Cisco Secure Workload OpenAPI

OpenAPI 为 Cisco Secure Workload 功能提供 REST API。

OpenAPI 身份验证

OpenAPI 使用基于摘要的身份验证方案。工作流程如下：

1. 登录 Cisco Secure Workload UI 控制面板。
2. 通过所需功能生成 API 密钥和 API 密钥。
3. 使用 Cisco Secure Workload API SDK 以 JSON 格式发送 REST 请求。
4. 要使用 Python SDK，请使用 `pip install tetpyclient` 来安装 SDK。
5. 安装 Python SDK 后，下面是一些实例化 RestClient 的模板代码：

```
from tetpyclient import RestClient

API_ENDPOINT="https://<UI_VIP_OR_DNS_FOR_TETRATION_DASHBOARD>"

# ``verify`` is an optional param to disable SSL server authentication.
# By default, cluster dashboard IP uses self signed cert after
# deployment. Hence, ``verify=False`` might be used to disable server
# authentication in SSL for API clients. If users upload their own
# certificate to cluster (from ``Platform > SSL Certificate``)
# which is signed by their enterprise CA, then server side authentication
# should be enabled; in such scenarios, in the code below, verify=False
# should be replaced with verify="path-to-CA-file"
# credentials.json looks like:
# {
#   "api_key": "<hex string>",
#   "api_secret": "<hex string>"
# }

restclient = RestClient(API_ENDPOINT,
                credentials_file='<path_to_credentials_file>/credentials.json',
                verify=False)
# followed by API calls, for example API to retrieve list of agents.
# API can be passed /openapi/v1/sensors or just /sensors.
```

```
resp = restclient.get('/sensors')
```

生成 API 密钥和秘密

Procedure

步骤 1 在 Cisco Secure Workload UI 的右上角，点击已登录的帐户，然后选择 **API 密钥 (API Keys)**。

步骤 2 点击**创建 API 密钥 (Create API Key)**。

步骤 3 （可选）输入 API 密钥的说明。

步骤 4 为密钥和秘密选择所需的功能。

选择用于使用 API 密钥 + 秘密对的有限功能集。

Note API 功能的可用性会因用户角色而异。

***Table 1: API* 功能**

功能	说明
sensor_management	配置和监控软件代理的状态
software_download	下载用于代理或虚拟设备的软件包
flow_inventory_query	查询 Cisco Secure Workload 集群中的流和资产项目
user_role_scope_management	读取、添加、修改或删除用户、角色和范围
user_data_upload	允许用户上传用于注释流和资产项目的数据，或者上传良好或不良文件散列
app_policy_management	管理工作空间（应用）和执行策略
external_integration	允许与外部系统（例如 vCenter 和 Kubernetes）集成

***Table 2:* 面向站点管理员的 *API* 功能**

功能	说明
appliance_management	管理 Cisco Secure Workload 设备
appliance_monitoring	监控 Cisco Secure Workload 设备设置和配置（只读）

步骤 5 点击**创建 (Create)**。

生成 API 密钥和秘密，必须将其复制到文件中，然后保存在安全位置。或者，您可以下载包含密钥和秘密的 JSON 文件。

Note 如果启用了“使用 LDAP 进行外部身份验证”(External Auth with LDAP) 和“LDAP 授权”(LDAP Authorization)，则使用 API 密钥对 OpenAPI 的访问会停止，因为在用户会话终止后，将从 LDAP MemberOf 组派生的 Cisco Secure Workload 角色重新评估。因此，为确保 OpenAPI 访问不中断，建议任何拥有 API 密钥的用户在“编辑用户详细信息”流程中为该用户启用**使用本地身份验证 (Use Local Authentication)** 选项。

工作空间和安全策略

以下页面介绍用于管理分段的 OpenAPI 终端。

工作空间

工作空间（以前称为 “应用工作空间 ”或 “应用”）是为特定范围内的工作负载定义、分析和执行策略的容器。有关其工作原理的详细信息，请参阅 工作空间 文档。这组 API 需要使用与 API 密钥关联的 `app_policy_management` 功能。

工作空间对象

工作空间（“应用”）JSON 对象以单个对象或对象数组的形式返回，具体取决于 API 终端。对象的属性如下所述：

属性	类型	说明
id	字符串	工作空间的唯一标识符。
name	字符串	用户指定的工作空间名称。
description	字符串	用户指定的工作空间说明。
app_scope_id	字符串	与工作空间关联的范围的 ID。
author	字符串	创建工作空间的用户的名字和姓氏。
primary	布尔值	指明工作空间是否是其范围的主工作空间。
alternate_query_mode	布尔值	指明工作空间是否使用“动态模式”。在动态模式下，自动策略发现运行会为每个集群创建一个或多个候选查询。默认值为 True。
created_at	整数	创建工作空间时的 Unix 时间戳。

属性	类型	说明
latest_adm_version	整数	工作空间的最新 adm (v*) 版本。
analysis_enabled	布尔值	指明是否已对工作空间启用分析。
analyzed_version	整数	工作空间的已分析 p* 版本。
enforcement_enabled	布尔值	指明是否已为工作空间启用执行。
enforced_version	整数	工作空间的强制 p* 版本。

列出应用

此终端将返回一组工作空间（“应用”）。

```
GET /openapi/v1/applications
```

Table 3: 参数

名称	类型	说明
app_scope_id	字符串	匹配与特定应用范围关联的工作空间。
exact_name	字符串	将工作空间与提供的值完全匹配。

响应对象：返回工作空间对象的数组。

示例 python 代码

```
restclient.get('/applications')
```

检索单个工作空间

此终端会将请求的工作空间（“应用”）作为单个 JSON 对象返回。

```
GET /openapi/v1/applications/{application_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。

响应对象：返回指定 ID 的工作空间对象。

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
restclient.get('/applications/%s' % application_id)
```

创建工作空间

此终端创建一个工作空间（“应用”）。可以通过发布包含集群和策略定义的 JSON 主体来定义策略。

Note 如果存在用于同一范围的主工作空间并提供了新的策略，则这些策略将作为新版本添加到现有工作空间。

```
POST /openapi/v1/applications
```

参数：JSON 查询正文包含以下键

名称	类型	说明
app_scope_id	字符串	要分配给工作空间的范围 ID。
name	字符串	（可选）工作空间的名称。
description	字符串	（可选）工作空间的说明。
alternate_query_mode	布尔值	（可选）指明工作空间是否使用“动态模式”。在动态模式下，自动策略发现运行会为每个集群创建一个或多个候选查询。默认值为 True。
strict_validation	布尔值	（可选）如果上传的数据中存在未知键或属性，将返回错误。可用于捕获存在拼写错误的键。默认值为 false。
primary	字符串	（可选）如果此工作空间应作为关联范围的主工作空间，则设置为“true”。**默认值为 true**

还可包含其他可选参数，用于描述在工作空间内创建的策略。

Note 方案对应于在导出期间从 UI 和**详细信息**终端返回的方案。

名称	类型	说明
clusters	集群数组	用于定义策略的节点组。
inventory_filters	资产过滤器数组	数据中心资产过滤器。

名称	类型	说明
absolute_policies	策略数组	要创建的具有绝对等级的有序策略。
default_policies	策略数组	要创建的具有默认等级的有序策略。
catch_all_action	字符串	“ALLOW”或“DENY”

集群对象属性：

名称	类型	说明
id	字符串	要与策略一起使用的唯一标识符。
name	字符串	显示的集群名称。
description	字符串	集群的说明。
nodes	节点数组	属于集群一部分的节点或 终端。
consistent_uuid	字符串	对于给定的工作空间必须是唯一的。自动策略发现运行后，下一版本中的类似/相同集群将保持 consistent_uuid。

节点对象属性：

名称	类型	说明
ip	字符串	节点的 IP 或子网。例如 10.0.0.0/8 或 1.2.3.4
name	字符串	节点的显示名称。

资产过滤器对象属性：

名称	类型	说明
id	字符串	要与策略一起使用的唯一标识符。
name	字符串	显示的集群名称。
query	对象	资产过滤器查询的 JSON 对象表示。

策略对象属性：

名称	类型	说明
consumer_filter_id	字符串	集群、用户资产过滤器或应用范围的ID。
provider_filter_id	字符串	集群、用户资产过滤器或应用范围的ID。
action	字符串	“ALLOW”或“DENY”
l4_params	array of l4params	允许的端口和协议列表。

L4Params 对象属性：

名称	类型	说明
proto	整数	协议整数值（NULL 表示所有协议）。
port	数组	端口所包含的范围。例如，[80, 80] 或 [5000, 6000]。
approved	布尔值	（可选）指明策略是否已获批准。默认值为 False。

响应对象：返回新创建的工作空间对象。

示例 python 代码

```
  name = 'test'
  scope_id = '5ce480cc497d4f1b4b9a9e8d'
  filter_id = '5ce480cd497d4f1b4b9a9ea4'
  application = {
      'app_scope_id': scope_id,
      'name': name,
      'absolute_policies': [
          {
       # consumer/provider filter IDs can be ID of a cluster identified during automatic
policy discovery (formerly known as ADM),
       # user inventory filter or app scope.
              'provider_filter_id': filter_id,
              'consumer_filter_id': filter_id,
              'action': 'ALLOW',
       # ALLOW policy for TCP on port 80.
              'l4_params': [
                  {
                      'proto': 6, # TCP
                      'port': [80, 80],  # port range
                  }
              ],
          }
      ],
      'catch_all_action': 'ALLOW'
  }
  restclient.post('/applications', json_body=json.dumps(application))
```

导入新版本

导入策略并为工作空间（“应用”）创建新的 v* 版本。

```
POST /openapi/v1/applications/{application_id}/import
```

参数与创建工作空间终端相同。

响应对象：返回工作空间对象。

验证策略集

验证一组策略，而不创建新版本。

```
POST /openapi/v1/applications/validate_policies
```

app_scope_id 为必填。其余参数与创建工作空间终端相同。

响应对象：

属性	类型	说明
valid	布尔值	指明策略是否有效
errors	数组	如果无效，则提供有关错误的详细信息

删除工作空间

删除工作空间（“应用”）

```
DELETE /openapi/v1/applications/{application_id}
```

必须先在工作空间上禁用执行，然后才能将其删除。

如果工作空间或其集群被其他应用使用（通过提供的服务关系），则此终端将返回 `422 个无法处理的实体`。返回的 Error 对象将包含一个 `details` 属性，该属性具有依赖对象的计数以及每种类型的前 10 个对象的 ID。此信息可用于查找和删除阻止依赖关系。

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。

响应对象：无

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
restclient.delete('/applications/%s' % application_id)
```

更新工作空间

此终端更新现有工作空间（“应用”）。

```
PUT /openapi/v1/applications/{application_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。

JSON 查询正文包含以下键

名称	类型	说明
name	字符串	（可选）工作空间的更新名称。
descrip	字符串	（可选）工作空间的更新说明。
primary	字符串	（可选）设置为“true”可将工作空间设置为主工作空间。设置为“false”可将工作空间设为辅助工作空间。

响应对象：指定 ID 的已更新工作空间对象。

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
req_payload = {
      'name': 'Updated Name',
      'description': 'Updated Description',
      'primary': 'true'
}
resp = restclient.put('/applications/%s' % application_id,
                      json_body=json.dumps(req_payload))
```

检索工作空间详细信息

此终端会返回工作空间的完整导出 JSON 文件。这将包括策略和集群定义。

```
GET /openapi/v1/applications/{application_id}/details
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。
version	字符串	（可选）“v10”形式的版本或“p10”，默认为“latest”。

响应对象：返回给定工作空间版本的集群和策略。

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
# For v* version v10 and for p* version p10
version = 'v10'
resp = restclient.get('/applications/%s/details?version=%s' % (application_id, version))
```

列出工作空间版本

此终端将返回给定工作空间的所有版本的列表。

```
GET /openapi/v1/applications/{application_id}/versions
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。
created_before	整数	（可选）对于分页，请设置为上一个响应中最后一个版本的“created_at”。
limit	整数	（可选）要返回的最大结果数，默认值为 50。

响应对象：具有以下属性的对象数组：

属性	类型	说明
version	字符串	“v10”或“p10”形式的版本。
created_at	整数	创建工作空间时的Unix时间戳。
description	字符串	用户提供的说明。
name	字符串	显示的名称。

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
created_before = 1612325705
limit = 10
resp = restclient.get('/applications/%s/versions?created_before=%s&limit=%s' %
                      (application_id, created_before, limit))
```

删除工作空间版本

此终端将删除给定的版本，包括集群和策略。无法删除已执行或已分析的版本。如果成员被另一个工作空间通过外部策略引用，响应将返回错误信息，并列出引用列表。

```
DELETE /openapi/v1/applications/{application_id}/versions/{version}
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。
version	字符串	“v10”或“p10”形式的版本。

响应对象：无

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
version = 'v10'
resp = restclient.delete('/applications/%s/versions/%s' %
                         (application_id, version))
```

比较工作空间版本

此终端会计算所提供的工作空间版本之间的差值。它将返回已添加、已删除以及可选的未更改策略。如果集群在两个版本中都存在，由匹配的 consistent_uuid 定义，且查询已更改，则集群更改会被包括在内。

```
GET /openapi/v1/applications/{application_id}/version_diff
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。
base_version	字符串	完整版本，例如“v10”或“p10”。
draft_version	字符串	完整版本，例如“v10”或“p10”。
include_unchanged	布尔值	默认值为 false。返回响应中未更改的策略。

响应对象：返回具有以下属性的对象：

属性	类型	说明
clusters	数组	在不同版本之间更改的集群。
policies	数组	在不同版本之间更改的策略。

分析最新策略

在工作空间中启用对最新策略集的分析。

```
POST /openapi/v1/applications/{application_id}/enable_analysis
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。

参数：可选的 JSON 查询正文包含以下键

名称	类型	说明
action_note	字符串	（可选）发布策略操作的原因。
name	字符串	（可选）已发布策略版本的名称。
description	字符串	（可选）已发布策略版本的说明。

响应对象：返回具有以下属性的对象：

属性	类型	说明
data_set	对象	数据集的 JSON 对象表示。
analyzed_policy_version	整数	工作空间的已分析 p* 版本。

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
req_payload = {
  'action_note': 'Policy analysis',
  'name': 'Test run 1',
  'description': 'New workloads added.'
}
resp = restclient.post('/applications/%s/enable_analysis' % application_id,
                       json_body=json.dumps(req_payload))
```

在单个工作空间上禁用策略分析

在工作空间上禁用策略分析。

```
POST /openapi/v1/applications/{application_id}/disable_analysis
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。

响应对象：返回具有以下属性的对象：

属性	类型	说明

data_set	对象	数据集的 JSON 对象表示。
analyzed_policy_version	整数	上次分析的 p* 版本工作空间。

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
resp = restclient.post('/applications/%s/disable_analysis' % application_id)
```

执行单个工作空间

在工作空间中的最新策略集上启用执行。

```
POST /openapi/v1/applications/{application_id}/enable_enforce
```

Warning 将在相关主机上插入新的主机防火墙规则，并删除任何现有规则。

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。
version	字符串	（可选）要执行的策略版本。

如果未提供版本，则将执行工作空间的最新策略。版本的首选格式为“p*”，如果仅提供整数，则将执行相应的“p*”版本。

响应对象：返回具有以下属性的对象：

名称	类型	说明
epoch	字符串	最新执行配置文件的唯一标识符。

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
req_payload = {
  'version': 'p10'
}
resp = restclient.post('/applications/%s/enable_enforce' % application_id,
                       json_body=json.dumps(req_payload))
```

对单个工作空间禁用执行

在工作空间上禁用执行。

```
POST /openapi/v1/applications/{application_id}/disable_enforce
```

Warning 将在相关主机上插入新的主机防火墙规则，并删除任何现有规则。

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。

响应对象：返回具有以下属性的对象：

名称	类型	说明
epoch	字符串	最新执行配置文件的唯一标识符。

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
resp = restclient.post('/applications/%s/disable_enforce' %
                       application_id)
```

启动自动策略发现

自动发现工作空间的策略。（以前称为“提交 ADM 运行”）。

```
POST /openapi/v1/applications/{application_id}/submit_run
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。

参数：JSON 查询正文包含以下键

名称	类型	说明
start_time	字符串	自动发现策略运行的输入时间间隔的开始时间。
end_time	字符串	自动策略发现运行的输入时间间隔的结束时间。
clustering_granularity	字符串	（可选） 集群粒度 允许用户控制自动策略发现生成的集群的大小。预期值：VERY_FINE、FINE、MEDIUM、COARSE 或 VERY_COARSE

名称	类型	说明
port_generalization	字符串	（可选） 端口泛化 控制执行端口泛化时所需的统计显著性级别。预期值：DISABLED、CONSERVATIVE、Moderate、AGGRESSIVE 或 VERY_AGGRESSIVE
policy_compression	字符串	（可选）启用策略压缩时，在工作空间内生成的集群中足够频繁的策略（即它们使用相同的提供者端口）可能会被“分解”到父项，即替换为适用于的一个或多个策略整个父范围。预期值：DISABLED、CONSERVATIVE、Moderate、AGGRESSIVE 或 VERY_AGGRESSIVE
auto_accept_policy_connectors	布尔值	（可选）自动接受策略连接器将自动接受在自动策略发现期间创建的任何传出策略请求。
enable_exclusion_filter	布尔值	（可选）启用排除过滤器选项可灵活地忽略与任何用户定义的排除过滤器（如有）匹配的所有对话。有关详细信息，请参阅排除过滤器。
enable_default_exclusion_filter	布尔值	（可选）启用默认排除过滤器选项可灵活地忽略与任何默认排除过滤器（如有）匹配的所有对话。有关详细信息，请参阅默认排除过滤器了解详细信息。
enable_service_discovery	布尔值	（可选）当设置在代理上启用服务发现时，系统会报告有关代理节点上存在的服务的临时端口范围信息。然后，根据报告的端口范围信息生成策略。
carry_over_policies	布尔值	（可选）设置沿用已批准的策略时，系统将保留用户通过 UI 或 OpenAPI 标记为已批准的所有策略。

名称	类型	说明
skip_clustering	布尔值	（可选）设置跳过集群时不会生成新集群，而是从任何已批准的现有集群或资产过滤器生成策略，否则会涉及范围内的所有工作负载。
deep_policy_generation	布尔值	（可选）您可以为范围树的分支而不是单个范围生成策略。有关信息，请参阅发现一个范围或范围树分支的策略和子主题。
use_default_config	布尔值	（可选）设置此选项后，自动发现策略将使用默认策略发现配置，而不是之前的运行配置。有关详细信息，请参阅默认策略发现配置。

Note 未指定的可选参数默认值将取自先前的自动策略发现运行配置（如果在工作空间的早期执行过自动策略发现），否则默认值将取自默认策略发现配置。

响应对象：返回具有以下属性的对象：

名称	类型	说明
message	字符串	有关自动策略发现运行成功或失败的消息。

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
req_payload = {
      'start_time': '2020-09-17T10:00:00-0700',
      'end_time': '2020-09-17T11:00:00-0700',
      # Optional Parameters.
      'clustering_granularity': 'FINE',
      'port_generalization': 'AGGRESSIVE',
      'policy_compression': 'AGGRESSIVE',
      'auto_accept_policy_connectors': False,
      'enable_exclusion_filter': True,
      'enable_default_exclusion_filter': True,
      'enable_service_discovery': True,
      'carry_over_policies': True,
      'skip_clustering': False,
      'deep_policy_generation': True,
      'use_default_config': False
}
resp = restclient.post('/applications/%s/submit_run' % application_id,
                       json_body=json.dumps(req_payload))
```

获取策略发现运行的状态

查询工作空间中运行的自动策略发现的状态。

```
GET /openapi/v1/applications/{application_id}/adm_run_status
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。

响应对象：返回具有以下属性的对象：

名称	类型	说明
status	字符串	自动策略发现运行的状态。值：PENDING、COMPLETE 或 FAILED

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
resp = restclient.get('/applications/%s/adm_run_status' % application_id)
```

策略

这组 API 可用于管理添加、编辑或删除策略。创建和更新“全部捕获”操作需要使用 `version` 参数。它们需要使用与 API 密钥关联的 `user_role_scope_management` 功能。

策略对象

策略对象属性如下所述：

属性	类型	说明
id	字符串	策略的唯一标识符。
application_id	字符串	策略所属的工作空间的 ID。
consumer_filter_id	字符串	已定义过滤器的 ID。目前，任何集群、用户定义的过滤器或范围均可用作策略的使用者。
provider_filter_id	字符串	已定义过滤器的 ID。目前，任何集群、用户定义的过滤器或范围均可用作策略的提供者。
version	字符串	指明策略所属的工作空间的版本。

属性	类型	说明
rank	字符串	策略等级，可能的值：DEFAULT、ABSOLUTE 或 CATCHALL。
policy_action	字符串	可能的值可以是 ALLOW 或 DENY。指明对于使用者和提供者之间的给定服务端口/协议，是允许还是丢弃流量。
priority	整数	用于对策略进行排序。
l4_params	array of l4params	允许的端口和协议列表。

L4Params 对象属性：

名称	类型	说明
proto	整数	协议整数值（NULL 表示所有协议）。
port	数组	端口的包含范围。例如 [80, 80] 或 [5000, 6000]。
description	字符串	有关此协议和端口的简短字符串。
approved	布尔值	如果策略已被用户批准。

获取策略

此终端将返回特定工作空间中的策略列表。此 API 可用于具有 `app_policy_management` 功能的 API 密钥。

```
GET /openapi/v1/applications/{application_id}/policies
```

参数：请求 URL 包含以下参数

名称	类型	说明
version	字符串	指明要从中获取策略的工作空间的版本。
consumer_filter_id	字符串	（可选）按使用者过滤器 ID 过滤输出。
provider_filter_id	字符串	（可选）按使用者过滤器 ID 过滤输出。

策略 ID 可能因版本而异。要从已发布版本获取策略列表，则版本号应以“p”作为前缀。例如，要获取已发布版本 3 中的所有策略，可以执行如下请求：

```
GET /openapi/v1/applications/{application_id}/policies?version=p3
```

返回此特定工作空间中所有策略的对象，如下所示

```
    {
      absolute_policies: [ ... ],
      default_policies: [ ... ],
      catch_all_action:
    }
```

示例 Python 代码

```
  application_id = '5f88c996755f023f3bafe163'
  restclient.get('/applications/%s/policies' % application_id, params={'version': '1'})
```

获取默认策略

此终端将返回给定工作空间的默认策略列表。此API可用于具有 `app_policy_management` 功能的API密钥。

```
GET /openapi/v1/applications/{application_id}/default_policies
```

参数：

名称	类型	说明
id	字符串	策略的唯一标识符。
version	字符串	指明要从其获取策略的工作空间的版本。
limit	整数	限制每个请求的策略数。
offset	整数	（可选）从上一个响应中收到的偏移量，应始终与 `limit` 一起使用。
consumer_filter_id	字符串	（可选）按使用者过滤器 ID 过滤输出。
provider_filter_id	字符串	（可选）按提供者过滤器 ID 过滤输出。

