Crosswork Cloud API

Crosswork Cloud API の概要

Crosswork Cloud API は、ネットワーク管理および運用アプリケーションで API を使用するプログラマ向けです。

Crosswork Cloud Network Insights API により、プレフィックスまたは ASN への登録、通知エンドポイントの設定、アラームがトリガーされる条件の指定などの設定タスクを実行できます。Crosswork Cloud Traffic Analysis API はトラフィック統計を取得します。

API ヘルプおよびドキュメント

Crosswork Cloud API ドキュメントにアクセスするには、Crosswork Cloud にログインする必要があります。API コールの定義とドキュメントを表示するには、[ヘルプとサポート（Help and Support）]アイコン > [API] に移動するか、https://crosswork.cisco.com/apiDoc/CiscoCrossworkCloudAPI [英語] に移動します。

こちらからシスココミュニティに参加して、Crosswork Developer Hub にアクセスしてください。[ヘルプとサポート（Help and Support）]アイコン > [サポート（Support）] > [コミュニティフォーラム（Community Forum）] に移動して、シスココミュニティにアクセスすることもできます。Crosswork Cloud ディスカッションを識別しやすくするために、必ず「Crosswork」ラベルを使用して登録してください。

API の使用開始

Crosswork Cloud API にアクセスするには、管理者権限が必要です。管理者権限がない場合、API オプションは表示されません。ユーザー権限の変更については、ユーザ権限の変更を参照してください。

API コールの定義とドキュメントを表示するには、Crosswork Cloud にログインし、[ヘルプとサポート（Help & Support）] > [API] をクリックするか、または https://crosswork.cisco.com/apiDoc/CiscoCrossworkCloudAPI にアクセスする必要があります。

API の使用を開始するには、次のタスクを実行します。

ステップ 1 API キーを要求するには、Crosswork Cloud Network Insights ウィンドウの右上隅にあるユーザのイニシャルをクリックしてから、[APIキー/トークン（API Key / Tokens）] をクリックします。

ステップ 2 [APIキーの追加（Add API Key）] をクリックします。

ステップ 3 API キーの名前、説明（任意）、および API キーの開始日と終了日を入力し、[保存（Save）] をクリックします。

ステップ 4 [作成（Create）] をクリックします。

新しい API キーが作成され、Crosswork Cloud アプリケーションでキーの詳細が表示されます。これは、キーが表示される唯一の機会です。

ステップ 5 [コピー（Copy）] をクリックして API キーをコピーし、安全な場所に保存できるようにします。

（注） API キーをパスワードのように保護します。API キーはアカウントへのアクセスを提供するため、必ず安全に保管してください。

ステップ 6 使用を開始する方法の例については、Crosswork Cloud Network Insights クライアントスクリプト例 （4 ページ）およびCrosswork トラフィック分析クライアントスクリプトの例 （10 ページ）のセクションを参照してください。

API キーの定義

Crosswork Cloud API キーは次の設定は次のとおりです。

- API キーは、16 進数で符号化された 32 バイトの対称キーです。クライアント アプリケーションは API キーを使用して、Crosswork Cloud Network Insights または Crosswork Cloud Traffic Analysis 宛ての REST API 要求に署名します。
- API キー識別子（ID）は、キーの一意の値であり、署名済みの各要求に含める必要があります。Crosswork Cloud サービスは、キー ID を使用して API キーのコピーを取得し、着信要求を確認します。

（注） パスワードと同じようにAPIキーを保護します。APIキーはアカウントへのアクセスを提供するため、必ず安全に保管してください。

クライアントアプリケーションはAPIキーを使用して、Crosswork Cloudに送信されるすべての要求に署名します。各要求には以下にものが含まれます。

- 署名要求
- API キー ID
- 署名を決定するために使用されるフィールドの詳細を示すメタデータ

Crosswork Cloud は REST API 要求を受信すると、次の手順を実行します。

1. 要求されたパラメータを抽出します。
2. API キー ID を使用して、API キーと関連するメタデータを取得します。
3. 署名を再計算します。
4. 計算された署名を要求された署名と比較します。
5. 計算された署名と要求された署名が一致する場合、Crosswork Cloudは要求を転送します。署名が一致しない場合、Crosswork Cloud は要求を拒否します。

