
[ {
  "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
```


APIs com parâmetros de consulta

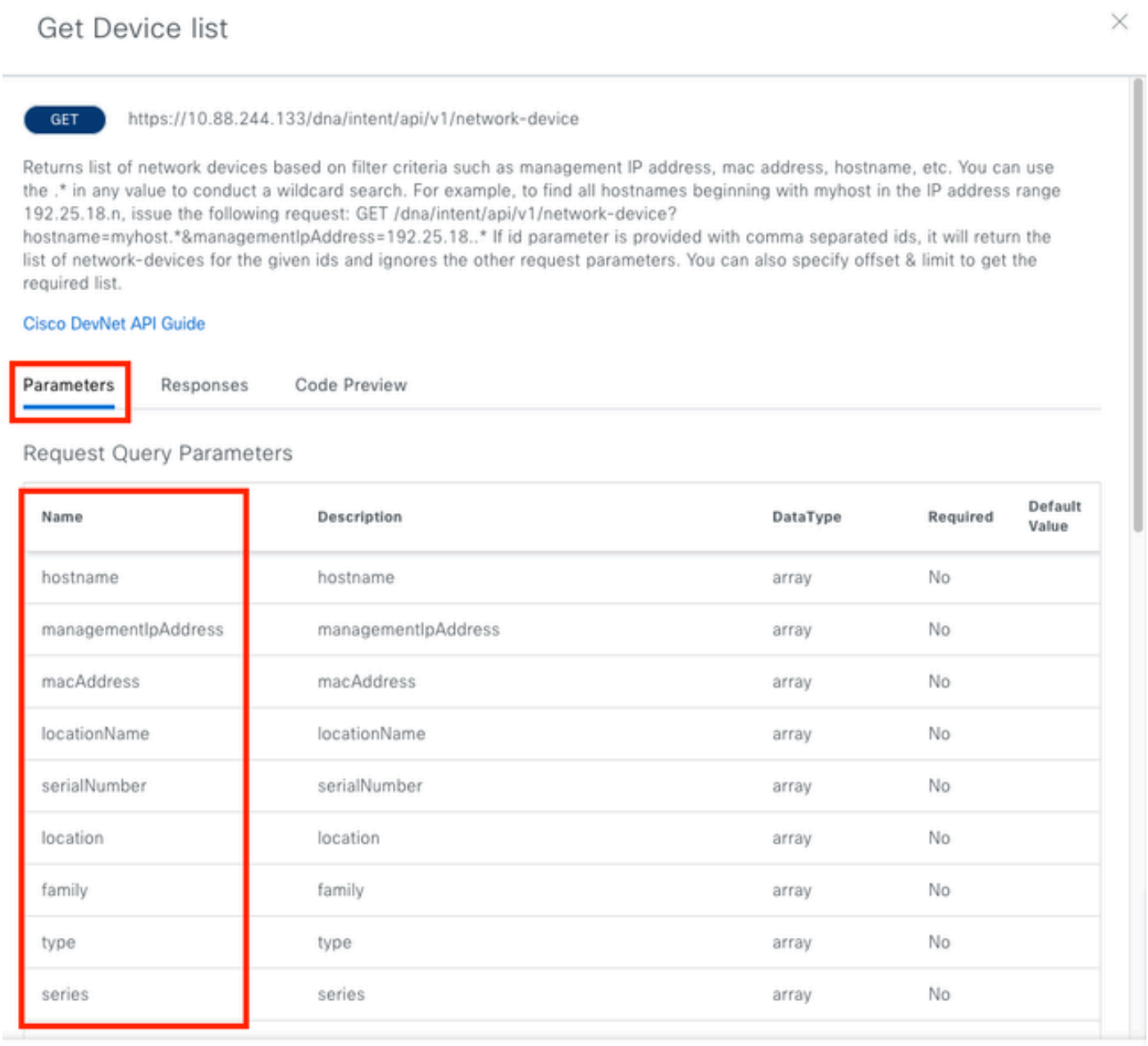
Os parâmetros de consulta podem ser usados para filtrar um número específico de resultados retornados por uma API. Esses parâmetros são adicionados à URL da API.

A chamada à API Get Device List foi testada.

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

A API Get Device List retorna uma lista de todos os dispositivos adicionados ao Catalyst Center. Se forem solicitados detalhes de um dispositivo específico, os parâmetros de consulta podem ajudar a filtrar informações específicas.

Para verificar quais parâmetros de consulta estão disponíveis para a API, navegue para Platform > Developer Toolkit > APIs > Get Device List e clique no nome da API. Uma nova janela é aberta e, na opção Parâmetros, os parâmetros de consulta disponíveis para a API são exibidos.



Neste exemplo, os parâmetros de consulta managementIpAddress e serialNumber são usados

(leve em consideração que não é necessário usar todos os parâmetros de consulta para a chamada de API). Prossiga para criar e atribuir os valores correspondentes para ambos os parâmetros de consulta.

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

Como foi mencionado acima, os parâmetros de consulta são adicionados na URL da API, especificamente no final da mesma, usando um ‘?’ seguido pelos parâmetros de consulta.

No caso de vários parâmetros de consulta serem usados, um sinal "&" é colocado entre eles para formar o que é chamado de sequência de caracteres de consulta.