返回此工作空间的所提供版本的默认策略列表。响应包含请求的策略数量和偏移，要获取下一组策略，请在后续请求中使用此偏移。响应中没有偏移表示已检索所有策略。

示例 Python 代码

```
  application_id = '5f88c996755f023f3bafe163'
  restclient.get('/applications/%s/default_policies' % application_id, params={'version':
'1', 'limit': 3, 'offset': 3})
```

响应示例

```
{
  "results": [
  PolicyObject4,
  PolicyObject5,
  PolicyObject6
  ],
  "offset": 6
}
```

获取绝对策略

此终端会返回给定工作空间中的绝对策略列表。此 API 可用于具有 `app_policy_management` 功能的 API 密钥。

```
GET /openapi/v1/applications/{application_id}/absolute_policies
```

参数：

名称	类型	说明
version	字符串	指明要从中获取策略的工作空间的版本。
limit	整数	限制每个请求的策略数。
offset	整数	（可选）从上一个响应中收到的偏移量，应始终与 `limit` 一起使用。
consumer_filter_id	字符串	（可选）按使用者过滤器 ID 过滤输出。
provider_filter_id	字符串	（可选）按提供者过滤器 ID 过滤输出。

返回此工作空间所提供版本中的绝对策略列表。响应包含请求的策略数量和偏移，要获取下一组策略，请在后续请求中使用此偏移。响应中没有偏移表示已检索所有策略。

示例 Python 代码

```
 application_id = '5f88c996755f023f3bafe163'
 restclient.get('/applications/%s/absolute_policies' % application_id, params={'version':
'1', 'limit': 3})
```

响应示例

```
{
  "results": [
  PolicyObject1,
  PolicyObject2,
  PolicyObject3
  ],
```

```
    "offset": 3
  }
```

获取“全部捕获”策略

此终端会返回给定工作空间的“全部捕获”策略。此 API 可用于具有 `app_policy_management` 功能的 API 密钥。

```
GET /openapi/v1/applications/{application_id}/catch_all
```

参数：

名称	类型	说明
version	字符串	指明要从中获取策略的工作空间的版本。

返回表示给定版本工作空间的“全部捕获”策略的单个策略对象。

示例 Python 代码

```
application_id = '5f88c996755f023f3bafe163'
restclient.get('/applications/%s/catch_all' % application_id, params={'version': '1'})
```

获取特定策略

此终端会返回策略实例。

```
GET /openapi/v1/policies/{policy_id}
```

返回与指定 ID 关联的策略对象。

示例 Python 代码

```
policy_id = '5f88ca1e755f0222f85ce85c'
restclient.get('/policies/%s' % policy_id)
```

使用策略标识符搜索特定策略

此终端使用策略标识符参数作为复合密钥来搜索指定策略。

```
POST /openapi/v1/policies/search
```

查询正文包含具有以下架构的 JSON 正文：

名称	类型	说明
application_id	字符串	应用工作空间的 ID。
policy_identifier	对象	构成一致策略标识符的字段。

策略标识符字段使用以下架构组成：

名称	类型	说明
version	字符串	（可选）表示要为其获取策略的应用版本；未指定时，默认为应用的最新“v”版本。
consumer_consistent_uuid	字符串	使用者或源的一致 UUID。
provider_consistent_uuid	字符串	提供者或目标的一致 UUID。
rank	字符串	策略排名必须是“DEFAULT”或“ABSOUTE”之一。
action	字符串	策略操作必须是“ALLOW”或“DENY”之一。
priority	整数	策略的优先级值。
protocol	整数	的 IP 协议号 (0-255)。
start_port	整数	（可选）端口范围开始 (0-65535)；若未指定，则默认为 0
end_port	整数	（可选）端口范围结束 (0-65535)；如果 start_port 为 0，则默认为 65535，否则为 start_prot。

示例 Python 代码

```
application_id = '5f88ca1e755f0222f85ce85c'
consumer_id = '5f88ca1e755f0222f85ce85d'
provider_id = '5f88ca1e755f0222f85ce85d'
rank = 'DEFAULT'
action = 'ALLOW'
priority = 100
protocol = 6
start_port = 80
version = 'p3'

req_body = f'''
{{
    "application_id": "{application_id}",
    "policy_identifier": {{
        "consumer_consistent_uuid": "{consumer_id}",
        "provider_consistent_uuid": "{provider_id}",
        "rank": "{rank}",
        "action": "{action}",
        "priority": {priority},
        "protocol": "{protocol}",
        "start_port": "{start_port}",
        "version": "{version}"
    }}
}}'''
restclient.post('/policies/search', json_body=req_body)
```

创建策略

此终端用于创建新策略。

```
POST /openapi/v1/applications/{application_id}/policies
```

参数：

属性	类型	说明
consumer_filter_id	字符串	已定义过滤器的 ID。
provider_filter_id	字符串	已定义过滤器的 ID。.
version	字符串	指明要在其中更新策略的工作空间的版本。
排名	字符串	用于排名的值可以是 DEFAULT、ABSOLUTE 或 CATCHALL
policy_action	字符串	值可以是 ALLOW 或 DENY：表示是否应允许或丢弃在给定服务端口/协议上从使用者到提供者的流量
priority	整数	用于对策略进行排序。

示例 Python 代码

```
  req_payload = {
    "version": "v1",
    "rank" : "DEFAULT",
    "policy_action" : "ALLOW",
    "priority" : 100,
    "consumer_filter_id" : "123456789",
    "provider_filter_id" : "987654321",
  }
  resp = restclient.post('/openapi/v1/applications/{application_id}/policies',
json_body=json.dumps(req_payload))
```

创建默认策略

此终端用于创建新的默认策略。此终端创建的默认策略与创建策略终端类似。

```
POST /openapi/v1/applications/{application_id}/default_policies
```

创建绝对策略

此终端用于创建新的绝对策略。此终端创建的绝对策略与创建策略终端类似。

```
POST /openapi/v1/applications/{application_id}/absolute_policies
```

更新策略

此终端会更新策略。

```
PUT /openapi/v1/policies/{policy_id}
```

参数：

属性	类型	说明
consumer_filter_id	字符串	已定义过滤器的 ID。
provider_filter_id	字符串	已定义过滤器的 ID。
policy_action	字符串	可能的值可以是 ALLOW 或 DENY。指明对于使用者和提供者之间的给定服务端口/协议，是允许还是丢弃流量。
priority	整数	用于对策略优先级进行排序

返回与指定 ID 相关的已修改策略对象。

更新“全部捕获”

此终端会更新特定工作空间的“全部捕获”。

```
PUT /openapi/v1/applications/{application_id}/catch_all
```

参数：

属性	类型	说明
version	字符串	指明要在其中更新策略的工作空间的版本。
policy_action	字符串	可能的值可以是 ALLOW 或 DENY。指明是否允许或丢弃与此工作空间中的任何策略不匹配的流量。

将服务端口添加到策略

此终端用于为特定策略创建服务端口。

```
POST /openapi/v1/policies/{policy_id}/l4_params
```

参数：

属性	类型	说明
version	字符串	指明要从中获取策略的工作空间的版本。

属性	类型	说明
start_port	整数	范围的起始端口。
end_port	整数	范围的结束端口。
proto	整数	协议整数值（NULL 表示所有协议）。
description	字符串	（可选）有关此协议和端口的简短字符串。

更新策略的服务端口

此终端更新策略的指定服务端口。

```
PUT /openapi/v1/policies/{policy_id}/l4_params/{l4_params_id}
```

参数：

属性	类型	说明
approved	布尔值	将策略标记为已批准。

删除策略的服务端口

此终端将删除策略的指定服务端口。（可选）有关详细信息，请参阅 排除过滤器 。

```
DELETE /openapi/v1/policies/{policy_id}/l4_params/{l4_params_id}
```

参数：

属性	类型	说明
create_exclusion_filter	布尔值	（可选）如果为 true，则创建与策略匹配的排除过滤器。与此过滤器匹配的流将从未来的自动策略发现运行中排除。有关详细信息，请参阅 排除过滤器 。

删除策略

此终端将删除指定的策略。不创建排除过滤器。

```
DELETE /openapi/v1/policies/{policy_id}
```

删除包含标识符的策略

此终端使用策略标识符参数来删除指定的策略。不创建排除过滤器。

```
DELETE /openapi/v1/policies/destroy_with_identifier
```

查询正文包含具有以下架构的 JSON 正文：

名称	类型	说明
application_id	字符串	应用工作空间的 ID。
policy_identifier	对象	构成一致策略标识符的字段。

策略标识符字段使用以下架构组成：

名称	类型	说明
version	字符串	（可选）执行删除操作的应用工作空间的“v”版本；未指定时，默认为工作空间的最新“v”版本。
consumer_consistent_uuid	字符串	使用者或源的一致 UUID
provider_consistent_uuid	字符串	提供者或目标的一致 UUID
rank	字符串	策略排名必须是“DEFAULT”或“ABSOUTE”之一
action	字符串	策略操作必须是“ALLOW”或“DENY”之一
priority	整数	策略的优先级值
protocol	整数	策略的 IP 协议号 (0-255)
start_port	整数	（可选）端口范围开始 (0-65535)；若未指定，则默认为 0
end_port	整数	（可选）端口范围结束 (0-65535)；如果 start_port 为 0，则默认为 65535，否则为 start_prot

示例 Python 代码

```
application_id = '5f88ca1e755f0222f85ce85c'
consumer_id = '5f88ca1e755f0222f85ce85d'
provider_id = '5f88ca1e755f0222f85ce85d'
action = 'ALLOW'
rank = 'DEFAULT'
protocol = 6
start_port = 80
priority = 100
version = '5'
```

```
req_body = f'''
{{
    "application_id": "{application_id}",
    "policy_identifier": {{
        "consumer_consistent_uuid": "{consumer_id}",
        "provider_consistent_uuid": "{provider_id}",
        "rank": "{rank}",
        "priority": {priority},
        "action": "{action}",
        "protocol": "{protocol}",
        "start_port": "{start_port}",
        "version": "{version}"
    }}
}}'''
restclient.delete('/policies/destroy_with_identifier', json_body=req_body)
```

策略快速分析

此终端可用于为任何假设流找到与根范围中已分析/已执行策略相匹配的策略集。有关详细信息，请参阅 快速分析

此 API 仅适用于对根范围具有最低读取访问权限的用户，并且需要使用与 API 密钥关联的 `app_policy_management` 功能。

```
POST /openapi/v1/policies/{rootScopeID}/quick_analysis
```

查询正文包含具有以下架构的 JSON 正文：

名称	类型	说明
consumer_ip	字符串	客户端/使用者的 IP 地址。
provider_ip	字符串	服务器/提供者的 IP 地址。
provider_port	整数	（可选）提供者端口，仅与TCP或 UDP 流相关。
protocol	字符串	流的协议，例如 TCP。
analysis_type	字符串	分析类型可以是 **analyzed** 或 **enforced**。“analyzed”分析类型通过将流与根范围中的所有已分析策略进行匹配来做出流决策。“enforced”分析类型通过将流与根范围中的所有已执行策略进行匹配来做出流决策。

名称	类型	说明
application_id	字符串	（可选）主工作空间的 ID，如已指定，则总是伴随着工作空间“v”版本，通过使用指定版本的策略和根范围内其他工作空间的分析/执行策略来做出流决策。如果跳过此字段，则会通过考虑根范围内所有已分析/已执行的策略来做出流量决策。
version	整数	（可选）上述“v”版本的工作空间。如果指定了 application_id，则必须指定此项，否则必须跳过。

示例请求

请求的正文应为 JSON 格式的查询。

查询正文的一个示例，其中的流决策基于所有已分析的策略：

```
    req_payload = {
        "consumer_ip": "4.4.1.1",
        "provider_ip": "4.4.2.1",
        "provider_port": 9081,
        "protocol": "TCP",
        "analysis_type": "analyzed"
    }
    resp = restclient.post('/openapi/v1/policies/{rootScopeID}/quick_analysis',
json_body=json.dumps(req_payload))
```

查询正文的一个示例，其中流决策基于工作空间“v”版本的策略以及根范围内所有其他工作空间的已分析策略：

```
    req_payload = {
        "consumer_ip": "4.4.1.1",
        "provider_ip": "4.4.2.1",
        "provider_port": 9081,
        "protocol": "TCP",
        "analysis_type": "analyzed",
        "application_id": "5e7e5f56497d4f0bc26c7bb3",
        "version": 1
    }
    resp = restclient.post('/openapi/v1/policies/{rootScopeID}/quick_analysis',
json_body=json.dumps(req_payload))
```

响应示例

响应是正文中具有以下属性的 JSON 对象：

密钥	值
policy_decision	决定允许还是拒绝假设流。

密钥	值
outbound_policy	允许/拒绝传出流量的使用者的策略
inbound_policy	允许/拒绝传入流量的提供者的策略


```
{
    "policy_decision": "ALLOW",
    "outbound_policy": {
        "policy_rank": "DEFAULT",
        "start_port": 9082,
        "l4_detail_id": "5e7e600f497d4f7341f4f6d0",
        "src_filter_id": "5e7e600e497d4f7341f4f459",
        "end_port": 9082,
        "cluster_edge_id": "5e7e600f497d4f7341f4f6d1",
        "dst_filter_id": "5e7d0efc497d4f44b6b09351",
        "action": "ALLOW",
        "protocol": "TCP",
        "app_scope_id": "5e7e5f3a497d4f0bc26c7bb0"
    },
    "inbound_policy": {
        "policy_rank": "DEFAULT",
        "start_port": 9082,
        "l4_detail_id": "5e7e600f497d4f7341f4f6d0",
        "src_filter_id": "5e7e600e497d4f7341f4f459",
        "end_port": 9082,
        "cluster_edge_id": "5e7e600f497d4f7341f4f6d1",
        "dst_filter_id": "5e7d0efc497d4f44b6b09351",
        "action": "ALLOW",
        "protocol": "TCP",
        "app_scope_id": "5e7e5f3a497d4f0bc26c7bb0"
    }
}
```


策略统计信息

此终端会返回在一定时间间隔内观察到的策略数据包、字节和对话的数量。对话可以概括地描述为符合策略的流观测，其汇总粒度为一小时。在一小时内对特定策略进行测量的对话次数，表示在这一小时内通过网络进行通信的使用者和提供者资产项目的不同对数。

虽然此终端接受策略标识符参数作为输入，但建议您使用已发布的工作空间版本中的策略和L4参数ID。

注释 新版本的应用工作空间发布后，可能需要 6 个小时才能获得结果。所有时间戳解析的最小粒度也将为 6 小时。

要跨应用工作空间的执行版本获取策略的策略统计信息，URL 路径为：

```
POST /openapi/v1/policies/stats/enforced
```

要跨应用工作空间的已分析版本获取策略的策略统计信息，URL 路径为：

```
POST /openapi/v1/policies/stats/analyzed
```

查询正文包含具有以下架构的 JSON 正文：

表 *4:*

名称	类型	说明
application_id	字符串	应用工作空间的 ID。
t0	字符串	以 RFC-3339 格式表示的时间间隔开始。
t1	字符串	（可选）以 RFC-3339 格式表示的时间间隔结束时间；如果未指定，则默认为当前时间。
policy_id	字符串	策略的 ID；如果存在策略标识符，则不需要。
l4_param_id	字符串	l4 参数的 ID；如果存在策略标识符，或者对于“CATCH_ALL”策略，则不需要。
policy_identifier	对象	构成一致策略标识符的字段。

策略标识符字段使用以下架构组成：

名称	类型	说明
consumer_consistent_uuid	字符串	使用者或源的一致 UUID。
provider_consistent_uuid	字符串	提供者或目标的一致 UUID。
rank	字符串	策略排名必须是“DEFAULT”或“ABSOUTE”之一。
action	字符串	策略操作必须是“ALLOW”或“DENY”之一。
priority	整数	策略的优先级值。
protocol	整数	策略的 IP 协议号 (0-255)。
start_port	整数	（可选）端口范围开始 (0-65535)；若未指定，则默认为 0
end_port	整数	（可选）端口范围结束 (0-65535)；如果 start_port 为 0，则默认为 65535，否则为 start_prot。

示例 Python 代码

```
application_id = '5f88ca1e755f0222f85ce85c'
consumer_id = '5f88ca1e755f0222f85ce85d'
provider_id = '5f88ca1e755f0222f85ce85d'
action = 'ALLOW'
rank = 'DEFAULT'
protocol = 6
start_port = 80
priority = 100

req_body = f'''
{{
    "application_id": "{application_id}",
    "t0":"2022-07-06T00:00:00Z",
    "t1":"2022-07-28T19:00:00Z",
    "policy_identifier": {{
        "consumer_consistent_uuid": "{consumer_id}",
        "provider_consistent_uuid": "{provider_id}",
        "rank": "{rank}",
        "priority": {priority},
        "action": "{action}",
        "protocol": "{protocol}",
        "start_port": "{start_port}"
    }}
}}'''
restclient.post('/policies/stats/analyzed', json_body=req_body)

# For CATCH_ALL policies:
root_app_scope_id = '6f88ca1e755f0222f85ce85e'
rank = 'CATCH_ALL'
action = 'DENY'
req_body = f'''
{{
    "application_id": "{application_id}",
    "t0":"2022-07-06T00:00:00Z",
    "t1":"2022-07-28T19:00:00Z",
    "policy_identifier": {{
        "consumer_consistent_uuid": "{root_app_scope_id}",
        "provider_consistent_uuid": "{root_app_scope_id}",
        "rank": "{rank}",
        "action": "{action}"
    }}
}}'''

restclient.post('/policies/stats/analyzed', json_body=req_body)
```

响应示例

响应是正文中具有以下属性的 JSON 对象。

表 5:

密钥	值
conversation_count	在指定持续时间和策略内观察到的对话数。
packet_count	在指定持续时间和策略内观察到的数据包数。
byte_count	在指定的持续时间和策略内观察到的字节数。

密钥	值
first_seen_at	首次观察到此策略的流时的时间戳（RFC-3339 格式）。
last_seen_at	上次观察到此策略的流时的时间戳（采用 RFC-3339 格式）。
agg_start_version	有记录以来此策略的最早发布版本（自时间t0起）。
agg_start_time	发布 agg_start_version 的时间戳。

```
{
  "conversation_count": 72,
  "packet_count": 800,
  "byte_count": 1960,
  "first_seen_at": "2022-09-09T11:00:00.000Z",
  "last_seen_at": "2022-09-09T11:00:00.000Z",
  "agg_start_version": 4,
  "agg_start_time": "2022-08-10T23:00:00.000Z"
}
```

未使用的策略

此终端会返回已发布工作空间中在指定时间间隔内未观察到任何对话的策略标识符。

策略标识符

即使基础过滤器查询或策略端口和协议未更改，所有策略和 ADM 生成的集群也可以在不同的应用工作空间版本中更改其ID。为了跨工作空间版本跟踪特定策略的流命中计数，我们对不会在版本之间更改的过滤器使用一致的 UUID，并使用称为策略标识符的复合密钥，该密钥由提供者和使用者一致的 UUID 以及等级、操作、优先级、端口和协议。

因此，策略标识符是一种复合密钥，可以跨应用工作空间版本识别和描述策略的重要方面，而策略 ID（如常规 CRUD 终端中使用的策略 ID）则可能在工作空间的不同版本中发生变化。

注释 提供者/使用者一致的 UUID 和策略标识符不会唯一标识过滤器或策略，因为它们在不同的应用工作空间版本之间共享。

要对特定集群或策略执行 CRUD 操作，建议将标识符解析为特定应用工作空间版本来搜索终端的具体策略。

常规 CRUD 操作可使用策略标识符执行，而只有策略统计和使用标识符销毁操作接受策略标识符作为输入。这样做主要是为了方便，避免中间调用搜索，而是直接验证和销毁工作空间中所有未使用的策略。

强烈建议尽可能使用策略和过滤器 ID，不要为策略统计或带标识符的销毁 API 终端手动生成策略标识符。但是，以下示例说明了从策略对象生成策略标识符的一种方法：

```
resp = restclient.get(f'/policies/631b0590497d4f09b537b973')
```

```
policy = resp.json() # policy object
policy_identifer = {
    'consumer_consistent_uuid': policy['consumer_filter']['consistent_uuid'],
    'provider_consistent_uuid': policy['provider_filter']['consistent_uuid'],
    'rank': policy['rank'],
    'action': policy['action'],
    'priority': policy['priority'],
    'protocol': policy['l4_params'][0]['proto'],
    'start_port': policy['l4_params'][0]['port'][0],
    'end_port': policy['l4_params'][0]['port'][1]
}
```

注释 新版本的应用工作空间发布后，可能需要 6 个小时才能获得结果。所有时间戳解析的最小粒度也将为 6 小时。

要跨应用工作空间的执行版本获取未使用的策略，URL 路径为：

```
POST /openapi/v1/unused_policies/{application_id}/enforced
```

要跨应用工作空间的已分析版本获取未使用的策略，URL 路径为：

```
POST /openapi/v1/unused_policies/{application_id}/analyzed
```

查询正文包含具有以下架构的 JSON 正文：

名称	类型	说明
t0	字符串	以 RFC-3339 格式表示的时间间隔开始。
t1	字符串	（可选）以 RFC-3339 格式表示的时间间隔结束时间；如果未指定，则默认为当前时间。
limit	整数	（可选）限制每个请求的策略数。
offset	字符串	（可选）从上一个响应收到的偏移量 - 用于分页。

```
application_id = '62e1915e755f026f2bcdd805'
resp = restclient.post(f'/unused_policies/{application_id}/analyzed', json_body=f'''
{{
    "t0":"2022-07-06T00:00:00Z",
    "t1":"2022-07-28T19:00:00Z"
}}''')'
```

响应示例

响应是正文中具有以下属性的 JSON 对象。

密钥	值
application_id	应用工作空间的 ID。

密钥	值
policy_identifiers	未使用策略的策略标识符列表。
offset	要为下一页结果传递的响应偏移量。

要生成下一页结果，请获取响应中接收的对象的偏移，并将其作为下一个查询的偏移值传递。


```
{
  "application_id": "63054a97497d4f2dc113a9c4",
  "policy_identifiers": [
    {
      "consumer_consistent_uuid": "62fff45c497d4f5064973c4d",
      "provider_consistent_uuid": "62fff45c497d4f5064973c4d",
      "version": "p1",
      "rank": "DEFAULT",
      "policy_action": "ALLOW",
      "priority": 10,
      "proto": 6,
      "start_port": 10000,
      "end_port": 10000,
      "agg_start_version": 1,
      "agg_start_time": "2022-08-10T23:00:00.000Z"
    },
    {
      "consumer_consistent_uuid": "62fff45c497d4f5064973c4d",
      "provider_consistent_uuid": "62fff45c497d4f5064973c4d",
      "version": "p1",
      "rank": "DEFAULT",
      "policy_action": "ALLOW",
      "priority": 10,
      "protocol": 6,
      "start_port": 10001,
      "end_port": 10001,
      "agg_start_version": 1,
      "agg_start_time": "2022-08-10T23:00:00.000Z"
    }
  ],
  "offset": "eyJvZmZzZXQiOjZ9"
}
```


策略模板

这组 API 可用于添加、编辑或删除策略模板，并且需要与 API 密钥关联的 app_policy_management 功能。

获取策略模板

此终端会返回特定根范围的策略模板列表。此 API 可用于具有 `app_policy_management` 功能的 API 密钥。

```
GET /openapi/v1/application_templates?root_app_scope_id={root_app_scope_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
root_app_scope_id	字符串	根范围的唯一标识符。

响应对象：返回指定根范围的策略模板对象列表。

示例 python 代码