Crosswork Cloud Network Insights クライアントスクリプト

このセクションには、Crosswork Cloud Network Insights クライアントスクリプトの使用方法の例と情報が含まれています。

クライアントスクリプトのオプション

クライアントスクリプトの実行時には、次のオプションを使用できます。

```
(ramius) ~> ./crosswork.py -h
usage: crosswork.py [-h] [--uri URI] --key KEY --keyid KEYID
                    [--payload PAYLOAD] [--method {GET,POST}] [--host HOST]
                    [--port PORT]

Exercise the REST API.

optional arguments:
  -h, --help            show this help message and exit
  --uri URI             The URI to run
  --key KEY             A Cisco Crosswork Network Insights API Key
  --keyid KEYID         A Cisco Crosswork Network Insights API Key ID
  --payload PAYLOAD     The name of a file containing JSON data for POST API
                        requests. Note: This option is available only for POST
                        commands.
  --method {GET,POST}   The HTTP method for the request
  --host HOST           The Cisco Crosswork Network Insights URL
```

```
  --port PORT          The Cisco Crosswork Network Insights port number
(ramius) ~>
```

Crosswork Cloud Network Insights クライアントスクリプト例

次のクライアントスクリプトの例はPythonで記述されており、Crosswork Cloud Network Insightsの REST API コールを作成、署名、および実行する方法を示しています。

```
#!/usr/bin/env python3

#
# Copyright 2019 Cisco Systems Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

import argparse
import binascii
import datetime
import hashlib
import hmac
import json
from typing import Dict, Any

import requests
import rfc3339
import sys
import urllib

from string import Template
from urllib.parse import urlparse


class Signature(object):
    # The order and white space usage is very important. Any change
    # can alter the signature and cause the request to fail.
    SIGNATURE_TEMPLATE = Template("""\
$param_method
$param_uri
$param_query_parameters
$param_key_id
$param_timestamp
$param_signature_version
$param_content_sha256
$param_content_type
$param_content_length""")

    def __init__(self, exrest):
        self.exrest = exrest

    def sign(self):
        exrest = self.exrest
```

```
        string_to_sign = self.SIGNATURE_TEMPLATE.substitute({
            "param_method": exrest.method.upper(),
            "param_uri": exrest.url_encoded_uri,
            "param_query_parameters": exrest.url_encoded_query_parameters,
            "param_key_id": exrest.key_id,
            "param_timestamp": exrest.timestamp,
            "param_signature_version": exrest.signature_version,
            "param_content_sha256": exrest.content_sha256,
            "param_content_type": exrest.content_type,
            "param_content_length": exrest.content_length
        })

        # Decode the key and create the signature.
        secret_key_data = binascii.unhexlify(exrest.key)
        hasher = hmac.new(secret_key_data, msg=string_to_sign.encode('utf-8'),
digestmod=hashlib.sha256)
        signature = binascii.hexlify(hasher.digest())
        return signature.decode('utf-8')


class ExRest(object):
    SIGNATURE_VERSION = "1.0"
    CONTENT_TYPE = "application/json"

    HEADER_CONTENT_TYPE = "Content-Type"
    HEADER_CONTENT_LENGTH = "Content-Length"
    HEADER_SIGNATURE_VERSION = "X-Cisco-Crosswork-Cloud-Signature-Version"
    HEADER_TIMESTAMP = "Timestamp"
    HEADER_AUTHORIZATION = "Authorization"

    def __init__(self):
        # Input arguments to the script.
        self.uri = None
        self.payload = None
        self.method = None
        self.host = None
        self.port = None
        self.key = None
        self.key_id = None

        # Values used to calculate the signature.
        self.url_encoded_uri = None
        self.url_encoded_query_parameters = None
        self.timestamp = None
        self.content_sha256 = None
        self.content_length = 0
        self.content_type = self.CONTENT_TYPE
        self.signature_version = self.SIGNATURE_VERSION

    def run(self):
        # Calculate the full URI to be run.