O próximo exemplo mostra como adicionar os parâmetros de consulta à variável deviceListUrl que armazena a URL da chamada de API.

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

Observe que as variáveis criadas anteriormente foram anexadas à string de URL. Em outras palavras, a string inteira da URL se parece com isto:

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

Continue com a chamada de API, a variável deviceListHeaders é criada para armazenar os Cabeçalhos de API junto com a variável token passada como parâmetro e a variável deviceListResponse armazena a resposta de API.

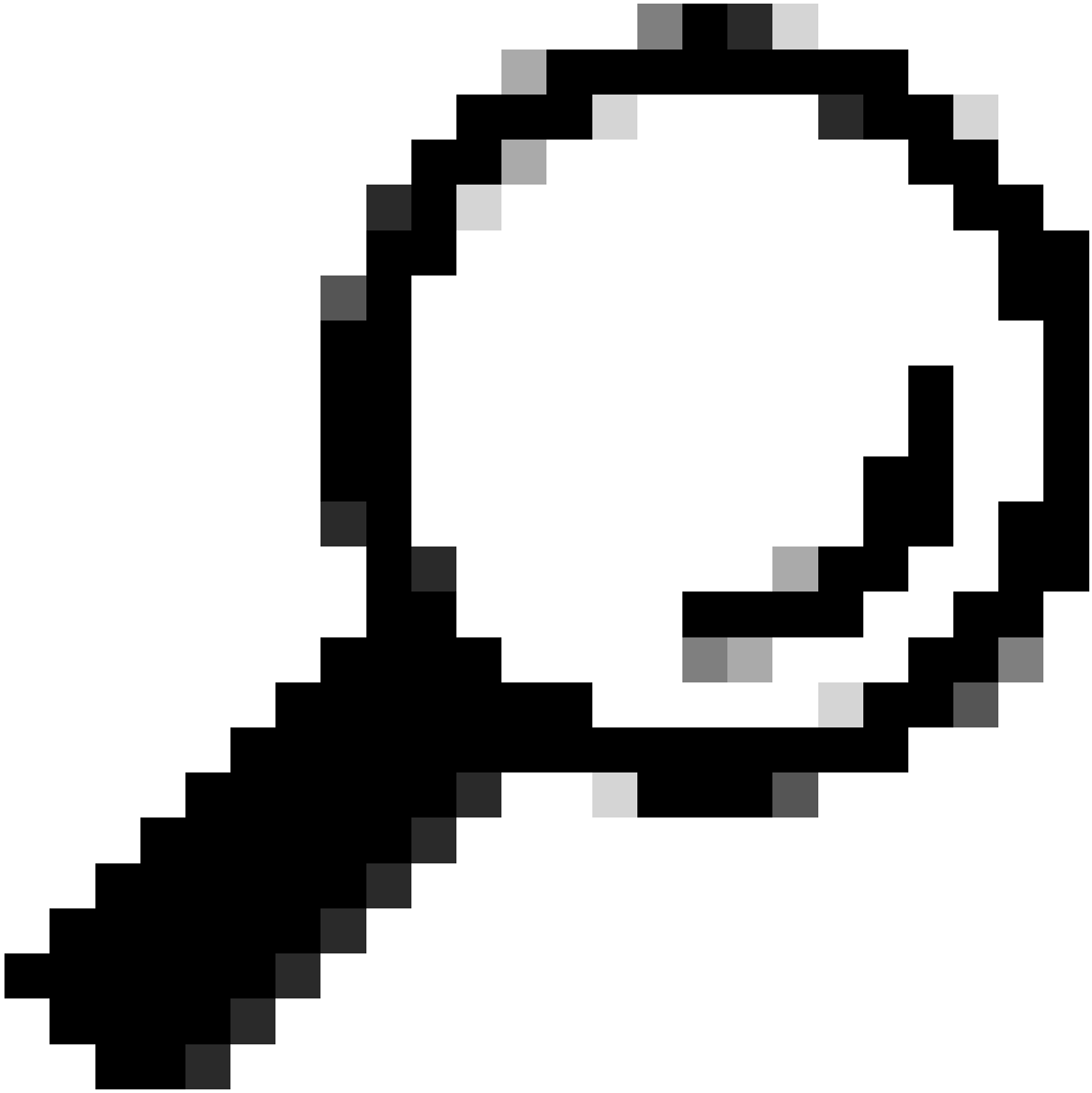
```
deviceListHeaders = {
            'X-Auth-Token': token,
        }

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

Para validar a resposta recebida, você pode usar a função print().

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

Exemplo de saída:

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



Dica: para imprimir a resposta de uma maneira mais legível, você pode usar as funções json.loads() e json.dumps() descritas na seção Testing API.

Sobre esta tradução
A Cisco traduziu este documento com a ajuda de tecnologias de tradução automática e humana para oferecer conteúdo de suporte aos seus usuários no seu próprio idioma, independentemente da localização.

Observe que mesmo a melhor tradução automática não será tão precisa quanto as realizadas por um tradutor profissional.

A Cisco Systems, Inc. não se responsabiliza pela precisão destas traduções e recomenda que o documento original em inglês (link fornecido) seja sempre consultado.