```
root_app_scope_id = '<root-app-scope-id>'
restclient.get('/application_templates?root_app_scope_id=%s' % root_app_scope_id)
```

获取特定策略模板

此终端会返回策略模板的实例。

```
GET /openapi/v1/application_templates/{template_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
template_id	字符串	策略模板的唯一标识符。

响应对象：返回具有指定 ID 的策略模板对象。

示例 python 代码

```
template_id = '<template-id>'
restclient.get('/application_templates/%s' % template_id)
```

创建策略模板

此终端用于创建新的策略模板。

```
POST /openapi/v1/application_templates
```

JSON 请求正文包含以下键

属性	类型	说明
name	字符串	在导入时用作模板的名称。
description	字符串	（可选）应用过程中显示的模板说明
parameters	参数对象	模板参数，请参阅以下内容。
absolute_policies	策略对象数组	（可选）绝对策略数组。
default_policies	策略对象数组	（必需）默认策略数组，可以为空。

响应对象：返回已创建的策略模板对象。

示例 python 代码

```
root_app_scope_id = '<root-app-scope-id>'
payload = {'root_app_scope_id': root_app_scope_id,
           'name': "policy_name",
           'default_policies': [
              {
                'action': 'ALLOW',
                'priority': 100,
                'l4_params': [
                  {
                    'proto': 17,
                    'port': [80, 90]
                  }
                ]
              }
            ]
          }
restclient.post('/application_templates',
                json_body=json.dumps(payload))
```

更新策略模板

此终端会更新策略模板。

```
PUT /openapi/v1/application_templates/{template_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
template_id	字符串	策略模板的唯一标识符。

JSON 请求正文包含以下键

属性	类型	说明
name	字符串	（可选）在导入期间用作模板的名称。
description	字符串	（可选）应用过程中显示的模板说明

响应对象：返回具有指定 ID 的已修改策略模板对象。

示例 python 代码

```
new_name = <new-name>
payload = {'name': new_name}
template_id = '<template-id>'
restclient.post('/application_templates/%s' % template_id,
                json_body=json.dumps(payload))
```

删除策略模板

此终端将删除指定的策略模板。

```
DELETE /openapi/v1/application_templates/{template_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
template_id	字符串	策略模板的唯一标识符。

响应对象：无

示例 python 代码

```
template_id = '<template-id>'
restclient.delete('/application_templates/%s' % template_id)
```

下载策略模板

此终端会下载策略模板。

```
GET /openapi/v1/application_templates/{template_id}/download
```

参数：请求 URL 包含以下参数

名称	类型	说明
template_id	字符串	策略模板的唯一标识符。

响应对象：返回具有指定 ID 的完整策略模板定义。

示例 python 代码

```
template_id = '<template-id>'
restclient.get('/application_templates/%s/download' % template_id)
```

集群

这组 API 可用于添加、编辑或删除作为工作空间（“应用”）成员的集群。它们需要使用与 API 密钥关联的 `user_role_scope_management` 功能。

集群对象

集群对象属性如下所述：

属性	类型	说明
id	字符串	集群的唯一标识符。
consistent_uuid	字符串	在自动策略发现运行期间保持一致的 ID。

属性	类型	说明
application_id	字符串	集群所属的工作空间的 ID。
version	字符串	集群所属工作空间的版本
name	字符串	集群的名称。
description	字符串	集群的说明。
approved	布尔值	如果集群已被用户“批准”。
query	JSON	与过滤器关联的过滤器（或匹配条件）以及父范围的过滤器。
short_query	JSON	与过滤器关联的过滤器（或匹配条件）。
alternate_queries	查询数组	自动策略发现生成的备用建议查询在动态模式下运行。
inventory	资产数组	如果请求，则会返回集群的成员资产，包括 IP、主机名、vrf_id 和 uuid。

获取集群

此终端会返回特定工作空间（“应用”）的集群列表。此 API 可用于具有 `app_policy_management` 功能的 API 密钥。

```
GET /openapi/v1/applications/{application_id}/clusters
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	集群所属的工作空间的 ID。
version	字符串	指明要为其获取集群的工作空间的版本。
include_inventory	布尔值	包括集群的资产。

响应对象：返回此特定工作空间和版本的所有集群的数组。

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
restclient.get('/applications/%s/clusters' % application_id)
```

获取特定集群

此终端节点会返回集群的实例。

```
GET /openapi/v1/clusters/{cluster_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
cluster_id	字符串	集群的唯一标识符。
include_inventory	布尔值	包括集群的资产。

响应对象：返回与指定 ID 关联的集群对象。

示例 python 代码

```
cluster_id = '5d02d021497d4f0949ba74e4'
restclient.get('/clusters/%s' % cluster_id)
```

创建集群

此终端用于创建新集群。

```
POST /openapi/v1/applications/{application_id}/clusters
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	集群所属的工作空间的 ID。

JSON 查询正文包含以下键

属性	类型	说明
name	字符串	集群的名称。
version	字符串	指明将向其添加集群的工作空间的版本。
description	字符串	（可选）集群的说明。
approved	布尔值	（可选）在自动策略发现运行期间，不会更新已批准的集群。默认值为 False。
query	JSON	与过滤器关联的过滤器（或匹配条件）。必须在工作空间上启用备用查询模式（也称为动态模式），否则将被忽略。

属性	类型	说明
query	JSON	与过滤器关联的过滤器（或匹配条件）。必须在工作空间上启用备用查询模式（也称为动态模式），否则将被忽略。
nodes	数组	IP地址或终端列表。将用于创建与这些 IP 匹配的查询，除非提供了查询且工作空间处于动态模式。

节点对象属性：

名称	类型	说明
ip	字符串	IP 地址
name	字符串	（可选）节点的名称。
prefix_len	整数	（可选）子网掩码。

Note 除非提供了查询且工作空间处于动态模式，否则节点将用于创建查询。

响应对象：返回新创建的集群对象。

示例 python 代码

```
application_id = '5d02b493755f0237a3d6e078'
payload = {
  'name': 'test_cluster',
  'version': 'v2',
  'description': 'basic granularity',
  'approved': False,
  'query': {
      'type': 'eq',
      'field': 'host_name',
      'value': 'centos6001'
  }
}
restclient.post('/applications/%s/clusters' % application_id)
```

更新集群

此终端会更新集群。

```
PUT /openapi/v1/clusters/{cluster_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
cluster_id	字符串	集群的唯一标识符。

JSON 查询正文包含以下键

属性	类型	说明
name	字符串	集群的名称。
description	字符串	（可选）集群的说明。
approved	布尔值	在自动策略发现运行期间，已批准的集群不会进行更新。
query	JSON	与过滤器关联的过滤器（或匹配条件）。必须在工作空间上启用备用查询模式（也称为动态模式），否则将被忽略。

响应对象：返回与指定 ID 关联的修改后的集群对象。

示例 python 代码

```
cluster_id = '5d02d2a4497d4f5194f104ef'
payload = {
  'name': 'new_test_cluster',
}
restclient.put('/clusters/%s' % cluster_id, json_body=json.dumps(payload))
```

删除集群

此终端将删除指定的集群。如果集群被任何策略使用，则集群将不会被删除，并将返回一个依赖关系列表。

```
DELETE /openapi/v1/clusters/{cluster_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
cluster_id	字符串	集群的唯一标识符。

响应对象：无

示例 python 代码

```
cluster_id = '5d02d2a4497d4f5194f104ef'
restclient.delete('/clusters/%s' % cluster_id)
```

对话

对话是自动策略发现运行的时间范围内的汇聚流，其中的使用者端口已被删除。有关对话的更详细说明，请参阅对话。

通过此 API，您可以搜索在自动策略发现运行期间为给定工作空间生成的对话。它需要使用与 API 密钥关联的 `app_policy_management` 功能才能调用此 API。

搜索策略发现运行中的对话

通过此终端，您可以在自动策略发现运行中搜索特定工作空间的对话。您还可以指定支持的维度和指标子集，您可能希望将其作为下载对话的一部分进行查看。或者，您还可以选择使用所支持维度和指标的过滤器来查询对话的子集。

```
POST /openapi/v1/conversations/{application_id}
```

查询包含具有以下键的 JSON 正文。

名称	类型	说明
version	整数	自动策略发现运行的版本
filter	JSON	（可选）查询过滤器。如果过滤器为空（即 {}），则 query 匹配所有对话。可以使用支持的维度和指标上的过滤器下载更具体的对话。有关过滤器的语法，请参阅 过滤器 。
dimensions	数组	（可选）要为下载的对话返回的维度列表。可在 支持的维度 找到支持的维度列表。
metrics	数组	（可选）要返回的已下载对话的指标列表。可在 支持的指标 找到支持的指标列表。
limit	整数	（可选）要在单个 API 响应中返回的对话数。
offset	字符串	（可选）从上一个响应收到的偏移量 - 用于分页。

请求的正文应为 JSON 格式的查询。查询正文的示例如下所示。

```
{
    "version": 1,
    "filter": {
        "type": "and",
        "filters":[
            {
```

```
                    "type": "eq",
                    "field": "excluded",
                    "value": False
                },
                {
                    "type": "eq",
                    "field": "protocol",
                    "value": "TCP"
                },
            ]
        },
        "dimensions": ["src_ip", "dst_ip", "port"],
        "metrics": ["byte_count", "packet_count"],
        "limit" : 2,
        "offset": <offset-object>
    }
```

解决方案

响应是正文中具有以下属性的 JSON 对象。

密钥	值
offset	要为下一页结果传递的响应偏移量
results	结果列表

要生成下一页结果，请获取响应中接收的对象的偏移，并将其作为下一个查询的偏移值传递。

```
   req_payload = {"version": 1,
                  "limit": 10,
                  "filter": {"type": "and",
                      "filters": [
                          {"type": "eq", "field": "excluded", "value": False},
                          {"type": "eq", "field": "protocol", "value": "TCP"}
                       ]
                  }
    }

    resp = restclient.post('/conversations/{application_id}',
json_body=json.dumps(req_payload))
    print resp.status_code
    if resp.status_code == 200:
        parsed_resp = json.loads(resp.content)
        print json.dumps(parsed_resp, indent=4, sort_keys=True)
```

策略发现运行中的前 N 个对话

通过此终端，您可以搜索自动策略发现排名靠前的对话，该策略发现是根据指标和维度对给定工作空间进行分组后运行的。当前支持的指标位于 支持的指标，当前支持的按维度分组位于支持的维度。您可以使用支持的维度和指标上的过滤器来查询对话的子集。例如，您可以使用包含 `src_ip` 维度和 `byte_count` 指标的查询来搜索具有最多字节流量通信的源 IP 地址。

```
POST /openapi/v1/conversations/{application_id}/topn
```

查询包含具有以下键的 JSON 正文。

名称	类型	说明
version	整数	自动策略发现运行的版本
dimension	字符串	要作为前N个查询的分组依据的对话的维度。 支持的维度：src_ip、dst_ip
metric	字符串	前N个对话的排序依据的指标。可在 支持的指标 找到支持的指标列表。
filter	JSON	（可选）查询过滤器。如果过滤器为空（即 {}），则 query 匹配所有对话。可以使用支持的维度和指标上的过滤器下载更具体的对话。有关过滤器的语法，请参阅 过滤器 。
threshold	整数	要在单个 API 响应中返回的前 N 个结果的数量。

请求的正文应为 JSON 格式的查询。查询正文的示例如下所示。

```
{
    "version": 1,
    "dimension": "src_ip",
    "metric": "byte_count",
    "filter": {
        "type": "and",
        "filters":[
            {
                "type": "eq",
                "field": "excluded",
                "value": False
            },
            {
                "type": "eq",
                "field": "protocol",
                "value": "TCP"
            },
        ]
    },
    "threshold" : 10
}
```

解决方案

响应是正文中具有以下属性的 JSON 对象。

密钥	值
results	包含一个 JSON 对象的列表，其中包含一个结果键和一个结果对象列表的值，结果对象的键与查询维度和度量相匹配。

```
  [ {"result": [
    {
      "byte_count": 1795195565,
      "src_ip": "192.168.1.6"
    },
    {
      "byte_count": 1781002379,
      "src_ip": "192.168.1.28"
    },
    ...
  ] } ]
```

```
    req_payload = {"version": 1, "dimension": "src_ip", "metric": "byte_count",
      "filter": {"type": "and",
        "filters": [
          {"type": "eq", "field": "excluded", "value": False},
          {"type": "eq", "field": "protocol", "value": "TCP"},
         {"type": "eq", "field": "consumer_filter_id", "value": "16b12a5614c5af5b68afa7ce"},

          {"type": "subnet", "field": "src_ip", "value": "192.168.1.0/24"}
        ]
      },
      "threshold" : 10
    }

    resp = restclient.post('/conversations/{application_id}/topn',
json_body=json.dumps(req_payload))
    print resp.status_code
    if resp.status_code == 200:
        parsed_resp = json.loads(resp.content)
        print json.dumps(parsed_resp, indent=4, sort_keys=True)
```

支持的维度

名称	类型	说明
src_ip	字符串	使用者的 IP 地址
dst_ip	字符串	提供者的 IP 地址
protocol	字符串	通信中使用的协议。例如：“TCP”、“UDP”等
port	整数	提供者的端口。
address_type	字符串	“IPv4”或“IPv6”

名称	类型	说明
consumer_filter_id	字符串	如果使用者 IP 属于集群，则为集群的集群 ID，否则为使用者 IP 所属的范围 ID。
provider_filter_id	字符串	如果提供者 IP 属于集群，则为集群的集群 ID，否则为提供者 IP 所属的范围 ID。
excluded	布尔值	生成策略时是否排除此对话。
confidence	双宽	使用者和提供者分类的可信度。值在 0.0 到 1.0 之间变化，1.0 表示分类的可信度更高。

支持的指标

名称	类型	说明
byte_count	整数	对话中的总字节数
packet_count	整数	对话中的数据包总数

排除过滤器

这组 API 可用于添加、编辑或删除排除过滤器，并且需要使用与 API 密钥关联的 `user_role_scope_management` 功能。

排除过滤器将流排除在自动策略发现集群算法之外。有关详细信息，请参阅排除过滤器。

排除过滤器对象

排除过滤器对象属性如下所述：

属性	类型	说明
id	字符串	集群的唯一标识符。
application_id	字符串	排除过滤器所属的工作空间的 ID。
version	字符串	排除过滤器所属的工作空间的版本。

属性	类型	说明
consumer_filter_id	字符串	已定义过滤器的 ID。目前，属于工作空间、用户定义的过滤器或范围的任何集群都可以用作策略的使用者。
provider_filter_id	字符串	已定义过滤器的 ID。目前，属于工作空间、用户定义的过滤器或范围的任何集群都可以用作策略的提供者。
proto	整数	协议整数值（NULL 表示所有协议）。
port	数组	端口的包含范围。例如 [80, 80] 或 [5000, 6000]。NULL 表示所有端口。
updated_at	整数	更新排除过滤器时的 Unix 时间戳。

获取排除过滤器

此终端会返回特定工作空间的排除过滤器列表。此 API 可用于具有 `app_policy_management` 功能的 API 密钥。

```
GET /openapi/v1/applications/{application_id}/exclusion_filters
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。
version	字符串	指明要为其获取排除过滤器的工作空间的版本。

响应对象：返回指定工作空间和版本的排除过滤器对象列表。

示例 python 代码

```
application_id = '<application-id>'
params = {'version': 'v10'}
restclient.get('/applications/%s/exclusion_filters' % application_id,
               params=params)
```

获取特定排除过滤器

此终端会返回排除过滤器的实例。

```
GET /openapi/v1/exclusion_filters/{exclusion_filter_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
exclusion_filter_id	字符串	排除过滤器的唯一标识符。

响应对象：返回具有指定 ID 的排除过滤器对象。

示例 python 代码

```
exclusion_filter_id = '<exclusion-filter-id>'
restclient.get('/exclusion_filters/%s' % exclusion_filter_id)
```

创建排除过滤器

此终端用于创建新的排除过滤器。

```
POST /openapi/v1/applications/{application_id}/exclusion_filters
```

参数：请求 URL 包含以下参数

名称	类型	说明
application_id	字符串	工作空间的唯一标识符。

JSON 请求正文包含以下键

属性	类型	说明
version	字符串	排除过滤器所属的工作空间的版本。
consumer_filter_id	字符串	（可选）已定义过滤器的ID。目前，属于工作空间、用户定义的过滤器或范围的任何集群都可以用作策略的使用者。
provider_filter_id	字符串	（可选）已定义过滤器的ID。目前，属于工作空间、用户定义的过滤器或范围的任何集群都可以用作策略的提供者。
proto	整数	（可选）协议整数值（NULL 表示所有协议）。
start_port	整数	（可选）范围的起始端口。
end_port	整数	（可选）范围的结束端口。

缺少的可选参数将被视为通配符（匹配任意参数）。

响应对象：返回已创建的排除过滤器对象。

示例 python 代码

```
provider_filter_id = '<provider-filter-id>'
```

```
consumer_filter_id = '<consumer-filter-id>'
payload = {'version': 'v0',
           'consumer_filter_id': consumer_filter_id,
           'provider_filter_id': provider_filter_id,
           'proto': 6,
           'start_port': 800,
           'end_port': 1000}
application_id = '<application-id>'
restclient.post('/applications/%s/exclusion_filters' % application_id,
                json_body=json.dumps(payload))
```

更新排除过滤器

此终端会更新排除过滤器。

```
PUT /openapi/v1/exclusion_filters/{exclusion_filter_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
exclusion_filter_id	字符串	排除过滤器的唯一标识符。

JSON 请求正文包含以下键

属性	类型	说明
consumer_filter_id	字符串	（可选）已定义过滤器的ID。目前，属于工作空间、用户定义的过滤器或范围的任何集群都可以用作策略的使用者。
provider_filter_id	字符串	（可选）已定义过滤器的ID。目前，属于工作空间、用户定义的过滤器或范围的任何集群都可以用作策略的提供者。
proto	整数	协议整数值（NULL 表示所有协议）。
start_port	整数	（可选）范围的起始端口。
end_port	整数	（可选）范围的结束端口。

响应对象：返回具有指定 ID 的修改后的排除过滤器对象。

示例 python 代码

```
payload = {'proto': 17}
exclusion_filter_id = '<exclusion-filter-id>'
restclient.post('/exclusion_filters/%s' % exclusion_filter_id,
                json_body=json.dumps(payload))
```

删除排除过滤器

此终端将删除指定的排除过滤器。

```
DELETE /openapi/v1/exclusion_filters/{exclusion_filter_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
exclusion_filter_id	字符串	排除过滤器的唯一标识符。

响应对象：无

示例 python 代码

```
exclusion_filter_id = '<exclusion-filter-id>'
restclient.delete('/exclusion_filters/%s' % exclusion_filter_id)
```

默认排除过滤器

这组 API 可用于添加、编辑或删除默认排除过滤器，并且需要使用与 API 密钥关联的 `app_policy_management` 功能。

排除过滤器将流排除在自动策略发现集群算法之外。有关详细信息，请参阅排除过滤器。

默认排除过滤器对象

排除过滤器对象属性如下所述：

属性	类型	说明
id	字符串	默认排除过滤器的唯一标识符。
consumer_filter_id	字符串	已定义过滤器的 ID。目前，属于工作空间、用户定义的过滤器或范围的任何集群都可以用作策略的使用者。
provider_filter_id	字符串	已定义过滤器的 ID。目前，属于工作空间、用户定义的过滤器或范围的任何集群都可以用作策略的提供者。
proto	整数	协议整数值（NULL 表示所有协议）。
port	数组	端口的包含范围。例如 [80, 80] 或 [5000, 6000]。NULL 表示所有端口。
updated_at	整数	更新排除过滤器时的 Unix 时间戳。

获取默认排除过滤器

此终端会返回默认排除过滤器列表。此 API 可用于具有 `app_policy_management` 功能的 API 密钥。

```
GET /openapi/v1/default_exclusion_filters?root_app_scope_id={root_app_scope_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
root_app_scope_id	字符串	根范围的唯一标识符。

响应对象：返回根范围的默认排除过滤器对象列表。

示例 python 代码

```
root_app_scope_id = '<root-app-scope-id>'
 restclient.get('/default_exclusion_filters?root_app_scope_id=%s' % root_app_scope_id)
```

获取特定默认排除过滤器

此终端会返回默认排除过滤器的实例。

```
default_exclusion_filter_id = '<default-exclusion-filter-id>'

restclient.get('/default_exclusion_filters/%s' % default_exclusion_filter_id)
```

参数：请求 URL 包含以下参数

名称	类型	说明
default_exclusion_filter_id	字符串	排除过滤器的唯一标识符。

响应对象：返回具有指定 ID 的默认排除过滤器对象。

示例 python 代码

```
  default_exclusion_filter_id = '<default-exclusion-filter-id>'
  restclient.get('/default_exclusion_filters/%s' % default_exclusion_filter_id)
```

创建默认排除过滤器

此终端用于创建新的默认排除过滤器。

```
POST /openapi/v1/default_exclusion_filters?root_app_scope_id={root_app_scope_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
root_app_scope_id	字符串	根范围的唯一标识符。

JSON 请求正文包含以下键

属性	类型	说明
consumer_filter_id	字符串	（可选）已定义范围或资产过滤器的 ID。

属性	类型	说明
provider_filter_id	字符串	（可选）已定义范围或资产过滤器的 ID。
proto	整数	（可选）协议整数值（NULL 表示所有协议）。
start_port	整数	（可选）范围的起始端口。
end_port	整数	（可选）范围的结束端口。

响应对象：返回已创建的默认排除过滤器对象。

示例 python 代码

```
provider_filter_id = '<provider-filter-id>'
consumer_filter_id = '<consumer-filter-id>'
payload = {'consumer_filter_id': consumer_filter_id,
           'provider_filter_id': provider_filter_id,
           'proto': 6,
           'start_port': 800,
           'end_port': 1000}
root_app_scope_id = '<root-app-scope-id>'
restclient.post('/default_exclusion_filters?root_app_scope_id=%s' % root_app_scope_id,
                json_body=json.dumps(payload))
```

更新默认排除过滤器

此终端会更新默认排除过滤器。

```
PUT /openapi/v1/default_exclusion_filters/{default_exclusion_filter_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
default_exclusion_filter_id	字符串	默认排除过滤器的唯一标识符。

JSON 请求正文包含以下键

属性	类型	说明
consumer_filter_id	字符串	（可选）已定义范围或资产过滤器的ID。
provider_filter_id	字符串	（可选）已定义范围或资产过滤器的ID。
proto	整数	协议整数值（NULL 表示所有协议）。
start_port	整数	（可选）范围的起始端口。

end_port	整数	（可选）范围的结束端口。

响应对象：返回具有指定 ID 的已修改默认排除过滤器对象。

示例 python 代码

```
payload = {'proto': 17}
default_exclusion_filter_id = '<default-exclusion-filter-id>'
restclient.post('/default_exclusion_filters/%s' % default_exclusion_filter_id,
                json_body=json.dumps(payload))
```

删除默认排除过滤器

此终端将删除指定的默认排除过滤器。