        uri = self.uri[1:] if self.uri.startswith("/") else self.uri
        self.uri = f"https://{self.host}:{self.port}/{uri}"

        # The url encoded uri is used when calculating the request signature.
        parsed_uri = urlparse(self.uri)
        self.url_encoded_uri = urllib.parse.quote(parsed_uri.path, safe="")
        self.url_encoded_query_parameters = urllib.parse.quote(parsed_uri.query)

        # Calculate the rfc3339 timestamp for the request.
        now = datetime.datetime.now()
        self.timestamp = rfc3339.rfc3339(now)
```

```
        # Calculate the SHA256 of the body of the request, even if the body is empty.
        self.content_sha256, self.content_length, payload_contents =
self.calculate_content_sha256(self.payload)

        # Calculate a signature for the request.
        signer = Signature(self)
        request_signature_b64 = signer.sign()

        # Create the request object and set the required http headers.
        headers = dict()

        headers[self.HEADER_AUTHORIZATION] = "hmac {}:{}".format(self.key_id,
request_signature_b64)
        headers[self.HEADER_TIMESTAMP] = self.timestamp
        headers[self.HEADER_CONTENT_TYPE] = self.content_type
        headers[self.HEADER_SIGNATURE_VERSION] = self.SIGNATURE_VERSION

        session = requests.Session()

        response = session.request(self.method, self.uri, data=payload_contents,
headers=headers)

        parsed_response: Dict[str, Any] = dict()
        if len(response.content) > 0:
            content = response.content.decode('utf-8')
            try:
                parsed_response = json.loads(content)
            except ValueError:
                parsed_response = dict()
                parsed_response["Message"] = content.strip()

        if response.status_code != 200:
            parsed_response["HttpStatus"] = response.status_code

        print(json.dumps(parsed_response, indent=2))

    def calculate_content_sha256(self, payload):
        if payload:
            try:
                with open(payload) as fd:
                    payload_contents = fd.read()
            except Exception as error:
                raise Exception(f'Cannot read payload file {payload}: {error}')
        else:
            payload_contents = ""

        hasher = hashlib.sha256()
        hasher.update(payload_contents.encode('utf-8'))

        content_sha256 = binascii.hexlify(hasher.digest())

        return content_sha256.decode('utf-8'), len(payload_contents), payload_contents


def main():
    parser = argparse.ArgumentParser(description="Exercise the REST API.")

    parser.add_argument("--uri", default="/api/beta/truefalse/1/200",
                        help="The URI to run")

    parser.add_argument("--key", required=True,
                        help="A Cisco Crosswork Network Insights API Key")

    parser.add_argument("--keyid", required=True,
```

```
                        help="A Cisco Crosswork Network Insights API Key ID")

    parser.add_argument("--payload",
                        help="The name of a file containing JSON data for POST API
requests")

    parser.add_argument("--method", choices=["GET", "POST"], default="GET",
                        help="The HTTP method for the request")

    parser.add_argument("--host", default="crosswork.cisco.com",
                        help="The Cisco Crosswork Network Insights URL")

    parser.add_argument("--port", type=int, default=443,
                        help="The Cisco Crosswork Network Insights port number")

    # Parse the arguments
    args = parser.parse_args()

    exrest = ExRest()

    exrest.uri = args.uri
    exrest.payload = args.payload
    exrest.method = args.method
    exrest.host = args.host
    exrest.port = args.port
    exrest.key = args.key
    exrest.key_id = args.keyid

    exrest.run()


if __name__ == "__main__":
    sys.exit(main())
```