```
DELETE /openapi/v1/default_exclusion_filters/{default_exclusion_filter_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
default_exclusion_filter_id	字符串	排除过滤器的唯一标识符。

响应对象：无

示例 python 代码

```
default_exclusion_filter_id = '<default-exclusion-filter-id>'
restclient.delete('/default_exclusion_filters/%s' % default_exclusion_filter_id)
```

实时分析

实时分析或策略分析是生成安全策略的一个重要方面。在对工作负载实际执行这些策略之前，它可以让您评估一组策略的影响—无论是通过自动策略发现生成的，还是由用户手动添加的。实时分析允许用户在不中断任何应用流量的情况下对实时流量进行假设分析。

本部分提供的一组 API 允许下载流以及工作空间中当前已发布策略集对这些流的影响。它需要使用与 API 密钥关联的 `app_policy_management` 功能才能调用这组 API。

通过实时分析提供的流具有一些属性（维度和指标），而下载 API 允许用户根据维度上的不同标准过滤流。

实时分析中可用的流维度

此终端有助于了解可针对哪些列指定搜索条件（或过滤器）以通过实时分析下载可用流。最常见的使用案例是下载允许、转义或拒绝的流，可以通过将类别维度的搜索条件传递到下载 API 来实现。与 **type: eq** 一起使用时，流的入站和出站类别必须匹配。与 **type: contains** 一起使用时，流入站或出站类别必须匹配

```
GET /openapi/v1/live_analysis/dimensions
```

实时分析中可用的流指标

此终端会返回与实时分析相关的指标列表（例如，字节数、数据包数）。此终端的一个使用案例是在下载 API 中预测指标子集，即用户可以指定他们感兴趣的一小部分指标，而不是下载所有指标。

```
GET /openapi/v1/live_analysis/metrics
```

下载通过实时分析可用的流

此终端会返回匹配过滤条件的流列表。结果中的每个流对象的属性都是实时分析维度（由上述实时分析维度 API 返回）和实时分析指标（由上述实时分析指标 API 返回）的组合。或者，如果用户对全部可用维度和指标不感兴趣，也可以选择指定一小部分维度或度量子集 - 这种对较小维度或度量子集的投影也有使 API 调用更快的副作用。

```
POST /openapi/v1/live_analysis/{application_id}
```

查询正文包含具有以下键的 JSON 正文。

名称	类型	说明
t0	整数或字符串	时间间隔开始（纪元或 ISO 8601）
t1	整数或字符串	时间间隔结束（纪元或 ISO 8601）
filter	JSON	查询过滤器。如果过滤器为空（即 {}），则 query 匹配所有流。有关过滤器的语法，请参阅流搜索中的过滤器部分。
dimensions	数组	（可选）通过实时分析提供的下载流量要返回的流维度列表。如果未指定，则会返回所有可用维度。
metrics	数组	（可选）通过实时分析提供的下载流量要返回的流指标列表。
limit	整数	（可选）在单个 API 响应中返回的流量数。
offset	字符串	（可选）从上一个响应收到的偏移量 - 用于分页。

请求的正文应为 JSON 格式的查询。查询正文的示例如下所示。

```
{
    "t0": "2016-06-17T09:00:00-0700",
    "t1": "2016-06-17T17:00:00-0700",
```

```
        "filter": {
            "type": "and",
            "filters": [
                {
                    "type": "contains",
                    "field": "category",
                    "value": "escaped"
                },
                {
                    "type": "in",
                    "field": "dst_port",
                    "values": ["80", "443"]
                }
            ]
        },
        "limit": 100,
        "offset": <offset-object>
    }
```

解决方案

响应是正文中具有以下属性的 JSON 对象。

密钥	值
offset	要为下一页结果传递的响应偏移量
results	结果列表

要生成下一页结果，请获取响应中接收的对象的偏移，并将其作为下一个查询的偏移值传递。

示例 python 代码

```
   req_payload = {"t0": "2016-11-07T09:00:00-0700",
                  "t1": "2016-11-07T19:00:00-0700",
                  "limit": 10,
                  "filter": {"type": "and",
                      "filters": [
                          {"type": "contains", "field": "category", "value": "escaped"},
                          {"type": "regex", "field": "src_hostname", "value": "web*"}
                       ]
                  }
    }

    resp = restclient.post('/live_analysis/{application_id}',
json_body=json.dumps(req_payload))
    print resp.status_code
    if resp.status_code == 200:
        parsed_resp = json.loads(resp.content)
        print json.dumps(parsed_resp, indent=4, sort_keys=True)
```

范围

这组 API 可用于管理 Cisco Secure Workload 集群部署中的范围（或 AppScope）。它们需要使用与 API 密钥关联的 `user_role_scope_management` 功能。用于获取范围列表的 API 也可用于具有 `app_policy_management` 或 `sensor_management` 功能的 API 密钥。

范围对象

范围对象属性如下所述：

属性	类型	说明
id	字符串	范围的唯一标识符。
short_name	字符串	用户指定的范围名称。
name	字符串	范围的完全限定名称。这是一个完全限定名称，即它具有从父范围（如果适用）一直到根范围的名称。
description	字符串	用户指定的范围说明。
short_query	JSON	与范围关联的过滤器（或匹配条件）。
query	JSON	与范围关联的过滤器（或匹配条件）以及父范围的过滤器（一直到根范围）。
vrf_id	整数	范围所属的 VRF 的 ID。
parent_app_scope_id	字符串	父范围的 ID。
child_app_scope_ids	数组	子范围 ID 的数组。
policy_priority		用于对工作空间优先级进行排序。请参阅语义和查看。
dirty	布尔值	表示子查询或父查询已更新，需要提交更改。
dirty_short_query	JSON	如果此范围的查询已更新但尚未提交，则为非 null。

获取范围

此终端会返回 Cisco Secure Workload 设备已知的范围列表。此 API 可用于具有 `app_policy_management` 或 `user_role_scope_management` 功能的 API 密钥。

```
GET /openapi/v1/app_scopes
```

参数：

名称	类型	说明
vrf_id	整数	按 vrf_id 匹配应用范围。
root_app_scope_id	字符串	按根应用范围 ID 来匹配应用范围。
exact_name	字符串	返回与确切名称匹配的范围（区分大小写）。
exact_short_name	字符串	返回与确切的短名称匹配的范围（区分大小写）。

返回范围对象列表。

创建范围

此终端用于创建新的范围。

```
POST /openapi/v1/app_scopes
```

参数：

名称	类型	说明
short_name	字符串	用户指定的范围名称。
description	字符串	用户指定的范围说明。
short_query	JSON	与范围关联的过滤器（或匹配条件）。
parent_app_scope_id	字符串	父范围的 ID。
policy_priority	整数	默认值为“last”。用于对工作空间优先级进行排序。请参阅策略下的“策略排序”。

示例 python 代码

```
 req_payload = {
    "short_name": "App Scope Name",
    "short_query": {
        "type":"eq",
        "field":"ip",
        "value": <....>
    },
    "parent_app_scope_id": <parent_app_scope_id>
}
resp = restclient.post('/app_scopes', json_body=json.dumps(req_payload))
```

要基于子网创建范围，请使用以下 short_query：

```
"short_query":
  {
    "type":"subnet",
    "field":"ip",
    "value": "1.0.0.0/8"
  },
```

获取特定范围

此终端节点返回范围的实例。

```
GET /openapi/v1/app_scopes/{app_scope_id}
```

返回与指定 ID 关联的范围对象。

更新范围

此终端更新范围。对名称和说明所做的更改会立即应用。对 short_query 的更改会将范围标记为“dirty”，并设置 `dirty_short_query` 属性。在给定根范围下进行所有范围查询更改后，需要对提交范围查询更改终端执行 ping 操作，以提交所有必需的更新。

```
PUT /openapi/v1/app_scopes/{app_scope_id}
```

参数：

名称	类型	说明
short_name	字符串	用户指定的范围名称。
description	字符串	用户指定的范围说明。
short_query	JSON	与范围关联的过滤器（或匹配条件）。

返回与指定 ID 关联的修改后的范围对象。

删除特定范围

此终端将删除指定的范围。

```
DELETE /openapi/v1/app_scopes/{app_scope_id}
```

如果范围与工作空间、策略、用户资产过滤器等关联，则此终端将返回 `422 Unprocessable Entity`。返回的错误对象将包含一个详细信息属性，该属性具有从属对象的计数以及每种类型的前 10 个对象的 ID。此信息可用于查找和删除阻止依赖关系。

按策略优先级顺序获取范围

此终端按其相应主工作空间的执行顺序列出范围。

```
GET /openapi/v1/app_scopes/{root_app_scope_id}/policy_order
```

返回范围对象数组。

更新策略顺序

此终端将更新应用策略的顺序。

Warning 此终端会更改应用策略的顺序。因此，将在相关主机上插入新的主机防火墙规则并删除任何现有规则。

```
POST /openapi/v1/app_scopes/{root_app_scope_id}/policy_order
```

参数：

名称	类型	说明
root_app_scope_id	字符串	根范围或正在更改顺序的范围。
ids	数组	范围 ID 字符串数组。

`ids` 数组参数必须包括根范围的所有成员，包括根。

提交范围查询更改

此终端会触发异步后台作业，以更新给定根范围内的所有“dirty”子项。此作业更新范围和工作空间，有关详细信息，请参阅 范围 。

```
POST /openapi/v1/app_scopes/commit_dirty
```

参数：

名称	类型	说明
root_app_scope_id	字符串	将更新其所有子项的根范围的 ID。
sync	布尔值	（可选）指明请求是否应同步。

返回 202 以指明作业已进入队列。要检查作业是否已完成，请轮询根范围的“dirty”属性，以便查看其是否已设置为 false。

用户可以传递 `sync` 参数以使作业立即运行。完成后将返回请求，并显示 200 状态代码。如果需要应用许多更新，此请求可能需要一些时间。

提交小组建议请求

为范围提交组建议请求。

```
PUT /openapi/v1/app_scopes/{app_scope_id}/suggest_groups
```

参数：请求 URL 包含以下参数

名称	类型	说明
app_scope_id	字符串	范围的唯一标识符。

参数：JSON 查询正文包含以下键

名称	类型	说明
start_time	字符串	组建议输入时间间隔的开始时间。
end_time	字符串	组建议输入时间间隔的结束时间。

响应对象：返回具有以下属性的对象：

名称	类型	说明
message	字符串	有关提交组建议请求成功/失败的消息。

示例 python 代码

```
app_scope_id = '5d02b493755f0237a3d6e078'
req_payload = {
      'start_time': '2020-09-17T10:00:00-0700',
      'end_time': '2020-09-17T11:00:00-0700',
}
resp = restclient.put('/app_scopes/%s/suggest_groups' % app_scope_id,
                      json_body=json.dumps(req_payload))
```

获取组建议状态

范围的查询组建议状态。

```
GET /openapi/v1/app_scopes/{app_scope_id}/suggest_groups_status
```

参数：请求 URL 包含以下参数

名称	类型	说明
app_scope_id	字符串	范围的唯一标识符。

响应对象：返回具有以下属性的对象：

名称	类型	说明

status	字符串	组建议的状态。值：PENDING、COMPLETE 或 FAILED

示例 python 代码

```
app_scope_id = '5d02b493755f0237a3d6e078'
resp = restclient.get('/app_scopes/%s/suggest_groups_status' % app_scope_id)
```

配置告警

这组 API 可用于管理用户警报。它们需要使用与 API 密钥关联的 `user_alert_management` 功能。

警报对象

每个警报配置对象包含以下字段：

属性	类型	说明
app_name	字符串	与警报配置关联的应用名称。
rules	对象	警报配置必须满足的一组条件才能触发警报。
subjects	对象	应收到警报的用户列表。
severity	字符串	指明与警报配置关联的严重性级别。
individual_alert	布尔值	指明是否应发送触发警报配置的单个警报。
summary_alert_freq	字符串	为特定警报配置发送摘要警报的频率。
alert_type	字符串	警报配置的唯一标识符。

属性	类型	说明
app_instance_id	字符串	与警报配置关联的特定实例的唯一标识符

获取警报

此终端会检索用户的警报配置列表。警报可过滤至给定的根范围。如果未提供范围，则会返回用户有权访问的所有范围的所有警报。只有当用户是站点管理员时，才会返回运营商警报。

```
GET /openapi/v1/alert_confs
```

参数：

名称	类型	说明
root_app_scope_id	字符串	（可选）根范围的 ID，仅返回分配给该范围的警报。

响应对象：返回用户警报对象列表。

创建警报

此终端用于创建新警报。

```
POST openapi/v1/alert_confs
```

参数：

属性	类型	说明
app_name	字符串	与警报配置关联的应用名称。
rules	对象	警报配置必须满足的一组条件才能触发警报。
subjects	对象	应收到警报的用户列表。
severity	字符串	指明与警报配置关联的严重性级别。
individual_alert	布尔值	指明是否应发送触发警报配置的单个警报。
summary_alert_freq	字符串	为特定警报配置发送摘要警报的频率。
alert_type	字符串	警报配置的唯一标识符。

属性	类型	说明
app_instance_id	字符串	与警报配置关联的特定实例的唯一标识符。

发出请求的用户必须有权访问所提供的范围。没有范围的警报是“运营商警报”，只有站点管理员可以创建它们。

响应对象：返回新创建的警报对象。

获取特定警报

此终端会返回特定警报对象。

```
GET /openapi/v1/alert_confs/
```

参数：请求 URL 包含以下参数

名称	类型	说明
alert_id	字符串	唯一地标识警报。

响应对象：返回与指定 ID 关联的警报对象。

更新警报

此终端用于更新现有警报。

```
PUT /openapi/v1/alert_confs/
```

参数：请求 URL 包含以下参数

名称	类型	说明
alert_id	字符串	检索或修改警报的配置设置。

JSON 请求正文包含以下参数

名称	类型	说明
name	字符串	用户为警报指定的名称。
description	字符串	用户指定的警报说明。

发出请求的用户必须有权访问所提供的范围。没有范围的警报称为“运营商警报”，只有站点管理员可以更新它们。

响应对象：具有指定 ID 的已更新警报对象。

删除特定警报

此终端将删除指定的警报。

```
DELETE /openapi/v1/alert_confs/{alert_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
alert_id	字符串	唯一地标识警报。

响应对象：无。

角色

这组 API 可用于管理用户角色。它们需要使用与 API 密钥关联的 `user_role_scope_management` 功能。

Note 这些 API 仅适用于站点管理员和根范围的所有者。

角色对象

角色对象属性如下所述：

属性	类型	说明
id	字符串	角色的唯一标识符。
app_scope_id	字符串	定义范围的范围，对于“运营商角色”，它可能为空。
name	字符串	用户为角色指定的名称。
description	字符串	用户为角色指定的说明。

获取角色

此终端会返回用户可访问的角色列表。角色可过滤至给定的根范围。如果未提供范围，则会返回用户有权访问的所有范围的所有角色。只有当用户是站点管理员时，才会返回运营商角色。

```
GET /openapi/v1/roles
```

参数：

名称	类型	说明
app_scope_id	字符串	（可选）根范围的 ID，仅返回分配给该范围的角色。

响应对象：返回用户角色对象列表。

示例 python 代码

```
resp = restclient.get('/roles')
```

创建角色

此终端用于创建新角色。

```
POST /openapi/v1/roles
```

参数：

名称	类型	说明
name	字符串	用户为角色指定的名称。
description	字符串	用户为角色指定的说明。
app_scope_id	字符串	（可选）创建角色时所用的范围 ID。如果未提及范围 ID，则会将角色视为运营商角色。

发出请求的用户必须有权访问所提供的范围。没有范围的角色称为“运营商角色”，只有站点管理员可以创建它们。

响应对象：返回新创建的角色对象。

示例 python 代码

```
app_scope_id = '<app-scope-id>'
req_payload = {
      'name': 'Role Name',
      'description': 'Role Description',
      'app_scope_id': app_scope_id
}
restclient.post('/roles', json_body=json.dumps(req_payload))
```

获取特定角色

此终端会返回特定角色对象。

```
GET /openapi/v1/roles/{role_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
role_id	字符串	唯一地标识角色。

响应对象：返回与指定 ID 关联的角色对象。

示例 python 代码

```
role_id = '<role-id>'
restclient.get('/roles/%s' % role_id)
```

更新角色

此终端用于更新现有角色。

```
PUT /openapi/v1/roles/{role_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
role_id	字符串	唯一地标识角色。

JSON 请求正文包含以下参数

名称	类型	说明
name	字符串	用户为角色指定的名称。
description	字符串	用户为角色指定的说明。

发出请求的用户必须有权访问所提供的范围。没有范围的角色称为“运营商角色”，只有站点管理员才能更新这些角色。

响应对象：具有指定 ID 的已更新角色对象。

示例 python 代码

```
role_id = '<role-id>'
req_payload = {
      'name': 'Role Name',
      'description': 'Role Description',
}
restclient.put('/roles/%s' % role_id, json_body=json.dumps(req_payload))
```

授予角色访问范围的权限

此终端可为角色提供对范围的指定访问级别。

```
POST /openapi/v1/roles/{role_id}/capabilities
```

只能将功能添加到用户有权访问的角色。如果角色被分配给一个范围，则功能必须与该范围或其子范围相对应。运营商角色（未分配给某个范围的角色）可以为任何范围添加功能。

参数：请求 URL 包含以下参数

名称	类型	说明
role_id	字符串	唯一地标识角色。

JSON 请求正文包含以下参数

名称	类型	说明
app_scope_id	字符串	提供访问权限的范围的 ID。
ability	字符串	可能的值包括 SCOPE_READ、 SCOPE_WRITE、EXECUTE、ENFORCE、 SCOPE_OWNER、DEVELOPER

有关功能的更多说明，请参阅 角色 。

响应对象：

名称	类型	说明
app_scope_id	字符串	提供访问权限的范围的 ID。
role_id	字符串	角色的 ID。
ability	字符串	可能的值包括 SCOPE_READ、SCOPE_WRITE、EXECUTE、ENFORCE、SCOPE_OWNER、DEVELOPER
inherited	布尔值	

示例 python 代码

```
role_id = '<role-id>'
req_payload = {
  'app_scope_id': '<app-scope-id>',
  'ability': 'SCOPE_READ'
}
restclient.post('/roles/%s/capabilities' % role_id,
                json_body=json.dumps(req_payload))
```

删除特定角色

此终端将删除指定角色。

```
DELETE /openapi/v1/roles/{role_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
role_id	字符串	唯一地标识角色。

响应对象：无。

示例 python 代码

```
role_id = '<role-id>'
restclient.delete('/roles/%s' % role_id)
```

用户

这组 API 用于管理用户。它们需要使用与 API 密钥关联的 `user_role_scope_management` 功能。

Note 这些 API 仅适用于站点管理员和根范围的所有者。

用户对象

用户对象属性如下所述：

属性	类型	说明
id	字符串	用户角色的唯一标识符。
email	字符串	与用户帐户关联的邮箱。
first_name	字符串	名字。
last_name	字符串	姓氏。
app_scope_id	字符串	为用户分配的范围。如果用户是“运营商用户”，则可能为空。
role_ids	列表	分配给用户帐户的角色的 ID 列表。
by-pass_external_auth	布尔值	本地用户为 true，外部身份验证用户（ldap 或 sso）为 false。
disabled_at	整数	禁用用户时的 Unix 时间戳。否则为零或 null。

获取用户

此终端会返回 Cisco Secure Workload 设备已知的用户对象列表。

```
GET /openapi/v1/users
```

参数：请求 URL 包含以下参数

名称	类型	说明
include_disabled	布尔值	（可选）要包括已禁用的用户，默认值为 false。
app_scope_id	字符串	（可选）只返回分配给所提供范围的用户。

响应对象：返回用户对象列表。只有站点管理员可以查看“运营商用户”，即未分配到范围的用户。

示例 python 代码

```
GET /openapi/v1/users
```

创建新用户帐户

此终端用于创建新的用户帐户。

```
POST /openapi/v1/users
```

参数：JSON 请求正文包含以下参数

名称	类型	说明
email	字符串	与用户帐户关联的邮箱。
first_name	字符串	名字。
last_name	字符串	姓氏。
app_scope_id	字符串	（可选）用户所属的根范围。
role_ids	列表	（可选）应分配给用户的角色列表。

app_scope_id 是要为用户分配的根范围的 ID。如果 app_ scope_id 不存在，则用户是“运营商用户”。只有站点管理员才能创建运营商用户。 `role_ids` 是在指定应用范围下创建的角色的 ID。

响应对象：返回新创建的用户对象。

示例 python 代码

```
req_payload = {
    "first_name": "fname",
    "last_name": "lname",
```

```
    "email": "foo@bar.com"
    "app_scope_id": "root_appscope_id",
    "role_ids": ["roleid1", "roleid2"]
}
resp = restclient.post('/users', json_body=json.dumps(req_payload))
```

获取特定用户

此终端会返回特定用户对象。

```
GET /openapi/v1/users/{user_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
user_id	字符串	用户对象的 ID。

响应对象：返回与指定 ID 关联的用户对象。

示例 python 代码

```
user_id = '5ce480db497d4f1ca1fc2b2b'
resp = restclient.get('/users/%s' % user_id)
```

更新用户

此终端会更新现有用户。

```
PUT /openapi/v1/users/{user_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
user_id	字符串	正在更新的用户对象的 ID。

JSON 请求正文包含以下参数

名称	类型	说明
email	字符串	与用户帐户关联的邮箱。
first_name	字符串	名字。
last_name	字符串	姓氏。
app_scope_id	字符串	根应用范围ID（仅允许站点管理员使用）

响应对象：返回最新更新的用户对象。

示例 python 代码

```
req_payload = {
    "first_name": "fname",
    "last_name": "lname",
    "email": "foo@bar.com"
    "app_scope_id": "root_appscope_id",
}
restclient.put('/users', json_body=json.dumps(req_payload))
```

启用/重新激活已停用的用户

此终端用于重新启用已停用的用户。

```
POST /openapi/v1/users/{user_id}/enable
```

参数：请求 URL 包含以下参数

名称	类型	说明
user_id	字符串	正在启用的用户对象的 ID。

响应对象：返回与指定 ID 关联的已重新激活的用户对象。

示例 python 代码

```
user_id = '5ce480db497d4f1ca1fc2b2b'
resp = restclient.post('/users/%s/enable' % user_id)
```

向用户帐户添加角色

此终端用于向用户帐户添加角色。

```
PUT /openapi/v1/users/{user_id}/add_role
```

参数：请求 URL 包含以下参数

名称	类型	说明
user_id	字符串	正在修改的用户对象的 ID。

JSON 请求正文包含以下参数

名称	类型	说明
role_id	字符串	要添加的角色对象的 ID。

响应对象：返回与指定 ID 关联的已修改用户对象。

示例 python 代码

```
user_id = '5ce480db497d4f1ca1fc2b2b'
req_payload = {
    "role_id": "5ce480d4497d4f1c155d0cef",
}
resp = restclient.put('/users/%s/add_role' % user_id,
                      json_body=json.dumps(req_payload))
```

从用户帐户中删除角色

此终端用于从用户帐户中删除角色。

```
DELETE /openapi/v1/users/{user_id}/remove_role
```

参数：请求 URL 包含以下参数

名称	类型	说明
user_id	字符串	正在删除的用户对象的 ID。

JSON 请求正文包含以下参数

名称	类型	说明
role_id	字符串	要删除的角色对象的 ID。

响应对象：返回与指定 ID 关联的已修改用户对象。

示例 python 代码

```
user_id = '5ce480db497d4f1ca1fc2b2b'
req_payload = {
    "role_id": "5ce480d4497d4f1c155d0cef",
}
resp = restclient.delete('/users/%s/remove_role' % user_id,
                         json_body=json.dumps(req_payload))
```

删除特定用户

此终端将删除指定的用户帐户。

```
DELETE /openapi/v1/users/{user_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
user_id	字符串	正在删除的用户对象的 ID。

响应对象：返回与指定 ID 关联的已删除用户对象。

示例 python 代码

```
user_id = '5ce480db497d4f1ca1fc2b2b'
resp = restclient.delete('/users/%s' % user_id)
```

资产过滤器

资产过滤器对资产搜索查询的匹配条件进行编码。这组 API 提供的功能类似于资产 过滤器 中所述的功能。它们需要使用与 API 密钥关联的 `sensor_management or app_policy_management` 功能。

资产过滤器对象

资产过滤器 JSON 对象以单个对象或对象数组的形式返回，具体取决于 API 终端。对象的属性如下所述：

属性	类型	说明
id	字符串	资产过滤器的唯一标识符。
name	字符串	用户指定的资产过滤器名称。
app_scope_id	字符串	与过滤器关联的范围的 ID。
short_query	JSON	与过滤器关联的过滤器（或匹配条件）。
primary	布尔值	如果为“true”，则过滤器仅限制为所有权范围。
public	布尔值	设置为“true”时，过滤器会为其范围提供服务。还必须是主要或范围受限。
query	JSON	与过滤器关联的过滤器（或匹配条件）以及父范围的过滤器。如果选中“仅限制为所有权范围”(restricted to ownership scope)复选框，则这些连接将生效。如果“primary”字段为 false，则查询与 short_query 相同。

获取资产过滤器

此终端会返回用户可见的资产过滤器列表。

```
GET /openapi/v1/filters/inventories
```

参数：

名称	类型	说明
vrf_id	整数	按 VRF ID 匹配资产过滤器。
root_app_scope_id	字符串	按根应用范围 ID 匹配资产过滤器。
name	字符串	返回与名称的一部分匹配的资产过滤器（不区分大小写）。
exact_name	字符串	返回与确切名称匹配的资产过滤器（区分大小写）。

创建资产过滤器

此终端用于创建资产过滤器。

```
POST /openapi/v1/filters/inventories
```

参数：

名称	类型	说明
name	字符串	用户指定的应用范围名称。
query	JSON	与过滤器关联的过滤器（或匹配条件）。
app_scope_id	字符串	与过滤器关联的范围的 ID。
primary	布尔值	如果为“true”，则过滤器仅限制为所有权范围。
public	布尔值	设置为“true”时，过滤器会为其范围提供服务。还必须是主要/范围限制。

示例 python 代码

```
req_payload = {
  "app_scope_id": <app_scope_id>,
  "name": "sensor_config_inventory_filter",
  "query": {
    "type": "eq",
    "field": "ip",
    "value": <sensor_interface_ip>
  },
}
resp = restclient.post('/filters/inventories', json_body=json.dumps(req_payload))
```

验证资产过滤器查询

此终端将根据所需架构来验证查询的结构。

```
POST /openapi/v1/filters/inventories/validate_query
```

参数：

名称	类型	说明
查询	JSON	与范围关联的过滤器（或匹配条件）。

响应对象：

属性	类型	说明
valid	布尔值	指明查询是否有效
errors	数组	如果无效，则提供有关错误的详细信息

获取特定资产过滤器

此终端会返回资产过滤器的实例。

```
GET /openapi/v1/filters/inventories/{inventory_filter_id}
```

返回与指定 ID 关联的资产过滤器对象。

更新特定资产过滤器

此终端用于更新资产过滤器。

```
PUT /openapi/v1/filters/inventories/{inventory_filter_id}
```

参数：

名称	类型	说明
name	字符串	用户指定的范围名称。
查询	JSON	与范围关联的过滤器（或匹配条件）。
app_scope_id	字符串	与过滤器关联的范围的 ID。
primary	布尔值	如果为“true”，则过滤器仅限制为所有权范围。

名称	类型	说明
public	布尔值	为“true”时，过滤器将提供服务。可用作策略生成的一部分。还必须是主要/范围限制。
Usages	布尔值	成员策略和配置统计信息的收集。

删除特定资产过滤器

此终端将删除指定的资产过滤器。

```
DELETE /openapi/v1/filters/inventories/{inventory_filter_id}
```

流搜索

流搜索功能提供与流中所述的类似功能。这些API集需要使用与API密钥关联的`flow_Inventory_query`功能。

流维度的查询

此终端会返回可以为流搜索查询（见下文）指定搜索条件（或过滤器）的流列的列表。有关列说明的详细信息，请参阅列和过滤器。

```
GET /openapi/v1/flowsearch/dimensions
```

参数：无

响应对象：

名称	类型	说明
dimensions	字符串列表	用户上传的维度和协调器维度的列表。

示例 python 代码

```
restclient.get('/flowsearch/dimensions')
```

流指标查询

此终端会返回与流观察结果关联的指标列表，例如字节计数和数据包计数。

```
GET /openapi/v1/flowsearch/metrics
```

参数：无

响应对象：

名称	类型	说明
metrics	字符串列表。	可用指标的列表。

示例 python 代码

```
restclient.get('/flowsearch/metrics')
```

流的查询

此终端会返回匹配过滤条件的流列表。结果中的每个流对象都具有以下属性：流维度（由上述流维度 API 返回）以及流指标（由上述流指标 API 返回）的并集。

`POST /openapi/v1/flowsearch`

可通过 `/openapi/v1/flowsearch/dimensions` API 来获取可在过滤条件中指定的列的列表。

参数：查询正文包含具有以下键的 JSON 正文。

名称	类型	说明
t0	整数或字符串	流搜索开始时间（纪元或 ISO 8601）
t1	整数或字符串	流搜索结束时间（纪元或 ISO 8601）
filter	JSON	查询过滤器。如果过滤器为空（即 {}），则 query 匹配所有流。
scopeName	字符串	查询限制到的范围的全称。
dimensions	数组	（可选）要在 flowsearch API 的结果中返回的维度名称列表。这是一个可选参数。如果未指定，flowsearch 结果将返回所有可用维度。当调用方不关心其余维度时，此选项可用于指定可用维度的子集。
metrics	数组	（可选）要在 flowsearch API 的结果中返回的指标名称列表。这是一个可选参数。如果未指定，flowsearch 结果将返回所有可用指标。如果调用方不关心其余指标，则此选项可用于指定可用指标的子集。

名称	类型	说明
limit	整数	（可选）响应流数限制。
offset	字符串	（可选）从先前响应接收的偏移对象。
descending	布尔值	（可选）如果此参数为 false 或未指定，则结果按时间戳升序排列。如果参数值为 true，则结果按时间戳降序排列。

请求的正文应为 JSON 格式的查询。查询正文的示例如下所示。

```
{
    "t0": "2016-06-17T09:00:00-0700",
    "t1": "2016-06-17T17:00:00-0700",
    "filter": {
        "type": "and",
        "filters": [
            {
                "type": "contains",
                "field": "dst_hostname",
                "value": "prod"
            },
            {
                "type": "in",
                "field": "dst_port",
                "values": ["80", "443"]
            }
        ]
    },
    "scopeName": "Default:Production:Web",
    "limit": 100,
    "offset": <offset-object>
}
```

过滤器

过滤器支持基元过滤器和由一个或多个基元过滤器组成的逻辑过滤器（“not”、“and”、“or”）。基元过滤器的格式如下：

```
{"type" : "<OPERATOR>", "field": "<COLUMN_NAME>", "value": "<COLUMN_VALUE>"}
```

对于基元过滤器，运算符可以是比较运算符，例如`eq`、`ne`、`lt`、`lte`、`gt`或`gte`。运算符也可以是`in`、`regex`、`subnet`、`contains`或`range`。

基元过滤器的一些示例可能包括：

```
{"type": "eq", "field": "src_address", "value": "7.7.7.7"}

{"type": "regex", "field": "src_hostname", "value": "prod.*"}

{"type": "subnet", "field": "src_addr", "value": "1.1.11.0/24"}

# Note, 'in' clause uses 'values' key instead of 'value'
```

```
{"type": "in", "field": "src_port", "values": [80, 443]}
```

您还可以使用布尔运算（例如 not、and 或 or）来指定复杂过滤器。以下是这些类型的过滤器的一些示例：

```
# "and" and "or" operators need to specify list of "filters"
{"type": "and",
    "filters": [
        {"type": "in", "field": "src_port", "values": [80, 443]},
        {"type": "regex", "field": "src_hostname", "value": "prod.*"}
    ]
}

# "not" operator needs to specify a "filter"
{"type": "not",
    "filter": {"type": "subnet", "field": "src_addr", "value": "1.1.11.0/24"}
}
```

更正式地说，流搜索请求中的过滤器架构如下所示：

密钥	值
type	过滤器类型
field	基元过滤器的过滤器字段列
filter	过滤器对象（仅用于非过滤器类型）
filters	过滤器对象列表（用于 `and` 和 `or` 过滤器类型）
value	基元过滤器的值
values	过滤器类型 `in` 或 `range` 的原始过滤器的值列表

基元过滤器类型

eq, ne - 在“`field`”指定的列中，使用“`value`”指定的值，分别搜索流是否相等或不等。支持以下字段：`src_hostname`、`dst_hostname`、`src_address`、`dst_address`、`src_port`、`dst_port`、`src_ scope_name`、`dst_ scope_name`、`vrf_name`、`src_enforcement_epg_name`、`dst_enforcement_epg_name`、`proto`。这些运算符也适用于用户标记的列。

lt, lte, gt, gte - 搜索“`field`”指定的列的值小于、小于等于、大于或大于等于“`value`”指定的值（如适用）的流。支持以下字段：`[src_port, dst_port]`。

range - 在“values”列表指定的范围开始和范围结束之间搜索“field”指定列的值（对于“*range*”过滤器类型，此列表的大小必须为 2 - 第一个值是范围开始，第二个值是范围结束）。支持以下字段：[src_port, dst_port]。

in - 在“`field`”指定的列中搜索流中的成员身份，成员身份列表由“`values`”指定。支持以下字段：`src_hostname`、`dst_hostname`、`src_address`、`dst_address`、`src_port`、`dst_port`、`src_ scope_name`、`dst_ scope_name`、`vrf_name`、`src_enforcement_epg_name`、`dst_enforcement_epg_name`、`proto`。此运算符也适用于用户标记的列。

regex, contains - 在“field”指定的列中，使用“value”来指定的正则表达式在流中搜索正则表达式匹配或包含匹配。支持以下字段：src_hostname、dst_hostname、src_ scope_name、dst_ scope_name、vrf_name、src_enforcement_epg_name、dst_enforcement_epg_name。这些运算符也适用于用户标记的列。正则表达式类型的过滤器必须使用 Java 样式的正则表达式模式作为“值”。

subnet - 在流中搜索由“field”指定为 CIDR 表示法的字符串的子网成员关系。支持以下字段：["src_address", "dst_address"]

逻辑过滤器类型

- **not** -“filter”指定的对象的逻辑“not”过滤器。
- **and** -“filters”指定的过滤器对象列表的逻辑“and”过滤器。
- **or** -“filters”指定的过滤器对象列表的逻辑“or或”过滤器。

响应对象：

密钥	值
offset	要为下一页结果传递的响应偏移量
results	结果列表

要生成下一页结果，请获取响应中接收的对象的偏移，并将其作为下一个查询的偏移值传递。

示例 python 代码

```
req_payload = {"t0": "2016-11-07T09:00:00-0700",
               "t1": "2016-11-07T19:00:00-0700",
               "scopeName": "Default:Prod:Web",
               "limit": 10,
               "filter": {"type": "and",
                   "filters": [
                     {"type": "subnet", "field": "src_address", "value": "1.1.11.0/24"},

                        {"type": "regex", "field": "src_hostname", "value": "web*"}
                    ]
               }
 }

 resp = restclient.post('/flowsearch', json_body=json.dumps(req_payload))
 print resp.status_code
 if resp.status_code == 200:
     parsed_resp = json.loads(resp.content)
     print json.dumps(parsed_resp, indent=4, sort_keys=True)
```

流的 TopN 查询

此终端会返回指定维度的值的前 N 个排序列表，其中列表中的排名由指定指标的汇聚确定。

```
POST /openapi/v1/flowsearch/topn
```

参数：

可通过 /openapi/v1/flowsearch/dimensions API 来获取可在过滤条件中指定的列的列表。请求的正文应为 JSON 格式的查询。查询正文的示例如下所示。请求正文中的参数 t0 和 t1 可以是纪元格式或 ISO 8601 格式。TopN API 仅允许查询一天的最大时间范围。应通过维度指定必须在其上进行分组的维度。应在 JSON 正文的 metric 字段中指定前 N 个结果排名所依据的指标。您应指定一个最小值为 1 的阈值，表示“TopN”中的“N”。此阈值的最大值为 1000。即使用户指定的值超过 1000，API 也最多只能返回 1000 个结果。此外，还必须指定一个名为 scopeName 的参数，它是要限制搜索的范围的全称。过滤器与流搜索 过滤器, on page 79 的过滤器相同。如果未提及过滤器，则将 topN 应用于所有流条目。

```
{
  "t0": "2016-06-17T09:00:00-0700",     # t0 can also be 1466179200
  "t1": "2016-06-17T17:00:00-0700",     # t1 can also be 1466208000
  "dimension": "src_address",
  "metric": "fwd_pkts",
 "filter": {"type": "eq", "field": "src_address", "value": "172.29.203.193"},  #optional

  "threshold": 5,
  "scopeName": "Default"
}
```

查询正文包含具有以下键的 JSON 正文。

密钥	值
t0	流的开始时间（纪元或 ISO 8601）
t1	流的结束时间（纪元或 ISO 8601）
filter	查询过滤器。如果过滤器为空（即 {}）或过滤器不存在（可选），则对所有流条目应用 topN 查询
scopeName	查询限制到的范围的全称
dimension	维度是我们用于分组的字段。
metric	指标是维度值的总数。
threshold	阈值为 topN 中的 N。

响应对象：

密钥	值
result	前 N 个条目的数组

示例 python 代码

```
req_payload = {
    "t0": "2017-06-07T08:20:00-07:00",
    "t1": "2017-06-07T14:20:00-07:00",
    "dimension": "src_address",
```

```
    "metric": "fwd_pkts",
    "filter": {"type": "ne", "field": "src_address", "value": "172.29.203.193"},
    "threshold": 5,
    "scopeName": "Default"
}
resp = rc.post('/flowsearch/topn',
               json_body=json.dumps(req_payload))
print resp.status_code
if resp.status_code == 200:
   parsed_resp = json.loads(resp.content)
   print json.dumps(parsed_resp)
```

响应示例

```
[
  { "result": [
    {"src_address": "172.31.239.163", "fwd_pkts": 23104},
    {"src_address": "172.31.239.162", "fwd_pkts": 22410},
    {"src_address": "172.31.239.166", "fwd_pkts": 16185},
    {"src_address": "172.31.239.168", "fwd_pkts": 15197},
    {"src_address": "172.31.239.169", "fwd_pkts": 15116}
    ]
  }
]
```

流计数

此终端会返回与指定条件匹配的流观察结果的数量。

`POST /openapi/v1/flowsearch/count`

参数：

请求的正文应为 JSON 格式的查询。查询正文的示例如下所示。请求正文中的参数 `t0` 和 `t1` 可以是纪元格式或 ISO 8601 格式。此 API 仅允许查询一天的最大时间范围。此外，您需要指定 `scopeName` 参数，该参数是要限制搜索的范围的全称。如果未指定此参数，则流观察结果计数 API 请求适用于您具有读取访问权限的所有范围。 过滤器与流搜索 过滤器 的过滤器相同。

```
  {
    "t0": "2016-06-17T09:00:00-0700",    # t0 can also be 1466179200
    "t1": "2016-06-17T17:00:00-0700",    # t1 can also be 1466208000
    "filter": {"type": "eq", "field": "src_address", "value": "172.29.203.193"},
    "scopeName": "Default"
  }
```

查询正文包含具有以下键的 JSON 正文。

密钥	值
t0	流的开始时间（纪元或 ISO 8601）
t1	流的结束时间（纪元或 ISO 8601）
filter	查询过滤器。如果过滤器为空（即 {}），则query匹配所有流。
scopeName	查询限制到的范围的全称

响应对象：

密钥	值
count	与流搜索条件匹配的流观察结果的数量。

示例 python 代码

```
req_payload = {
      "t0": "2017-07-20T08:20:00-07:00",
      "t1": "2017-07-20T10:20:00-07:00",
      "scopeName": "Tetration",
      "filter": {
                  "type": "eq",
                  "field": "dst_port",
                  "value": "5642"
          }
}
resp = rc.post('/flowsearch/count',
               json_body=json.dumps(req_payload))
print resp.status_code
if resp.status_code == 200:
   parsed_resp = json.loads(resp.content)
   print json.dumps(parsed_resp)
```

响应示例

```
{"count":508767}
```

资产

资产搜索 API 可提供与资产搜索中所述的类似功能。这些 API 集需要使用与 API 密钥关联的 `flow_Inventory_query` 功能。

资产维度的查询

此终端会返回可以为资产搜索查询指定搜索条件（或过滤器）的资产列的列表。

`GET /openapi/v1/inventory/search/dimensions`

资产搜索

此终端会返回与指定条件匹配的资产项目列表。

`POST /openapi/v1/inventory/search`

可以使用 /openapi/v1/Inventory/search/dimensions API 来获取可在过滤条件中指定的列的列表。

参数：

名称	类型	说明
filter	JSON	过滤器查询。
scopeName	字符串	（可选）限制结果的范围名称。
limit	整数	（可选）要返回的最大结果数。
offset	整数	（可选）与上一个请求的偏移量，用于获取下一页。

请求的正文必须为 JSON 格式的查询。查询正文的示例如下所示。

```
{
    "filter": {
        "type": "contains",
        "field": "hostname",
        "value": "collector"
    },
    "scopeName": "Default:Production:Web",  // optional
    "limit": 100,
    "offset": "<offset-object>"               // optional
}
```

要获取支持的不同类型的过滤器，请参阅过滤器, on page 79

查询正文包含具有以下键的 JSON 正文。

密钥	值
filter	查询过滤器。如果过滤器为空（即 {}），则query匹配所有资产项目。
scopeName	查询限制到的范围的全称（可选）
dimensions	要在资产搜索 API 的结果中返回的维度名称列表。这是一个可选参数。如果未指定，结果将返回所有可用的维度。当调用方不关心其余维度时，此选项可用于指定可用维度的子集。
limit	响应项目数限制（可选）
offset	从上一个响应中收到的偏移对象（可选）

响应

响应是正文中具有以下属性的 JSON 对象。

名称	类型	说明
offset	整数	要为下一页结果传递的响应偏移量。
results	对象数组	结果列表。

响应可能包含用于分页响应的 `offset` 字段。用户需要在后续请求中指定相同的偏移量，才能获取下一组结果。

示例 Python 代码

```
req_payload = {
  "scopeName": "Tetration", # optional
  "limit": 2,
  "filter": {"type": "and",
     "filters": [
        {"type": "eq", "field": "vrf_name", "value": "Tetration"},
        {"type": "subnet", "field": "ip", "value": "1.1.1.0/24"},
        {"type": "contains", "field": "hostname", "value": "collector"}
     ]
  }
}

resp = restclient.post('/inventory/search', json_body=json.dumps(req_payload))
print resp.status_code
if resp.status_code == 200:
   parsed_resp = json.loads(resp.content)
   print json.dumps(parsed_resp, indent=4, sort_keys=True)
```

资产统计信息

此终端会返回资产项目的统计信息。

```
GET /openapi/v1/inventory/{id}/stats?t0=<t0>&t1=<t1>&td=<td>
```

Table 6:

路径参数	说明描述
id	资产项目 ID 为 `{ip}-{vrf_id}`，例如 `1.1.1.1-123`

查询参数	说明
t0	以纪元时间表示的统计信息开始时间
t1	以纪元时间表示的统计信息结束时间
td	统计信息汇聚的粒度。整数指定秒数。可以传递字符串，例如“minute”、“hour”和“day”。

示例 Python 代码

```
resp = restclient.get('/inventory/1.1.1.1-123/stats?t0=1483228800&t1=1485907200&td=day')
```

资产计数

此终端会返回与指定条件匹配的资产项目计数。

```
POST /openapi/v1/inventory/count
```

可以使用 `/openapi/v1/Inventory/search/dimensions` API 来获取可在过滤条件中指定的列的列表。

参数：

名称	类型	说明
filter	JSON	过滤器查询。
scopeName	字符串	（可选）限制结果的范围名称。

请求的正文必须为 JSON 格式的查询。查询正文的示例如下所示。

```
{
    "filter": {
        "type": "and",
        "filters": [
            {
                "type": "contains",
                "field": "hostname",
                "value": "prod"
            },
            {
                "type": "subnet",
                "field": "ip"
                "value": "6.6.6.0/24"
            }
        ]
    },
    "scopeName": "Default:Production:Web",  # optional
}
```

响应

响应是正文中具有以下属性的 JSON 对象。

Table 7:

密钥	值
count	与过滤条件匹配的资产项目数

示例 python 代码

```
req_payload = {
  "scopeName": "Tetration", # optional
  "filter": {"type": "and",
     "filters": [
        {"type": "eq", "field": "vrf_name", "value": "Tetration"},
        {"type": "subnet", "field": "ip", "value": "1.1.1.0/24"},
        {"type": "contains", "field": "hostname", "value": "collector"}
     ]
  }
}

resp = restclient.post('/inventory/count', json_body=json.dumps(req_payload))
print resp.status_code
if resp.status_code == 200:
   parsed_resp = json.loads(resp.content)
   print json.dumps(parsed_resp, indent=4, sort_keys=True)
```

资产漏洞

此终端会返回与易受攻击的工作负载关联的 IP 地址对应的 CVE。

此 API 仅适用于对根范围具有最低读取访问权限的用户。

```
POST /openapi/v1/inventory/cves/{rootScopeID}
```

参数：

名称	类型	说明
ips	字符串列表	要获取 CVE 信息的 IP 列表。

请求的正文必须为 JSON 格式的查询。查询正文的示例如下所示。

```
    {
        "ips": [
            "10.18.187.72",
            "10.18.187.73"
        ]
    }
```

响应

响应是正文中具有以下属性的 JSON 对象数组。

名称	类型	说明
ip	字符串	IP 地址
cve_ids	字符串列表	资产中具有 IP 地址的 CVE ID 的列表。

示例 Python 代码

```
  root_scope_id = "5fa0d242497d4f7d968c669b"
  req_payload = {
    "ips":["10.18.187.72", "10.18.187.73"]
  }

  resp = restclient.post('/inventory/cves/' + root_scope_id,
json_body=json.dumps(req_payload))
  print resp.status_code
  if resp.status_code == 200:
     parsed_resp = json.loads(resp.content)
     print json.dumps(parsed_resp, indent=4, sort_keys=True)
```

检索恶意 IP 地址

终端会返回恶意 IP 地址的信息。此 API 可用于具有 `flow_Inventory_query` 功能的 API 密钥。