クライアントスクリプトの使用方法

この例では、次のタスクについて説明します：

- クライアントスクリプトから簡単なコールを行います。
- ペイロードオプションと設定ファイルを使用して、POST コマンドでプレフィックスを追加します。

始める前に

スクリプトを実行する前に、API キーを要求します（API の使用開始 （2 ページ） を参照）。API の詳細については、Crosswork Cloud UI から?をクリックし、**API** リンクをクリックしてください。

ステップ 1 次のスクリプトを実行します。

```
crosswork.py --uri '/api/beta/sourcedata?prefix=64.54.195.0%2F24&max=5' --key '<yourKeyHere>' --keyid
 '<yourKeyIdHere>'
```

結果の例：

```
{
  "data": [
    {
      "prefix": "64.54.195.0/24",
      "action": "ADD",
      "peerRemoteAsn": 22024,
      "timestamp": "2021-10-20T18:32:03Z",
      "origin": "IGP",
      "originAs": 5653,
      "asPath": [
        {
          "asn": [
            22024
          ]
        },
        {
          "asn": [
            6461
          ]
        },
        {
          "asn": [
            5653
          ]
        }
      ],
      "unicastPrefixType": "ADJ_RIB_IN",
      "nextHop": "4.4.94.118/32",
      "peerRemoteId": "549",
      "roaGenTime": "2021-06-29T05:25:53.844840001Z"
    },
    {
      "prefix": "64.54.195.0/24",
      "action": "ADD",
      "peerRemoteAsn": 202365,
      "timestamp": "2022-01-21T10:25:58Z",
      "origin": "IGP",
      "originAs": 5653,
      "med": {},
      "communities": [
        3792306480,
        3792306677,
        57866,
        41441,
        41441
      ],
      "asPath": [
        {
          "asn": [
            202365
          ]
        },
        {
          "asn": [
            57866
          ]
        },
        {
          "asn": [
            6461
          ]
        },
        {
          "asn": [
```

```
            5653
          ]
        }
      ],
      "unicastPrefixType": "ADJ_RIB_IN",
      "nextHop": "5.255.90.109/32",
      "peerRemoteId": "248",
      "roaGenTime": "2021-10-05T10:07:45.504885118Z"
    },
(truncated)
```

ステップ 2 POST コマンドと設定ファイルでプレフィックスを追加します：

```
crosswork.py --uri '/api/beta/provision' --key '<yourKeyHere>' --keyid '<yourKeyIdHere>' --payload
 "config.json" --method "POST"
```

config.json ファイルのコンテンツ例：

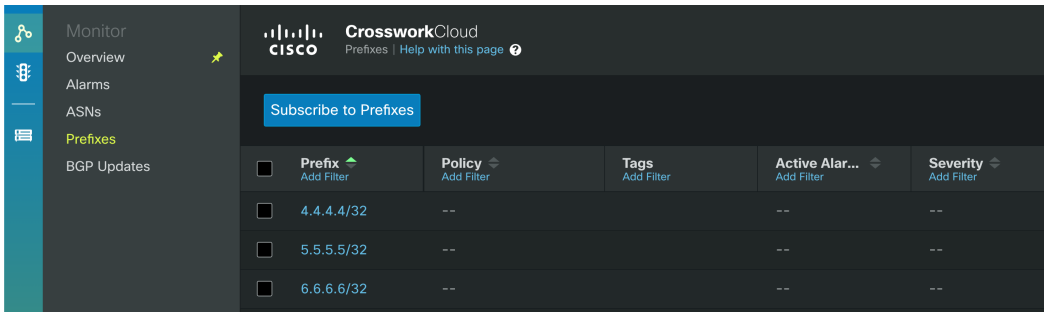
```
{
  "operations": [
    {
      "setPrefixRequest": {
        "prefix": "4.4.4.4/32"
      },
      "o_creat": true,
      "o_excl": true
    },
    {
      "setPrefixRequest": {
        "prefix": "5.5.5.5/32"
      },
      "o_creat": true,
      "o_excl": true
    },
    {
      "setPrefixRequest": {
        "prefix": "6.6.6.6/32"
      },
      "o_creat": true,
      "o_excl": true
    },
    {
      "setPrefixRequest": {
        "prefix": "2001:30:102::/48"
      },
      "o_creat": true,
      "o_excl": true
    }
  ]
}
```

結果の例：

```
{
  "results": [
    {
      "setPrefixResponse": {
        "prefix": "4.4.4.4/32"
      }
    },
    {
      "setPrefixResponse": {
        "prefix": "5.5.5.5/32"
      }
    },
```

```
    {
      "setPrefixResponse": {
        "prefix": "6.6.6.6/32"
      }
    },
    {
      "setPrefixResponse": {
        "prefix": "2001:30:102::/48"
      }
    }
  ]
}
```

UI 結果の例：



Crosswork トラフィック分析クライアントスクリプトの例

次のスクリプトの例は、Python で記述されています。Crosswork Traffic Analysis API を実行するには、**python/get_traffic_example.py** および **python/cctraffic/cctraffic.py** が必要です。`get_traffic_example.py` を実行する前に、次のことを行う必要があります。

1. Python の依存関係をインストールします：`pip3 install -r requirements.txt`
2. API ベアラートークン（`export TOKEN=<token string>`）を設定します
3. get_traffic_example.py ファイルを編集します。次の値を正しい値に置き換えます：`api_version`、`device_name`、`start` および `end`。