```
GET /openapi/v1/malicious_ips
```

请求 URL 包含以下参数：

路径参数	说明
offset	返回响应之前要跳过的记录数，默认为 0，允许的最大值为 100000。
limit	要返回的记录数，默认为 50，允许的最大值为 100000。

工作负载

工作负载 API 提供对工作负载配置文件 (Workload Profile) 页面内容的编程访问。这组 API 需要使用与 API 密钥关联的 sensor_management 或 flow_Inventory_query 功能。

工作负载详细信息

此终端会返回给定代理 UUID 的特定工作负载。

```
GET /openapi/v1/workload/{uuid}
```

路径参数	说明
uuid	代理 UUID

响应

响应是与指定 UUID 关联的工作负载对象。工作负载对象的属性架构如下所述：

Table 8:

属性	类型	说明
agent_type	整数	枚举中的代理类型
agent_type_str	字符串	纯文本代理类型
auto_upgrade_opt_out	布尔值	如果为 true，则代理不会在集群升级时自动升级
cpu_quota_mode	整数	CPU 配额控制
cpu_quota_us	整数	CPU 配额使用情况
current_sw_version	字符串	在工作负载上运行的代理软件的版本
data_plane_disabled	布尔值	如果为 true，则流遥测数据不会从代理导出到集群

属性	类型	说明
desired_sw_version	字符串	打算在工作负载上运行的代理软件版本
enable_conversation_flows	布尔值	如果为 true，则启用对话模式。
enable_cache_sidechannel	布尔值	如果为 true，则启用侧信道攻击检测
enable_forensics	布尔值	如果为 true，则启用取证
enable_meltdown	布尔值	如果为 true，则启用崩溃漏洞检测
enable_pid_lookup	布尔值	如果为 true，则启用进程查找
forensics_cpu_quota_mode	整数	取证 CPU 配额控制
forensics_cpu_quota_us	整数	取证配额使用情况
forensics_mem_quota_bytes	整数	取证内存配额（以字节为单位）
host_name	字符串	工作负载上的主机名
接口	数组	接口对象数组
kernel_version	字符串	内核版本
last_config_fetch_at	整数	上次获取配置的时间
last_software_update_at	整数	最后一个软件是代理报告其当前版本的时间戳
max_rss_limit	整数	最大内存限制
platform	字符串	工作负载的平台
uuid	字符串	代理的唯一 ID
windows_enforcement_mode	字符串	Windows 执行模式的类型、WAF（Windows 高级防火墙）或 WFP（Windows 过滤平台）

示例 Python 代码

```
agent_uuid = 'aa28b304f5c79b2f22d87a5af936f4a8fa555894'
resp = restclient.get('/workload/%s' % (agent_uuid))
```

工作负载统计信息

此终端会返回工作负载的统计信息。

```
GET /openapi/v1/workload/{uuid}/stats?t0=<t0>&t1=<t1>&td=<td>
```

路径参数	说明
uuid	代理 UUID

查询 URL 包含以下参数

查询参数	说明
t0	以纪元时间表示的统计信息开始时间
t1	以纪元时间表示的统计信息结束时间。结束时间不能超过开始时间一天。
td	统计信息汇聚的粒度。整数指定秒数。可以传递字符串，例如“minute”、“hour”和“day”。

响应

响应是正文中具有以下属性的 JSON 对象。

名称	类型	说明
timestamp	字符串	收集指标的时间（纪元或 ISO 8601）
results	对象	指标

指标是具有以下属性的 JSON 对象

名称	类型	说明
flow_count	整数	流数。
rx_byte_count	整数	接收的字节数。
rx_packet_count	整数	已接收的数据包数
tx_byte_count	整数	传输的字节数。
tx_packet_count	整数	传输的数据包数。

示例 Python 代码

```
agent_uuid = 'aa28b304f5c79b2f22d87a5af936f4a8fa555894'
td = 15 * 60 # 15 minutes
resp = restclient.get('/workload/%s/stats?t0=1483228800&t1=1485907200&td=%d' % (agent_uuid,
```

```
td))

 # This code queries workload statistics for a week
 t0 = 1483228800
 for _ in range(7):
   t1 = t0 + 24 * 60 * 60
   resp = restclient.get('/workload/%s/stats?t0=%d&t1=%d&td=day' % (agent_uuid, t0, t1))
   t0 = t1
```

安装的软件包

此终端会返回工作负载上安装的软件包的列表。

```
GET /openapi/v1/workload/{uuid}/packages
```

路径参数	说明
uuid	代理 UUID

响应

响应是包 JSON 对象的数组。包对象的架构如下所述：

属性	类型	说明
architecture	字符串	软件包的架构
name	字符串	软件包名称
publisher	字符串	软件包的发布服务器
version	字符串	软件包的版本

示例 Python 代码

```
agent_uuid = 'aa28b304f5c79b2f22d87a5af936f4a8fa555894'
resp = restclient.get('/workload/%s/packages' % (agent_uuid))
```

工作负载漏洞

此终端会返回在工作负载上观察到的漏洞列表。

```
GET /openapi/v1/workload/{uuid}/vulnerabilities
```

漏洞对象包含具有以下密钥的 JSON 正文。

路径参数	说明
uuid	代理 UUID

响应

响应是漏洞 JSON 对象的数组。漏洞对象的架构如下所述：

属性	类型	说明
cve_id	字符串	常见漏洞风险 ID
package_infos	数组	软件包信息对象数组
v2_score	浮点	CVSS V2 评分
v2_access_complexity	字符串	CVSS V2 访问复杂性
v2_access_vector	字符串	CVSS V2 访问向量
v2_authentication	字符串	CVSS V2 身份验证
v2_availability_impact	字符串	CVSS V2 可用性影响
v2_confidentiality_impact	字符串	CVSS V2 保密性影响
v2_integrity_impact	字符串	CVSS V2 完整性影响
v2_severity	字符串	CVSS V2 严重性
v3_score	浮点	CVSS V3 评分
v3_attack_complexity	字符串	CVSS V3 攻击复杂性
v3_attack_vector	字符串	CVSS V3 攻击向量
v3_availability_impact	字符串	CVSS V3 可用性影响
v3_base_severity	字符串	CVSS V3 基本严重性
v3_confidentiality_impact	字符串	CVSS V2 保密性影响
v3_integrity_impact	字符串	CVSS V3 完整性影响
v3_privileges_required	字符串	所需的 CVSS V3 权限
v3_scope	字符串	CVSS V3 范围
v3_user_interaction	字符串	CVSS V3 用户交互
cvm_score	浮点	思科安全风险评分
cvm_severity	字符串	思科安全风险评分严重性
cvm_easily_exploitable	布尔值	思科安全风险评分易被利用
cvm_malware_exploitable	布尔值	思科安全风险评分可被恶意软件利用

属性	类型	说明
cvm_active_internet_breach	布尔值	思科安全风险评分活动互联网漏洞
cvm_popular_target	布尔值	思科安全风险评分常见目标
cvm_predicted_exploitable	布尔值	思科安全风险评分预测可被利用
cvm_fix_available	布尔值	思科安全风险评分可用修复

示例 Python 代码

```
agent_uuid = 'aa28b304f5c79b2f22d87a5af936f4a8fa555894'
resp = restclient.get('/workload/%s/vulnerabilities' % (agent_uuid))
```

汇聚的工作负载漏洞摘要

此终端会返回在工作负载上观察到的漏洞的汇总统计信息。

```
POST /openapi/v1/workloads/cve/aggregated_stats
```

漏洞对象包含具有以下密钥的 JSON 正文：

路径参数	说明
uuid	主机 UUID

响应

响应是受影响工作负载和 CVE 的进程快照摘要：

属性	类型	说明
host_uuid	字符串	用于获取漏洞 CVE 数据的代理 ID 或工作负载 ID 列表
app_scope_id	字符串	使用 JSON 对象定义的所需 CVE 的范围
ips	字符串数组	要获取 CVE 信息的 IP 列表。
os	字符串	主机操作系统的列表
cve_infos	数组	CVE 信息数组

***Table 9: CVE* 信息结构**

属性	类型	说明
cve_id	字符串	包含 IP 地址的 CVE ID 列表

属性	类型	说明
v2_score	数字 float64	CVSS 评分
v2_severity	字符串	CVSS V2 严重性
v3_score	数字 float64	CVSS V3 评分
v3_base_severity	字符串	CVSS V3 基本严重性
cvm_score	数字 float64	思科安全风险评分
cvm_severity	字符串	思科安全风险评分严重性

示例 Python 代码

```
payload = {
"app_scope_id" : "66051883497d4f52437ba1b3",
"host_uuids" :
["39852c5221c4be28cd7c5e9786ac671c2faef13c","902c0977918ef3bfbd43b23782cc2574192f8bcb"]
}
resp = restclient.post('/workloads/cve/aggregated_stats', json_body=json.dumps(payload))
```

工作负载长时间运行的进程

此终端会返回工作负载上长期运行的进程的列表。长时间运行的进程定义为正常运行时间至少为 5 分钟的进程。

```
GET /openapi/v1/workload/{uuid}/process/list
```

路径参数	说明
uuid	代理 UUID

响应

响应是进程 JSON 对象的列表。

属性	类型	说明
cmd	字符串	进程的命令字符串
binary_hash	字符串	进程二进制的 Sha256 值（十六进制）
ctime	长度	进程二进制文件的 ctime（以 us 为单位）

属性	类型	说明
mtime	长度	进程二进制文件的 mtime（以 us 为单位）
exec_path	字符串	进程可执行文件路径
exit_usec	长度	进程退出时间（以微秒为单位）
num_libs	整数	进程加载的库数
pid	整数	进程 ID
ppid	整数	父进程 ID
pkg_info_name	字符串	与进程关联的软件包的名称
pkg_info_version	字符串	与进程关联的软件包的版本
proc_state	字符串	进程状态
uptime	长度	进程的正常运行时间（以 us 为单位）
username	字符串	进程的用户名
resource_usage	数组 对象	资源使用情况数组

示例 Python 代码

```
agent_uuid = 'aa28b304f5c79b2f22d87a5af936f4a8fa555894'
resp = restclient.get('/openapi/v1/workload/%s/process/list' % (agent_uuid))
```

工作负载进程快照摘要

此终端会返回此工作负载的进程快照摘要。进程快照包含工作负载在给定时间捕获的所有进程。当前保留一个最新进程快照的副本。终端支持使用空负载的 POST 方法，以便于将来进行扩展。

```
POST /openapi/v1/workload/{uuid}/process/tree/ids
```

路径参数	说明
uuid	代理 UUID

响应

响应是一个进程快照摘要 JSON 对象列表。

属性	类型	说明
sensor_uuid	字符串	代理 UUID
handle	字符串	要检索的进程快照的句柄
process_count	整数	快照中的进程数
ts_usec	整数	捕获快照时的时间戳

示例 Python 代码

```
agent_uuid = 'aa28b304f5c79b2f22d87a5af936f4a8fa555894'
payload = {
}
resp = restclient.post('/openapi/v1/workload/%s/process/tree/ids' %
                       agent_uuid, json_body=json.dumps(payload))
```

工作负载进程快照

此终端会返回此工作负载的进程快照。进程快照包含工作负载在给定时间捕获的所有进程。当前保留一个最新进程快照的副本。此终端需要与工作负载进程快照摘要终端一起使用。

```
POST /openapi/v1/workload/{uuid}/process/tree/details
```

路径参数	说明
uuid	代理 UUID

负载字段	类型	说明
handle	字符串	要检索的进程快照的句柄

响应

响应是一个属于 JSON 快照的进程列表。

属性	类型	说明
command_string	字符串	令牌化的命令字符串
command_string_raw	字符串	原始命令字符串
binary_hash	字符串	进程二进制的 Sha256 值（十六进制）
ctime	长度	进程二进制文件的 ctime（以 us 为单位）
mtime	长度	进程二进制文件的 mtime（以 us 为单位）

属性	类型	说明
exec_path	字符串	进程可执行文件路径
process_id	整数	进程 ID
parent_process_id	整数	父进程 ID
process_key	整数	进程的唯一密钥
parent_process_key	整数	父进程的唯一密钥
pkg_info_name	字符串	与进程关联的软件包的名称
pkg_info_version	字符串	与进程关联的软件包的版本
proc_state	字符串	进程状态
uptime	长度	进程的正常运行时间（以 us 为单位）
username	字符串	进程的用户名
cve_ids	数组	CVEID 对象数组

示例 Python 代码

```
agent_uuid = 'aa28b304f5c79b2f22d87a5af936f4a8fa555894'
payload = {
}
resp = restclient.post('/openapi/v1/workload/%s/process/tree/ids' %
            agent_uuid, json_body=json.dumps(payload))
handle = json.loads(resp.text)['process_summary'][0]['summary'][0]['handle']
payload = {
  "handle": handle,
}
resp = restclient.post('/openapi/v1/workload/%s/process/tree/details' %
            agent_uuid, json_body=json.dumps(payload))
```

JSON 对象定义

接口

属性	类型	说明
ip	字符串	接口的 IP 地址
mac	字符串	接口的 Mac 地址
name	字符串	接口的名称
netmask	字符串	接口的网络掩码

属性	类型	说明
pcap_opened	布尔值	如果为 false，则不会为接口启用数据包捕获
tags_scope_id	数组	与接口关联的范围 ID
vrf	字符串	VRF 名称
vrf_id	整数	VRF ID

软件包信息

属性	类型	说明
name	字符串	软件包名称
version	字符串	包版本

资源使用情况

属性	类型	说明
cpu_usage	浮点	CPU 使用率
memory_usage_kb	整数	内存使用率
ts_usec	长度	捕获资源使用情况时的时间戳（以微秒为单位）

CVE ID

属性	类型	说明
cve_id	字符串	cve ID
impact__cvss_v2__access_complexity	字符串	CVE 访问复杂性
impact__cvss_v2__access_vector	字符串	CVE 访问向量

默认策略生成配置

这组 API 用于读取和更新根范围的默认策略生成配置。

API 需要使用与 API 密钥关联的 `app_policy_management` 功能。

注释 这些 API 仅适用于站点管理员和根范围的所有者。

- 策略生成配置对象， 第 100 页
- 获取默认策略生成配置， 第 101 页
- 设置默认策略生成配置， 第 101 页

策略生成配置对象

属性	类型	说明
carry_over_policies	布尔值	如果可能，系统将保留标记为已批准的所有策略
deep_policy_generation	布尔值	为给定范围下的整个范围树创建策略，包括给定范围内的所有成员
skip_clustering	布尔值	设置为 true 时跳过集群，将生成包含现有已批准集群和范围的策略
auto_accept_policy_connectors	布尔值	自动接受所有传出策略连接器
enable_exclusion_filter	布尔值	将排除过滤器应用于输入流数据
enable_default_exclusion_filter	布尔值	将默认排除过滤器应用于输入流数据
remove_redundant_policies	布尔值	在深度策略生成期间删除冗余策略
enable_service_discovery	布尔值	设置为“false”时，会根据传感器报告的 adm 管道中的短暂端口范围跳过策略生成，目前用于为 Windows Active Directory 生成策略。
externals	数组	外部依赖关系对象的有序列表
clustering_granularity	字符串	VERY_COARSE、COARSE、MEDIUM、FINE、VERY_FINE 之一
policy_compression	字符串	DISABLED、CONSERVATIVE、MODERATE、AGGRESSIVE、VERY_AGGRESSIVE 之一
port_generalization	字符串	DISABLED、CONSERVATIVE、MODERATE、AGGRESSIVE、VERY_AGGRESSIVE 之一

属性	类型	说明
sim_policy	整数	1 => 流，2 => 进程，5 => 两者

外部依赖关系对象

名称	类型	说明
id	字符串	过滤器的 ID
filter_type	字符串	AppScope 或 UserInventoryFilter
incluce	数组	具有要启用的 user_filters 布尔值和要启用的 user_filter_list 的已提供服务资产过滤器的有序列表的对象

获取默认策略生成配置

此终端会返回当前默认策略生成配置。如果尚未创建任何对象，则可能返回空对象。

```
GET /openapi/v1/app_scopes/default_adm_run_config
```

参数：

请求 URL 包含以下参数

名称	类型	说明
root_app_scope_id	字符串	此默认配置所适用的根范围的唯一标识符

响应对象：返回当前默认策略生成配置，如果未创建，则返回空对象

设置默认策略生成配置

此终端会设置默认策略生成配置。

```
PUT /openapi/v1/app_scopes/default_adm_run_config
```

参数，以及上述策略生成配置对象列表中的值。

名称	类型	说明
root_app_scope_id	字符串	此默认配置所适用的根范围的唯一标识符

响应对象：返回默认策略生成配置。

取证意图

软件代理 API 与管理取证意图有关。

取证意图将取证配置文件与其适用的代理组联系起来。使用资产过滤器来定义代理组。

这些 API 集需要使用与 API 密钥关联的 `sensor_management` 功能。

注释 这些 API 仅适用于站点管理员和根范围的所有者。

取证意图对象

属性	类型	说明
id	字符串	意图的唯一标识符
name	字符串	意图的名称
inventory_filter_id	数组	与意图关联的资产过滤器的 ID
forensic_config_profile_id	整数	与此意图关联的配置文件的 ID
created_at	整数	意图创建时的 Unix 时间戳
updated_at	整数	上次更新意图的 Unix 时间戳

列出取证意图

此终端会列出所有现有的取证配置文件

```
GET /openapi/v1/inventory_config/forensic_intents
```

参数：无

此终端会返回取证意图对象摘要数组。

检索单个取证意图

```
GET /openapi/v1/inventory_config/forensic_intents/{intent_id}
```

参数：

名称	类型	说明
intent_id	字符串	意图的 ID

返回取证意图对象的详细表示形式。

创建取证意图

```
POST /openapi/v1/inventory_config/forensic_intents
```

参数：

名称	类型	说明
name	字符串	意图的名称
inventory_filter_id	字符串	与意图关联的资产过滤器的 ID
forensic_config_profile_id	字符串	与意图关联的取证配置文件的ID

返回取证意图对象。

更新取证意图

```
PUT /openapi/v1/inventory_config/forensic_intents/{intent_id}
```

参数：

名称	类型	说明
intent_id	字符串	意图的 ID
name	字符串	意图的名称
inventory_filter_id	字符串	与意图关联的资产过滤器的 ID
forensic_config_profile_id	字符串	与意图关联的取证配置文件的ID

返回取证意图对象。

删除取证意图

```
DELETE /openapi/v1/inventory_config/forensic_intents/{intent_id}
```

参数：

名称	类型	说明
intent_id	字符串	意图的 ID

成功时返回 200。

取证意图顺序

软件代理 API 与管理取证意图顺序相关联。

取证配置文件通过意图应用于代理。意图会使用资产过滤器来定义代理组。如果过滤器重叠，我们需要知道应用哪个过滤器。因此，我们使用排序功能来定义意图优先级。

这些 API 集需要使用与 API 密钥关联的 `sensor_management` 功能。

注释 这些 API 仅适用于站点管理员和根范围的所有者。

- 取证意图顺序对象， 第 104 页
- 检索当前取证意图顺序， 第 104 页
- 创建取证意图， 第 103 页

取证意图顺序对象

属性	类型	说明
version	字符串	当前排序的版本
intents	数组	按顺序排列意图对象的名称
intent_ids	数组	取证意图 ID 的数组

检索当前取证意图顺序

此终端会返回当前的取证意图顺序。

```
GET /openapi/v1/inventory_config/forensic_orders
```

参数：无

此终端会返回当前调查意图顺序对象。

创建取证意图顺序

```
POST /openapi/v1/inventory_config/forensic_orders
```

参数：

名称	类型	说明
version	字符串	必须与当前顺序匹配

名称	类型	说明
intent_ids	数组	意图 ID 数组

返回取证意图顺序对象。

取证配置文件

软件代理 API 与管理取证配置文件相关联。

取证配置文件是可应用于使用取证意图的代理组的规则集合。

这些 API 集需要使用与 API 密钥关联的 `sensor_management` 功能。

注释 这些 API 仅适用于站点管理员和根范围的所有者。

取证配置文件对象

属性	类型	说明
id	字符串	配置文件的唯一标识符
name	字符串	配置文件的名称
forensic_rules	数组	与配置文件关联的规则数组
created_at	整数	创建配置文件时的 Unix 时间戳
updated_at	整数	上次更新配置文件时的 Unix 时间戳
is_readonly	布尔值	指明配置文件是否为只读状态
root_app_scope_id	字符串	配置文件所属的根范围的 ID

列出取证配置文件

此终端会列出所有现有的取证配置文件

```
GET /openapi/v1/inventory_config/forensic_profiles
```

参数：无

此终端会返回取证配置文件对象摘要数组。

检索单个取证配置文件

```
GET /openapi/v1/inventory_config/forensic_profiles/{profile_id}
```

参数：

名称	类型	说明
profile_id	字符串	配置文件的 ID

返回取证配置文件对象的详细表示形式。

创建取证配置文件

```
POST /openapi/v1/inventory_config/forensic_profiles
```

参数：

名称	类型	说明
name	字符串	配置文件的名称
root_app_scope_id	字符串	配置文件所属的根范围的 ID
forensic_rule_ids	数组	要与此配置文件关联的取证规则 ID 数组

返回取证配置文件对象。

更新取证配置文件

```
PUT /openapi/v1/inventory_config/forensic_profiles/{id}
```

参数：

名称	类型	说明
profile_id	字符串	配置文件的 ID
name	字符串	配置文件的名称

名称	类型	说明
forensic_rule_ids	数组	要与此配置文件关联的取证规则 ID 数组

返回取证配置文件对象。

删除取证配置文件

```
DELETE /openapi/v1/inventory_config/forensic_profiles/{profile_id}
```

参数：

名称	类型	说明
profile_id	字符串	配置文件的 ID

成功时返回 200。

取证规则

软件代理 API 与管理取证规则相关联。

取证规则用于取证配置文件，然后将这些配置文件应用于代理组。

这些 API 集需要使用与 API 密钥关联的 `sensor_management` 功能。

注释　这些 API 仅适用于站点管理员和根范围的所有者。

取证规则对象

属性	类型	说明
id	字符串	规则的唯一标识符

属性	类型	说明
name	字符串	规则的名称
description	字符串	规则的说明
type	字符串	PREDEFINED 或 USER_DEFINED
eval_group_type	字符串	AS_INDIVIDUAL 或 AS_GROUP
severity	字符串	IMMEDIATE_ACTION、CRITICAL、HIGH、MEDIUM、LOW 之一
actions	数组	数组或 ALERT 或 REPORT 字符串
created_at	整数	规则创建时间的 Unix 时间戳
updated_at	整数	上次更新规则时的 Unix 时间戳

列出取证规则

此终端会列出所有现有的取证规则

```
GET /openapi/v1/inventory_config/forensic_rules
```

参数：无

此终端会返回取证规则对象摘要数组。

检索单个取证规则

```
GET /openapi/v1/inventory_config/forensic_rules/{rule_id}
```

参数：

名称	类型	说明
rule_id	字符串	规则 ID

返回取证规则对象的详细表示形式。

创建取证规则

```
POST /openapi/v1/inventory_config/forensic_rules
```

参数：

名称	类型	说明
root_app_scope_id	字符串	此规则所属的根范围的 ID
name	字符串	规则的名称
description	字符串	规则的说明
eval_group_type	字符串	规则的类型
severity	字符串	规则的严重性
actions	数组	数组或 ALERT 或 REPORT 字符串
clause	字符串	规则的查询子句。

返回取证规则对象。

更新取证规则

```
PUT /openapi/v1/inventory_config/forensic_rules/{rule_id}
```

参数：

名称	类型	说明
rule_id	字符串	规则 ID
name	字符串	规则的名称
description	字符串	规则的说明
eval_group_type	字符串	规则的类型
severity	字符串	规则的严重性
actions	数组	数组或 ALERT 或 REPORT 字符串
clause	字符串	规则的查询子句。

返回取证规则对象。

删除取证规则

```
DELETE /openapi/v1/inventory_config/forensic_rules/{rule_id}
```

参数：

名称	类型	说明
rule_id	字符串	规则 ID

成功时返回 200。

平台设置

这组 API 可用于添加、编辑或删除平台设置，并且需要使用与 API 密钥关联的 appliance_management 功能。

Note 这些 API 仅提供给客户支持和网站管理员。

获取证书

此终端用于检索 SSL/TLS 证书。

```
GET /openapi/v1/ platform_settings/ certs
```

响应是具有以下架构的 JSON 对象：

键	类型	值
key_pair_name	字符串	指定密钥对的名称。
cert_sha1	字符串	证书的唯一标识符。
created_at	字符串	检索在特定日期和时间当日或之后创建的证书。

获取使用情况分析设置

此终端用于检索平台中与使用分析或遥测数据收集相关的设置。

```
GET /openapi/v1/platform_settings/ usage_analytics
```

参数：

名称	类型	说明
查询	对象	检索与使用情况分析或遥测数据收集相关的设置。

响应是具有以下架构的 JSON 对象：

键	类型	值
usage_analytics_enabled	布尔值	指明是启用还是禁用使用情况分析或遥测数据收集。

获取登录消息

此终端用于检索在平台中向用户显示的自定义登录消息。

```
GET /openapi/v1/platfor_settings/ login_message
```

获取登录消息终端没有参数。

获取出站 HTTP 设置

此终端用于检索为平台配置的当前出站 HTTP 设置。

```
GET /openapi/v1/platform_settings/outbound_http
```

响应是具有以下架构的 JSON 对象：

键	类型	值
outbound_http_enabled	字符串	指明是否已启用出站 HTTP 连接。

更新出站 HTTP 设置

此终端用于检索为平台配置的当前出站 HTTP 设置。

```
POST /openapi/v1/ platform_settings/ outbound_http
```

参数：

名称	类型	说明
query	对象	更新为平台配置的出站 HTTP 设置。

响应是具有以下架构的 JSON 对象：

键	类型	值
`outbound_http_enabled`	字符串	指明是否已启用出站 HTTP 连接。

测试出站 HTTP 设置

此终端用于测试为平台配置的当前出站 HTTP 设置。

```
POST openapi/v1/platform_settings/outbound_http_test
```

响应是具有以下架构的 JSON 对象：

键	类型	值
success	布尔值	测试出站 HTTP 连接是否成功。

获取出站 HTTP 代理设置

此终端用于检索为平台配置的出站 HTTP 代理设置。

```
GET /openapi/v1/platform_settings/outbound_http_proxy
```

响应是具有以下架构的 JSON 对象：

键	类型	值
outbound_http_enabled	布尔值	指明是否已启用出站 HTTP 连接。
http_proxy_enabled	布尔值	指明是否已启用 HTTP 代理。
http_proxy_server	字符串	检索已配置的出站 HTTP 代理服务器。
http_proxy_port	整数	指定用于 HTTP 代理服务器的端口号。
http_proxy_username	字符串	指定用于向 HTTP 代理服务器进行身份验证的用户名。
http_proxy_password_present	布尔值	确定平台当前是否配置为使用密码对 HTTP 代理服务器进行身份验证。

更新出站 HTTP 代理设置

此终端用于检索为平台配置的出站 HTTP 代理设置。

```
POST /openapi/v1/ platform_settings/ outbound_http_proxy
```

响应是具有以下架构的 JSON 对象：

键	类型	值
outbound_http_enabled	布尔值	指明是否已启用出站 HTTP 连接。
http_proxy_enabled	布尔值	指明是否已启用 HTTP 代理。
http_proxy_server	字符串	检索已配置的出站 HTTP 代理服务器。
http_proxy_port	整数	指定用于 HTTP 代理服务器的端口号。
http_proxy_username	字符串	指定用于向 HTTP 代理服务器进行身份验证的用户名。
http_proxy_password_present	布尔值	确定密码是否可用。

执行

策略执行功能将生成的策略推送到与工作空间关联的范围内的资产，并编写新的防火墙规则。这组 API 需要使用与 API 密钥关联的 `app_policy_management` 功能。

有关详细信息，请参阅执行策略。

代理网络策略配置

此终端根据代理 ID 返回代理对象。它对于获取网络策略、代理配置及其版本等非常有用。

```
GET /openapi/v1/enforcement/agents/{aid}/network_policy_config
```

参数：

请求 URL 包含以下参数

名称	类型	说明
aid	字符串	网络策略配置的代理 UUID。

JSON 查询正文包含以下键

名称	类型	说明
include_filter_names	布尔值	在网络策略中包括过滤器名称和 ID。
inject_versions	布尔值	在网络策略中包括 ADM 工作空间版本。

响应

此终端的响应是 代理 对象。

具体策略统计信息

此终端会返回给定代理 ID 和具体策略 ID 的具体策略的统计信息。终端会返回 Timeseries 具体策略结果 对象的数组。

```
GET /openapi/v1/enforcement/agents/{aid}/concrete_policies/{cid}/stats?t0=<t0>&t1=<t1>
  →&td=<td>
```

参数：

请求 URL 包含以下参数

名称	类型	说明
aid	字符串	统计信息的代理 UUID。
cid	字符串	统计信息的具体策略 UUID。

JSON 查询正文包含以下键

Table 10:

名称	类型	说明
t0	整数	以纪元时间表示的统计信息开始时间
t1	整数	以纪元时间表示的统计信息结束时间
td	整数或 字符串	统计信息汇聚的粒度。整数指定秒数。可以传递字符串，例如“minute”、“hour”和“day”。

JSON 对象定义

代理

属性	类型	说明
agent_uuid	字符串	代理 UUID。
agent_config	对象	代理配置
agent_config_status	对象	代理配置状态

属性	类型	说明
desired_network_policy_config	对象	网络策略配置
provisioned_network_policy_config	对象	已调配的网络策略配置
provisioned_state_update_timestamp	整数	代理确认上述已调配策略时的纪元时间戳（以秒为单位）。
desired_policy_update_timestamp	整数	生成 required_network_policy_config 时的纪元时间戳（以秒为单位）。
agent_info	对象	代理信息
skipped	布尔值	如果跳过具体策略生成，则为 true。
message	字符串	跳过具体策略生成的原因。

代理配置

属性	类型	说明
agent_uuid	字符串	代理 UUID。
enforcement_enabled	布尔值	配置说明已在代理上启用执行。
fail_mode	字符串	故障模式。
version	数字	代理配置版本号。
control_tet_rules_only	布尔值	仅控制 Tet 规则配置。
allow_broadcast	布尔值	允许广播配置。
allow_multicast	布尔值	允许组播配置。
allow_link_local	布尔值	允许链接本地配置。
enforcement_cpu_quota_mode	字符串	执行代理 CPU 配额模式。
enforcement_cpu_quota_us	字符串	执行代理 CPU 配额微秒
enforcement_max_rss_limit	数字	执行代理最大 RSS 限制。

网络策略配置

属性	类型	说明
version	字符串	版本号
network_policy	数组	网络策略对象数组。
address_sets	数组	IP 集功能的地址集对象数组。
container_network_policy	数组	ContainerNetworkPolicy 对象数组。

网络策略

属性	类型	说明
priority	字符串	具体策略的优先级。
enforcement_intent_id	字符串	执行意图 ID。
concrete_policy_id	字符串	具体策略 ID。
match	对象	策略的匹配标准。不建议使用此字段。
action	对象	策略匹配的操作。
workspace_id	字符串	工作空间的 ID。
adm_data_set_id	字符串	工作空间的自动策略发现数据集 ID。
adm_data_set_version	字符串	工作空间的自动策略发现数据集版本。仅当在参数中传递 inject_versions=true 时设置。
cluster_edge_id	字符串	集群边缘 ID。
policy_intent_group_id	字符串	策略意图组 ID。
match_set	对象	用于 IP 集支持的匹配集对象。仅显示 match 或 match_set 之一。
src_filter_id	字符串	源资产过滤器 ID。这将在 include_filter_names=true 作为参数传递时进行设置。

属性	类型	说明
src_filter_name	字符串	源资产过滤器名称。这将在 include_filter_names=true 作为参数传递时进行设置。
dst_filter_id	字符串	目标资产过滤器 ID。这将在 include_filter_names=true 作为参数传递时进行设置。
dst_filter_name	字符串	目标资产过滤器名称。这将在 include_filter_names=true 作为参数传递时进行设置。

ContainerNetworkPolicy

属性	类型	说明
pod_id	字符串	POD ID。
network_policy	数组	网络策略对象数组。
deployment	字符串	部署名称。
service_endpoint	数组	服务终端名称列表。

匹配

属性	类型	说明
src_addr	对象	源地址的子网对象。
dst_addr	对象	目标地址的子网对象。
src_port_range_start	整数	源端口范围开始。
src_port_range_end	整数	源端口范围结束。
dst_port_range_start	整数	目标端口范围开始。
dst_port_range_end	整数	目标端口范围结束。
ip_protocol	字符串	IP 协议。
address_family	字符串	IPv4 或 IPv6 地址系列。
direction	字符串	匹配方向，INGRESS 或 EGRESS。
src_addr_range	对象	源地址的地址范围对象。

属性	类型	说明
dst_add_range	对象	目标地址的地址范围对象。

操作

属性	类型	说明
type	字符串	操作类型。

匹配集

属性	类型	说明
src_set_id	字符串	网络策略配置 address_sets 中的地址集 对象的源集 ID 数组。
dst_set_id	字符串	网络策略配置 address_sets 数组中的地址集 对象的目标集 ID
src_ports	ar- ray	源端口的端口范围对象数组。
dst_ports	ar- ray	目标端口的端口范围对象数组。
ip_protocol	字符串	IP 协议。
address_family	字符串	IPv4 或 IPv6 地址系列。
direction	字符串	匹配方向，INGRESS 或 EGRESS。

地址集

属性	类型	说明
set_id	字符串	地址集 ID。
addr_ranges	数组	地址范围对象数组。
subnets	数组	子网对象数组。
addr_family	字符串	IPv4 或 IPv6 地址系列。

子网

属性	类型	说明
ip_addr	字符串	IP 地址。
prefix_length	整数	子网的前缀长度。

地址范围

属性	类型	说明
start_ip_addr	字符串	范围的起始 IP 地址。
end_ip_addr	字符串	范围的结束 IP 地址。

端口范围

属性	类型	说明
start_port	整数	范围的起始端口。
end_port	整数	范围的结束端口。

代理配置状态

属性	类型	说明
disabled	布尔值	配置说明已在代理上禁用执行。
current_version	数字	代理上应用的当前代理配置版本。
highest_seen_version	数字	代理接收的代理配置的最高版本。

已调配的网络策略配置

属性	类型	说明
version	字符串	代理调配的网络策略配置版本。
error_reason	字符串	如果代理成功应用了策略，则为 CONFIG_SUCCESS，否则为错误原因。
disabled	布尔值	配置说明已在代理上禁用执行。

属性	类型	说明
current_version	数字	代理上应用的当前 NPC 版本。
highest_seen_version	数字	代理接收的最高版本 NPC。
policy_status	对象	每个网络策略状态。

代理信息

属性	类型	说明
agent_info_supported	布尔值	代理功能（如果支持 agent_info）。
ipset_supported	布尔值	代理功能（如果支持 IP 集）。

具体策略结果

属性	类型	说明
byte_count	整数	具体策略命中的字节数。
pkt_count	整数	具体策略命中的数据包计数。

Timeseries 具体策略结果

属性	类型	说明
timestamp	字符串	用于汇聚结果的时间戳字符串。
result	对象	具体策略结果

客户端服务器配置

检测客户端和服务器关系是 Cisco Secure Workload 中各种功能的核心，因此我们建议尽可能使用软件代理，因为它可以报告基本事实。网络中的任何遥测监测点都无法保证观测到给定流量的每个数据包，这是由于各种情况造成的，例如：TCP 流量的两个单向半流可能在网络中采取独特的路径，因此总是不可避免地受到一定程度的误差影响。

Cisco Secure Workload 尝试在不进行任何用户交互的情况下检测并尽可能减少这些错误，方法是对每个流应用机器学习算法，构建统计模型，以便在报告不一致的遥测数据时做出判断。在大多数情况下，用户无需担心这组 API。但在少数情况下，客户端服务器检测算法可能无法正确获取流方向。依赖流量方向的功能（如自动策略发现）可能会出现不希望出现的行为，如打开不必要的端口。

提供了一组 API，可用于向 Cisco Secure Workload 算法提供有关已知服务器端口的提示。这组 API 可供具有根范围所有权角色的用户使用，并且需要使用与这些用户的 API 密钥关联的 `app_policy_management` 功能。

客户端服务器配置有 2 个选项：

主机配置

配置适用于根范围内特定 IP 地址子集的已知服务器端口

添加服务器端口配置

此 API 可用于为 Cisco Secure Workload 算法提供有关给定根范围的已知服务器端口的提示。您可以为属于根范围的一组 IP 地址提供已知 TCP/UDP 服务器端口列表，以便帮助 Cisco Secure Workload 算法确定流中正确的客户端服务器方向。

```
POST /openapi/v1/adm/{root_scope_id}/server_ports
```

参数：请求 URL 包含以下参数

名称	类型	说明
root_scope_id	字符串	根范围的唯一标识符。

此外，作为该 API 输入的文本文件包含以下格式的终端服务器端口配置：

终端服务器端口配置

属性	类型	说明
ip_address	字符串	IP 地址（可以是 ipv4 或 ipv6 地址）。不允许使用子网。
tcp_server_ports	意图列表	与 ip_address 对应的已知 TCP 服务器端口列表。
udp_server_ports	意图列表	与 ip_address 对应的已知 UDP 服务器端口列表。

批量服务器端口配置

属性	类型	说明
host_config	终端服务器端口配置对象列表。	关联已知服务器端口的 IP 地址列表。

示例 python 代码

```
# contents of below file:
```

```
# {"host_config": [
#    {"ip_address": "1.1.1.1",
#     "tcp_server_ports": [100, 101, 102],
#     "udp_server_ports": [103]
#    },
#    {"ip_address": "1.1.1.2",
#     "tcp_server_ports": [200, 201, 202]
#    }
#  ]
# }

file_path = '/<path_to_file>/server_ports.txt'
root_scope_id = '<root-scope-id>'
restclient.upload(file_path,
                  '/adm/%s/server_ports' % root_scope_id,
                  timeout=200) # seconds
```

Note 上述 API 会覆盖后台已知服务器端口配置的全部状态。如果需要修改任何内容，他们需要在修改后重新上传完整配置。

获取服务器端口配置

此 API 会返回根范围内终端的已知上传服务器端口列表。

```
GET /openapi/v1/adm/{root_scope_id}/server_ports
```

参数：请求 URL 包含以下参数

名称	类型	说明
root_scope_id	字符串	根范围的唯一标识符。

响应对象：ref:*ServerPortConfig* 对象的列表。

示例 python 代码

```
root_scope_id = '<root-scope-id>'
restclient.get('/adm/%s/server_ports' % root_scope_id)
```

删除服务器端口配置

此 API 可删除指定根范围的服务器端口配置。

```
DELETE /openapi/v1/adm/{root_scope_id}/server_ports
```

参数：请求 URL 包含以下参数

名称	类型	说明
root_scope_id	字符串	根范围的唯一标识符。

响应对象：无。

示例 python 代码

```
root_scope_id = '<root-scope-id>'
restclient.delete('/adm/%s/server_ports' % root_scope_id)
```

端口配置

配置适用于属于根范围的所有 IP 地址的已知服务器端口

推送服务器端口配置

此 API 可用于为 Cisco Secure Workload 算法提供有关给定根范围的已知服务器端口的提示。用户可以提供给定根范围的已知 TCP/UDP 服务器端口列表，以帮助 Cisco Secure Workload 算法确定流中正确的客户端服务器方向。用户还可以选择指定与每个服务器端口相关的服务名称。

此外，还有一个适用于所有根范围的已知服务默认列表（以下称为全局服务）。用户可以随时覆盖此列表。

服务配置

服务的定义是一对（端口、名称）。

属性	类型	说明
port	整数	TCP/UDP 服务器端口号
name	字符串	与此端口关联的服务名称（可选）
override_in_conflicts	布尔值	发生冲突时强制主机成为提供者（可选）

批量服务配置

属性	子属性	类型	说明
server_ports_config	tcp_service_list	服务配置对象列表。	已知 TCP 服务列表
	udp_service_list	服务配置对象列表。	已知 UDP 服务列表

按根范围推送服务：

```
POST /openapi/v1/adm/{root_scope_id}/server_ports_config
```

示例 python 代码

```
# contents of below file:
#{"server_ports_config":
#    {
#        "tcp_service_list": [
#                {
#                        "port": 80,
#                        "name" : "http"
#
```

```
#                 },
#                 {
#                         "port": 53,
#                         "name" : "dns"
#                 },
#                 {
#                         "port": 514,
#                         "name" : "syslog",
#                         "override_in_conflicts": true
#                 }
#         ],
#         "udp_service_list": [
#                 {
#                         "port": 161
#                 },
#                 {
#                         "port": 53,
#                         "name": "dns"
#                 }
#         ]
#     }
#}

file_path = '/<path_to_file>/server_ports.json'

# Updating service list for a given root scope
#restclient.upload(file_path,
#                  '/openapi/v1/adm/{root_scope_id}/server_ports_config',
#                  timeout=200) # seconds
```

Note 上述 API 会覆盖后台已知服务器端口配置的全部状态。如果用户需要修改任何内容，则需要在修改后重新上传完整配置。

检索服务器端口配置

此 API 会返回用户上传的根范围内的已知服务器端口列表。响应为批量服务配置。

```
Retrieve configured services per root scope:
    GET /openapi/v1/adm/{root_scope_id}/server_ports_config

Retrieve configured global services:
    GET /openapi/v1/adm/server_ports_config
```

删除服务器端口配置

此 API 可删除指定根范围的服务器端口配置。

```
Remove configured services per root scope:
    DELETE /openapi/v1/adm/{root_scope_id}/server_ports_config
```

软件代理

代理 API

软件代理 API 与管理 Cisco Secure Workload 软件代理相关联。这些 API 集需要使用与 API 密钥关联的 `sensor_management` 功能。以下 *GET* API 也适用于与 API 密钥关联的 `flow_Inventory_query` 功能。

获取软件代理

此终端会返回软件代理列表。每个软件代理都有两个字段来描述其代理类型，其中 `agent_type_str` 是纯文本格式，而 `agent_type` 是枚举。

```
GET /openapi/v1/sensors
```

参数：

名称	类型	说明
limit	整数	限制返回的结果数量（可选）
offset	字符串	偏移量被用于分页请求。如果响应返回偏移量，则后续请求必须使用相同的偏移量方可在下一页中获得更多结果。（可选）

获取特定软件代理

此终端会返回 UUID 是 URI 一部分的软件代理的属性。每个软件代理都有两个字段来描述其代理类型，其中 agent_type_str 是纯文本格式，而 agent_type 是枚举。

```
GET /openapi/v1/sensors/{uuid}
```

删除软件代理

此终端用于根据软件代理的 UUID 来下线软件代理。

请谨慎使用 API；删除代理后，Cisco Secure Workload 控制面板中不再提供该代理，如果代理处于活动状态，则在 Cisco Secure Workload 中不允许从代理导出流。

```
DELETE /openapi/v1/sensors/{uuid}
```

使用意图的软件代理配置

此 API 工作流程使用下面定义的几个 REST 终端。

创建资产过滤器

此终端用于指定与用户要配置软件代理的代理主机相匹配的标准。

```
POST /openapi/v1/filters/inventories
```

参数：

名称	类型	说明
app_scope_id	字符串	要分配给资产过滤器的范围ID。
name	字符串	资产过滤器的名称。
query	json	代理主机的过滤或匹配条件。

示例 python 代码

```
# app_scope_id can be retrieved by /app_scopes API
req_payload = {
   "app_scope_id": <app_scope_id>,
   "name": "sensor_config_inventory_filter",
   "query": {
      "type": "eq",
      "field": "ip",
      "value": <sensor_interface_ip>
   }
}
resp = restclient.post('/filters/inventories',
                json_body=json.dumps(req_payload)))
print resp.status_code
# returned response will contain the created filter and it's ID.
```

创建软件代理配置文件

此终端用于指定要应用于目标软件代理集的配置选项集。

```
POST /openapi/v1/inventory_config/profiles
```

以下配置选项可作为代理配置文件的一部分进行指定：

- allow_broadcast：此选项可允许/禁止广播流量（此选项的默认值为 True）。
- allow_multicast：此选项可允许/禁止组播流量（此选项的默认值为 True）。
- allow_link_local：此选项可允许/禁止链接本地流量（此选项的默认值为 True）。
- auto_upgrade_opt_out：如果为 true，则在升级 Cisco Secure Workload 集群期间不会自动升级代理。
- cpu_quota_mode 和 cpu_quota_us：这些选项用于控制分配给终端主机上的代理的 CPU 配额量。
- data_plane_disabled：如果为 true，则代理会停止向 Cisco Secure Workload 报告流。
- enable_conversation_flows：此选项可在所有代理上启用对话模式。
- enable_forenstics：此选项可启用对工作负载的取证事件收集（代理会因此使用更多 CPU）。
- enable_Meltdown：对工作负载启用崩溃漏洞攻击检测（代理会因此使用更多 CPU）。

- enable_pid_lookup：如果为 true，代理会尝试将进程信息附加到流。请注意，此配置选项会在终端主机上使用更多 CPU。
- enforce_disabled：可用于在运行强制代理的主机上禁用强制。
- preserve_existing_rules：此选项可指定是否保留现有 iptable 规则。
- windows_enforcement_mode：此选项可使用 WAF（Windows 高级防火墙）或 WFP（Windows 过滤平台）（默认选项为 WAF）。

有关配置选项的更多详细信息，请参阅软件代理配置

示例 python 代码

```
# Define profile to disable data_plane on agent
req_payload = {
   "root_app_scope_id": <root_app_scope_id>,
   "data_plane_disabled": True,
   "name": "sensor_config_profile_1",
   "enable_pid_lookup": True,
   "enforcement_disabled": False
}
resp = restclient.post('/inventory_config/profiles',
                json_body=json.dumps(req_payload))
print resp.status_code
# returned response will contain the created profile and it's ID.
parsed_resp = json.loads(resp.content)
```

获取软件代理配置文件

此终端会返回用户可见的软件代理配置文件列表。