`get_traffic_example.py` ファイルを編集した後、スクリプトを実行します：`python3 get_traffic_example.py`

スクリプトの例：get_traffic_example.py

```
# get_traffic_example.py

import os
import sys
from cctraffic import CCTrafficRestClient

host = "https://crosswork.cisco.com"
```

```
api_version = "beta"
device_name = "flow-automation-1"

# start and end may be supplied as:
#   - ISO 8601 datetime string
#   - unix timestamp in seconds since 1970
#   - now
#   - "<number> <unit> ago" where unit can be: "seconds", "minutes", "hours", "days".
start = "7 days ago"
end = "now"

if "TOKEN" in os.environ:
    token = os.environ["TOKEN"]
else:
    print("Bearer token not found. Set bearer token with: export TOKEN=<token string>")
    sys.exit(-1)

client = CCTrafficRestClient(host, token, version=api_version, debug=False)

print(f"GetDevice for {device_name}")
device_info = client.GetDevice(device_name)
device_id = device_info["deviceId"]
print(f"Found device ID for {device_name}: {device_id}")

print(f"Traffic by interface for {device_name}")
traffic_for_my_device = client.GetInterfaceCounterTrafficTotals(start, end, device_id)
interface_name = traffic_for_my_device[0]["interfaces"][0]["interfaceName"]

print(f"Traffic by ASN for {device_name}/{interface_name}")
asn_traffic_for_my_device_interface = client.GetNetFlowTrafficTotalsByDevice(start, end,
 device_id, interface=interface_name, asn_breakdown=True)

print(f"Traffic by Prefix for {device_name}/{interface_name}")
prefix_traffic_for_my_device_interface = client.GetNetFlowTrafficTotalsByDevice(start,
end, device_id, interface=interface_name, prefix_breakdown=True)

asn = asn_traffic_for_my_device_interface[0]["interfaces"][0]["asns"][0]["asn"]
device_prefix =
prefix_traffic_for_my_device_interface[0]["interfaces"][0]["prefixes"][0]["prefix"]

print(f"Traffic by Prefix for {device_name}/{interface_name} ASN {asn}")
prefix_traffic_for_my_device_interface_asn = client.GetNetFlowTrafficTotalsByDevice(
    start, end, device_id, interface=interface_name, asn=asn, asn_breakdown=True)

print(f"Traffic by Prefix")
prefix_traffic = client.GetNetFlowTrafficTotalsByPrefix(start, end)
prefix = prefix_traffic[0]["prefix"]

print(f"Traffic by Device for {prefix}")
device_traffic_for_prefix = client.GetNetFlowTrafficTotalsByPrefix(start, end, prefix)

print(f"Time series for {device_name}")
time_series_for_device = client.GetInterfaceCounterTrafficTimeSeries(start, end, device_id)

print(f"Time series for {device_name}/{interface_name}")
time_series_for_interface = client.GetInterfaceCounterTrafficTimeSeries(start, end,
device_id, interface=interface_name)

print(f"Time series for {device_name}/{interface_name} {device_prefix}")
time_series_for_prefix = client.GetNetFlowTrafficTimeSeriesByDevice(start, end, device_id,
 interface=interface_name, prefix=device_prefix)

print(f"Time series for {device_name}/{interface_name} {asn}")
```

```
time_series_for_asn = client.GetNetFlowTrafficTimeSeriesByDevice(start, end, device_id,
 interface=interface_name, asn=asn)
```

スクリプトの例：cctraffic.py

```
# cctraffic.py
# Contains a very simple REST client to demonstrate how to call the Crosswork Cloud
Traffic APIs
# Copyright (c) 2021 Cisco Systems, Inc. and others.  All rights reserved.

import requests
from .util import UrlEncode

import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)


class CCTrafficRestClient:

    def __init__(self, host: str, token, version: str = "v1", debug: bool = False):
        self.version = version
        self.host = host
        self.debug = debug
        self.headers = {"content-type": "application/json", "Authorization": f"Bearer
{token}"}

    def DoApiCall(self, url):
        if self.debug == True:
            print(url)
        response = requests.get(url, headers=self.headers, verify=False)
        if self.debug == True:
            print(response.status_code)
            print(response.content)
        return response

    def GetDevice(self, device_name: str):
        url = f"{self.host}/api/{self.version}/devices?name={device_name}"
        response = self.DoApiCall(url)
        if response.status_code != 200:
            return ""
        return response.json()["devices"][0]["deviceInfo"]

    def GetInterfaceCounterTrafficTotals(self, start: str, end: str, device_id: str =
""):
        start = UrlEncode(start)
        end = UrlEncode(end)

        if device_id == "":
            url = f"{self.host}/api/{self.version}/devices/statistics/totals"
        else:
            url =
f"{self.host}/api/{self.version}/devices/{device_id}/interfaces/statistics/totals"

        url += f"?format=totals&timeStart={start}&timeEnd={end}"
        response = self.DoApiCall(url)
        if response.status_code != 200:
            return ""
        return response.json()["devices"]

    def GetNetFlowTrafficTotalsByDevice(self, start: str, end: str, device_id: str,
interface: str,
                                        asn: int = 0, prefix: str = "", asn_breakdown:
bool = False, prefix_breakdown: bool = False):
        interface = UrlEncode(UrlEncode(interface))
        prefix = UrlEncode(UrlEncode(prefix))
```

```
        start = UrlEncode(start)
        end = UrlEncode(end)

        if asn != 0:
            url =
f"{self.host}/api/{self.version}/traffic/devices/{device_id}/interfaces/{interface}/asns/{asn}/prefixes"

        elif asn_breakdown:
            url =
f"{self.host}/api/{self.version}/traffic/devices/{device_id}/interfaces/{interface}/asns"

        elif prefix_breakdown:
            url =
f"{self.host}/api/{self.version}/traffic/devices/{device_id}/interfaces/{interface}/prefixes"


        url += f"?format=totals&timeStart={start}&timeEnd={end}"
        response = self.DoApiCall(url)
        if response.status_code != 200:
            return ""
        return response.json()["devices"]

    def GetNetFlowTrafficTotalsByPrefix(self, start: str, end: str, prefix: str = "",
device_id: str = ""):
        prefix = UrlEncode(UrlEncode(prefix))
        start = UrlEncode(start)
        end = UrlEncode(end)

        if prefix == "":
            url = f"{self.host}/api/{self.version}/traffic/prefixes"
        elif device_id == "":
            url = f"{self.host}/api/{self.version}/traffic/prefixes/{prefix}/devices"
        else:
            url =
f"{self.host}/api/{self.version}/traffic/prefixes/{prefix}/devices/{device_id}/interfaces"


        url += f"?format=totals&timeStart={start}&timeEnd={end}"
        response = requests.get(url, headers=self.headers, verify=False)
        if response.status_code != 200:
            return ""
        return response.json()["prefixes"]

   def GetInterfaceCounterTrafficTimeSeries(self, start: str, end: str, device_id: str,
 interface: str = ""):
        interface = UrlEncode(UrlEncode(interface))
        start = UrlEncode(start)
        end = UrlEncode(end)

        if interface == "":
          url = f"{self.host}/api/{self.version}/devices/{device_id}/statistics/totals"

        else:
            url =
f"{self.host}/api/{self.version}/devices/{device_id}/interfaces/{interface}/statistics/totals"


        url += f"?format=timeseries&timeStart={start}&timeEnd={end}"
        response = requests.get(url, headers=self.headers, verify=False)
        if response.status_code != 200:
            return ""
        return response.json()["devices"]

    def GetNetFlowTrafficTimeSeriesByDevice(self, start: str, end: str, device_id: str,
```

```
 interface: str, asn: int = 0, prefix: str = ""):
        interface = UrlEncode(UrlEncode(interface))
        prefix = UrlEncode(UrlEncode(prefix))
        start = UrlEncode(start)
        end = UrlEncode(end)

        if asn == 0 and prefix != "":
            url =
f"{self.host}/api/{self.version}/traffic/devices/{device_id}/interfaces/{interface}/prefixes/{prefix}"

        elif asn != 0 and prefix == "":
            url =
f"{self.host}/api/{self.version}/traffic/devices/{device_id}/interfaces/{interface}/asns/{asn}"

        else:
            url =
f"{self.host}/api/{self.version}/traffic/devices/{device_id}/interfaces/{interface}/asns/{asn}/prefixes/{prefix}"


        url += f"?format=timeseries&timeStart={start}&timeEnd={end}"
        response = self.DoApiCall(url)
        if response.status_code != 200:
            return ""
        return response.json()["devices"]
```

翻訳について
このドキュメントは、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。