```
GET /openapi/v1/inventory_config/profiles
```

参数：无

获取特定软件代理配置文件

此终端会返回软件代理配置文件的实例。

```
GET /openapi/v1/inventory_config/profiles/{profile_id}
```

返回与指定 ID 关联的软件代理配置文件对象。

更新软件代理配置文件

此终端将更新软件代理配置文件。

```
PUT /openapi/v1/inventory_config/profiles/{profile_id}
```

以下配置选项可作为代理配置文件的一部分进行指定：

- allow_broadcast：此选项可允许/禁止广播流量（此选项的默认值为 True）。
- allow_multicast：此选项可允许/禁止组播流量（此选项的默认值为 True）。
- allow_link_local：此选项可允许/禁止链接本地流量（此选项的默认值为 True）。

- auto_upgrade_opt_out：如果为 true，则在升级 Cisco Secure Workload 集群期间不会自动升级代理。
- cpu_quota_mode 和 cpu_quota_us：这些选项用于控制分配给终端主机上的代理的 CPU 配额量。
- data_plane_disabled：如果为 true，则代理会停止向 Cisco Secure Workload 报告流。
- enable_conversation_flows：此选项可在所有代理上启用对话模式。
- enable_forenstics：此选项可启用对工作负载的取证事件收集（代理会因此使用更多 CPU）。
- enable_Meltdown：对工作负载启用崩溃漏洞攻击检测（代理会因此使用更多 CPU）。
- enable_pid_lookup：如果为 true，代理会尝试将进程信息附加到流。请注意，此配置选项会在终端主机上使用更多 CPU。
- enforce_disabled：可用于在运行强制代理的主机上禁用强制。
- preserve_existing_rules：此选项可指定是否保留现有 iptable 规则。
- windows_enforcement_mode：此选项可使用 WAF（Windows 高级防火墙）或 WFP（Windows 过滤平台）（默认选项为 WAF）。

有关配置选项的更多详细信息，请参阅软件代理配置

返回与指定 ID 关联的已修改软件代理配置文件对象。

删除软件代理配置文件

此终端将删除指定的软件代理配置文件。

```
DELETE /openapi/v1/inventory_config/profiles/{profile_id}
```

创建软件代理配置意图

此终端用于指定意图将一组配置选项应用于指定的软件代理集。这样将创建意图，并通过将新创建的意图添加到顺序中来更新意图顺序。

```
POST /openapi/v1/inventory_config/intents
```

示例 python 代码

```
req_payload = {
  "inventory_config_profile_id": <>,
  "inventory_filter_id": <>
}
resp = restclient.post('/inventory_config/intents',
                json_body=json.dumps(req_payload))
print resp.status_code
# returned response will contain the created intent object and it's ID.
```

指定意图的顺序

此终端用于指定各种软件代理配置意图的顺序。例如，可能有两个意图 - 第一个意图是在开发计算机上启用进程 ID 查找，第二个意图是在 Windows 计算机上禁用进程 ID 查找。如果第一个意图具有

更高的优先级，则开发 Windows 计算机将启用进程 ID 查找。注意：创建意图时，会将其默认添加到意图顺序列表的开头。此终端仅在最终用户需要修改现有意图顺序时使用。

```
POST /openapi/v1/inventory_config/orders
```

示例 python 代码

```
# Read the agent config intents ordered list
resp = restclient.get('/inventory_config/orders')
order_result_json = json.loads(resp.content)

# Modify the list by prepending the new intent in the list
order_rslt_json['intent_ids'].insert(0,<intent_id>)

# Post the new ordering back to the server
resp = restclient.post('/inventory_config/orders',
               json_body=json.dumps(order_rslt_json))
```

删除代理配置意图

此终端用于删除特定代理配置意图。

```
DELETE /openapi/v1/inventory_config/intents/{intent_id}
```

示例 python 代码

```
intent_id = '588a51dcb5b30d0ee6da084a'
resp = restclient.delete('/inventory_config/intents/%s' % intent_id)
```

接口配置意图

将 VRF 分配给代理的推荐方法是使用远程 VRF 配置设置。在极少数情况下，当代理主机可能有多个接口需要分配不同的 VRF 时，用户可以选择使用接口配置意图来为它们分配 VRF。转到**管理 (Manage)** > **代理 (Agents)**，然后点击**配置 (Configure)** 选项卡。

资产配置意图对象

GET 和 POST 方法会返回资产配置意图 JSON 对象数组。对象的属性如下所述：

属性	类型	说明
vrf_id	整数	VRF ID 整数
vrf_name	字符串	VRF 名称
inventory_filter_id	字符串	资产过滤器 ID
inventory_filter	JSON	资产过滤器。有关详细信息，请参阅“OpenAPI > 资产过滤器”。

获取接口配置意图

此终端会向用户返回资产配置意图列表。

```
GET /openapi/v1/inventory_config/interface_intents
```

参数：无

创建或更新接口配置意图列表

此终端用于创建或修改接口配置意图列表。API 会采用有序的意图列表。要删除此列表中的意图，用户需要读取现有的意图列表，修改列表并将修改后的列表写回。

```
POST /openapi/v1/inventory_config/interface_intents
```

参数：

名称	类型	说明
inventory_filter_id	字符串	要匹配接口的资产过滤器 ID
vrf_id	整数	要分配接口的 VRF ID

示例 python 代码

```
  req_payload = {
    "intents": [
      {"inventory_filter_id": <inventory_filter_id_1>, "vrf_id": <vrf_id_1>},
      {"inventory_filter_id": <inventory_filter_id_1>, "vrf_id": <vrf_id_2>}
    ]
  }
  resp = restclient.post('/inventory_config/interface_intents',
json_body=json.dumps(req_payload))
```

NAT 后代理的 VRF 配置

以下 API 集可用于指定为 NAT 设备后面的代理分配 VRF 的策略。这些 API 集需要使用与 API 密钥关联的 `sensor_management` 功能，并且仅对站点管理员用户可用。

列出 NAT 后代理的 VRF 配置规则

此终端会返回适用于 NAT 后面的代理的 VRF 配置规则列表。

```
GET /openapi/v1/agentnatconfig
```

创建适用于 NAT 后代理的新 VRF 配置

此终端用于根据 Cisco Secure Workload 设备看到的主机的源 IP 和源端口来为主机指定 VRF 标记条件。

```
POST /openapi/v1/agentnatconfig
```

参数：

名称	类型	说明
src_subnet	字符串	源 IP 可以属于的子网（CIDR 表示法）。
src_port_range_start	整数	源端口范围的下限 (0-65535)。
src_port_range_end	整数	源端口范围的上限 (0-65535)。
vrf_id	整数	用于标记源地址和端口在上述指定范围内的代理流的 VRF ID。

示例 python 代码

```
req_payload = {
    src_subnet: 10.1.1.0/24,          # src IP range for sensors
    src_port_range_start: 0,
    src_port_range_end: 65535,
    vrf_id: 676767                    # VRF ID to assign
}

resp = rc.post('/agentnatconfig', json_body=json.dumps(req_payload))
print resp.status_code
```

删除现有 VRF 配置

```
DELETE /openapi/v1/agentnatconfig/{nat_config_id}
```

Cisco Secure Workload 软件下载

Cisco Secure Workload 软件下载功能提供了一种为 Cisco Secure Workload 代理下载软件包的方法。这些API集需要使用与API密钥关联的`software_download`功能。此功能仅适用于站点管理员用户、根范围所有者和具有代理安装程序角色的用户。

用于获取受支持平台的 API

此终端会返回受支持平台的列表。

```
GET /openapi/v1/sw_assets/platforms
```

参数：无

响应对象：返回受支持平台的列表。

示例 python 代码

以下示例代码检索所有受支持的平台。

```
resp = restclient.get('/sw_assets/platforms')
if resp.status_code == 200:
   parsed_resp = json.loads(resp.content)
   print json.dumps(parsed_resp)
```

响应示例

```
{"results": [{"platform": "OracleServer-6.3", "agent_type": "enforcer", "arch": "x86_64"},
 {"platform": "MSWindows8Enterprise", "agent_type": "legacy_sensor", "arch": "x86_64"}]}
```

用于获取支持的软件版本的 API

此终端会返回指定“agent_type”、“package_type”、“platform”和“architecture”的受支持软件版本列表。

```
GET
/openapi/v1/sw_assets/download?platform=<platform>&agent_type=<agent_type>&pkg_type=<pkg_type>&arch=<arch>&list_version=<list_version>&installation_id=<installer_id>
```

其中 <agent_type>、<platform> 和 <arch> 可以是从 **API 检索的任何结果，以获取支持的平台**，并且 <pkg_type> 可以是“sensor_w_cfg”或“sensor_bin_pkg”。<pkg_type> 和 <agent_type> 均为可选，但至少应指定其中一个。<list_version> 必须为“True”才能启用此 API。

参数：请求 URL 包含以下参数。

名称	类型	说明
platform	字符串	指定平台。
agent_type	字符串	（可选）指定代理类型。
pkg_type	字符串	（可选）指定软件包类型，值可以是“sensor_w_cfg”或“sensor_bin_pkg”。
arch	字符串	指定架构。
list_version	字符串	设置为“True”可启用软件版本搜索。

响应对象：返回支持的软件版本列表。

示例 python 代码

```
resp =
restclient.get('/sw_assets/download?platform=OracleServer-6.3&pkg_type=sensor_w_cfg&arch=x86_64&list_version=True')
if resp.status_code == 200:
   print resp.content
```

```
resp =
[illegible]
if resp.status_code == 200:
   print resp.content
```

响应示例

```
3.3.1.30.devel
3.3.1.31.devel
```

用于创建安装程序 ID 的 API

此终端为 API 创建安装程序 ID，以下载 Cisco Secure Workload 软件。

```
GET /openapi/v1/sw_assets/installation_id
```

响应对象：返回可在 API 中用于下载 Cisco Secure Workload 软件的安装程序 ID。

示例 python 代码

```
resp = restclient.get('/sw_assets/installation_id')
if resp.status_code == 200:
   parsed_resp = json.loads(resp.content)
   print json.dumps(parsed_resp)
```

响应示例

"[redacted]"

用于下载 Cisco Secure Workload 软件的 API

此终端让客户端能够下载指定“agent_type”、“package_type”、“platform”、“architeure”和“sensor_version”的软件。

```
GET
/openapi/v1/sw_assets/download?platform=<platform>&agent_type=<agent_type>&pkg_type=<pkg_type>&arch=<arch>&sensor_version=<sensor_version>&installation_id=<installer_id>
```

其中 <agent_type>、<platform> 和 <arch> 可以是从 **API 检索的任何结果，以获取支持的平台**，并且 <pkg_type> 可以是“sensor_w_cfg”或“sensor_bin_pkg”。<pkg_type> 和 <agent_type> 均为可选，但至少应指定其中一个。<sensor_version> 可以是从 **API 检索的任何结果，以获取支持的软件版本**。如果未指定“sensor_version”，它将下载**最新**的软件。

参数：请求 URL 包含以下参数。

名称	类型	说明
platform	字符串	指定平台。
agent_type	字符串	（可选）指定代理类型。
pkg_type	字符串	（可选）指定软件包类型，值可以是“sensor_w_cfg”或“sensor_bin_pkg”。
arch	字符串	指定架构。
sensor_version	字符串	（可选）指定软件版本，默认为空字符串。

响应对象：返回给定参数的 Cisco Secure Workload 软件。

示例 python 代码

```
resp =
[redacted]
if resp.status_code == 200:
   print 'file downloaded successfully'
```

Cisco Secure Workload 代理升级

Cisco Secure Workload 代理升级功能提供将已安装的 Cisco Secure Workload 代理升级到特定版本的方法。它只会更新元数据，实际升级将在下次签入期间进行。API 需要使用与 API 密钥关联的 `software_download` 功能。只有站点管理员用户、根范围所有者或具有代理安装程序角色的用户才能使用此功能。

用于将代理升级到特定版本的 API

如果此终端的“UUID”升级到特定的“sensor_version”，则会触发代理，如果未提供“sensor_version”，则会应用最新版本。此 API 不会处理降级请求。

```
POST /openapi/v1/sensors/{UUID}/upgrade?sensor_version=<sensor_version>
```

其中 <sensor_version> 可以是从 用于获取支持的软件版本的 API 。

参数：请求 URL 包含以下参数

名称	类型	说明
sensor_version	字符串	（可选）指定所需版本，默认情况下将应用最新版本

返回此升级请求的状态。

示例 python 代码

```
resp = restclient.post('/openapi/v1/sensors/{UUID}/upgrade?sensor_version=3.4.1.1.devel')

if resp.status_code == 200:
  print 'agent upgrade was triggered successfully and in progress'
elif resp.status_code == 304:
  print 'provided version is not newer than current version'
elif resp.status_code == 400:
  print 'provided version is invalid'
elif resp.status_code == 403:
  print 'user does not have required capability'
elif resp.status_code == 404:
  print 'agent with {UUID} does not exist'
```

收集规则

这些 API 集可用于管理收集规则。Cisco Secure Workload 设备中的收集规则可供用户指定对其部署感兴趣的 IP 地址或子网。在收到这些收集规则后，代理只会提取符合这些收集规则的 IP 地址的流量信号。这些 API 需要使用与 API 密钥关联的 `flow_Inventory_query` 功能。

Note 这些 API 仅适用于站点管理员用户。

收集规则对象

收集规则对象属性如下所述：

属性	类型	说明
subnet	字符串	CIDR 格式的子网或 IP 地址。
action	字符串	可能的值为 ‘INCLUDE’ 或 ‘EXCLUDE’。

VRF 的更新收集规则

此终端可用于更新指定 VRF 的有序收集规则列表。请注意，POST 请求中的收集规则列表会被视为有序列表。

```
POST /openapi/v1/collection_rules/{vrf_name}
```

参数：

POST 正文中收集规则对象的有序列表。**最后两条规则必须为 IPv4 和 IPv6 的全部捕获规则**。规则可以分别指定子网 0.0.0.0/0 和 ::/0，类似于下面的示例。

响应对象：已更新 VRF 收集规则的有序列表。

示例 python 代码

```
req_payload = [
    {
       "subnet": "10.10.10.0/24",
       "action": "INCLUDE"
    },
    {
       "subnet": "11.11.11.0/24",
       "action": "INCLUDE"
    },
    {
       "subnet": "0.0.0.0/0",   # catch all rule for IPV4 addresses
       "action": "EXCLUDE"
    },
    {
       "subnet": "::/0",        # catch all rule for IPV6 addresses
       "action": "EXCLUDE"
    }
]
resp = restclient.post('/collection_rules/test_vrf', json_body=json.dumps(req_payload))
```

获取 VRF 的收集规则

此终端会返回指定 VRF 的收集规则的有序列表。

```
GET /openapi/v1/collection_rules/{vrf_name}
```

参数：无

响应对象：指定 VRF 的收集规则的有序列表。

示例 python 代码

```
resp = restclient.get('/collection_rules/test_vrf')
```

收集规则的影响

有两种资产项目：

- 传感器获知（工作负载配置文件）：包括属于运行 Cisco Secure Workload 个传感器的工作负载的所有 IP 地址
- 获知的流（资产配置文件）：包括在由 Cisco Secure Workload 收集的流信号中发现但未与运行 Cisco Secure Workload 代理的任何工作负载关联的所有 IP 地址。

EXCLUDE/INCLUDE 收集规则可控制跟踪哪些资产项目。无论收集规则如何，都始终跟踪传感器获知的资产项目。对于流获知的资产项目，如果收集规则将其排除，则资产项目将不存在。因此，资产搜索将不会返回此类资产的任何结果。

流搜索不受收集规则的影响，但标签列除外，因为被收集规则排除在外的 IP 将不会填入标签列。采集规则与确定任何给定流的客户端-服务器无关。

自动策略发现结果可能会受到影响，因为我们不会跟踪被收集规则排除在外的 IP 的标签。

用户上传的文件散列

用户可以将文件散列列表上传到 Cisco Secure Workload ，并指定这些散列是良性的还是已标记的。Cisco Secure Workload 相应地使用相应的二进制散列标记进程。

这组 API 可用于将文件散列列表上传或删除到 Cisco Secure Workload。要调用这些 API，请使用具有 `user_data_upload` 功能的 API 密钥。

Note 每个根范围最多可以有 100 万个文件散列值。500000（良性散列和标记散列）。

范围所有者和站点管理员可使用以下 API 在 |product| 设备上的单个根范围中上传/下载/删除文件散列。

用户文件散列上传

此终端用于为 Cisco Secure Workload 设备上的根范围上传包含 filehash 的 CSV 文件。CSV 文件中必须出现列标题 `HashType` 和 `FileHash`。`HashType` 应为 `SHA-1` 或 `SHA-256`，FileHash 不能为空且格式必须为 40 十六进制 SHA1 或 64 十六进制 SHA256。

`FileName` 和 `Notes` 标头为可选。给定文件名的最大长度不得超过 150 个字符，给定备注的最大长度不得超过 1024 个字符。

```
POST /openapi/v1/assets/user_filehash/upload/{rootAppScopeNameOrID}/{benignOrflagged}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeNameOrID	字符串	根范围名称或 ID。
benignOrflagged	字符串	可以是良性或已标记的其中之一。

响应对象：无

示例 python 代码

```
  # Sample CSV File
  # HashType,FileHash,FileName,Notes
  # SHA-1,1AF17E73721DBE0C40011B82ED4BB1A7DBE3CE29,application_1.exe,Sample Notes
  #
SHA-256,8F434346648F6B96DF89DDA901C5176B10A6D83961DD3C1AC88B59B2DC327AA4,application_2.exe,Sample
 Notes

  file_path = '/<path_to_file>/user_filehash.csv'
  root_app_scope_name = 'Tetration'
  restclient.upload(file_path, '/assets/user_filehash/upload/%s/benign' % root_app_scope_name)
```

用户文件散列删除

此终端用于上传 CSV 文件，以便从 Cisco Secure Workload 设备上的根范围删除文件散列。CSV 文件必须以 `FileHash` 作为标头。

```
POST /openapi/v1/assets/user_filehash/delete/{rootAppScopeNameOrID}/{benignOrflagged}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeNameOrID	字符串	根范围名称或 ID。
benignOrflagged	字符串	可以是 `benign` 和 `flagged` 之一。

响应对象：无

示例 python 代码

```
  # Sample CSV File
  # FileHash
  # 1AF17E73721DBE0C40011B82ED4BB1A7DBE3CE29
  # 8F434346648F6B96DF89DDA901C5176B10A6D83961DD3C1AC88B59B2DC327AA4

  file_path = '/<path_to_file>/user_filehash.csv'
  root_app_scope_name = 'Tetration'
  restclient.upload(file_path, '/assets/user_filehash/delete/' + root_app_scope_name +
'/benign')
```

用户文件散列下载

此终端会将 Cisco Secure Workload 设备上给定根范围的用户文件散列作为 CSV 文件返回。CSV 文件包含按相应顺序排列的标头 `HashType`、`FileHash`、`FileName` 和 `Notes`。

```
GET /openapi/v1/assets/user_filehash/download/{rootAppScopeNameOrID}/{benignOrflagged}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeNameOrID	字符串	根范围名称或 ID。
benignOrflagged	字符串	可以是良性或已标记的其中之一。

响应对象：无

示例 python 代码

```
  file_path = '/<path_to_file>/output_user_filehash.csv'
  root_app_scope_name = 'Tetration'
  restclient.download(file_path, '/assets/user_filehash/download/%s/benign' %
root_app_scope_name)
```

用户定义的标签

这些 API 用于添加或删除用于标记 Cisco Secure Workload 设备上的流和资产项目的用户定义标签。要调用这些 API，请使用具有 `user_data_upload` 功能的 API 密钥。有关用于管理用于标记流和资产项目的键和值的准则，请参阅 UI 用户指南的标签架构部分。

Note 有关通过 UI 访问此功能的说明，请参阅导入自定义标签。

Note 有关可上传的 IPv4/IPv6 地址或子网数量的限制，请参阅标签限制。

范围相关 API

以下 API 用于获取/设置/删除 Cisco Secure Workload 设备上根范围内的标签。它们可供根**范围所有者**和**站点管理员**使用。此外，对根范围具有**读取访问权限**的用户也可以使用 GET API 调用。

获取资产标签

此终端会返回 Cisco Secure Workload 设备上根范围内的 IPv4/IPv6 地址或子网的标签。用于查询此终端的地址/子网必须与用于上传标签的地址/子网完全匹配。

```
GET /openapi/v1/inventory/tags/{rootAppScopeName}?ip={IPorSubnet}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeName	字符串	根范围名称。
IPorSubnet	字符串	IPv4/IPv6 地址或子网。

响应对象：

名称	类型	说明
attributes	JSON	用于标记匹配流和资产项目的键/值映射

示例 python 代码

```
root_app_scope_name = 'Tetration'
restclient.get('/inventory/tags/%s' % root_app_scope_name, params={'ip': '10.1.1.1/24'})
```

搜索库存标签

此终端允许在 Cisco Secure Workload 设备上的根范围内搜索 IPv4/IPv6 地址或子网的标签。

```
GET /openapi/v1/inventory/tags/{rootAppScopeName}/search?ip={IPorSubnet}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeName	字符串	根范围名称。
IPorSubnet	字符串	IPv4/IPv6 地址或子网。

响应对象：此 API 返回以下格式的对象列表

名称	类型	说明
key	字符串	IPv4/IPv6 地址或子网。
updatedAt	整数	标签被更新时的 Unix 时间戳。
value	JSON	键的属性的键/值映射。

示例 python 代码

```
root_app_scope_name = 'Tetration Scope'
encoded_root_app_scope_name = urllib.quote(root_app_scope_name, safe='')
restclient.get('/inventory/tags/%s/search' % encoded_root_app_scope_name, params={'ip':
'10.1.1.1/24'})
```

设置资产标签

此终端用于设置标签，用于标记 Cisco Secure Workload 设备上根范围内的流和资产项目。

```
POST /openapi/v1/inventory/tags/{rootAppScopeName}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeName	字符串	根范围名称。

JSON 查询正文包含以下键

名称	类型	说明
ip	字符串	IPv4/IPv6 地址或子网。
attributes	JSON	用于标记匹配流和资产项目的键/值映射

响应对象：

名称	类型	说明
warnings	JSON	包含设置标签时遇到的警告的键/值映射。

示例 python 代码

```
  root_app_scope_name = 'Tetration'
  req_payload = {'ip': '10.1.1.1/24', 'attributes': {'datacenter': 'SJC', 'location': 'CA'}}

  restclient.post('/inventory/tags/%s' % root_app_scope_name,
json_body=json.dumps(req_payload))
```

删除资产标签

此终端会删除 Cisco Secure Workload 设备上根范围内的 IPv4/IPv6 地址或子网的标签。

```
DELETE /openapi/v1/inventory/tags/{rootAppScopeName}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeName	字符串	根范围名称。

JSON 查询正文包含以下键

名称	类型	说明
ip	字符串	IPv4/IPv6 地址或子网

示例 python 代码

```
  root_app_scope_name = 'Tetration'
  req_payload = {'ip': '10.1.1.1/24'}
  restclient.delete('/inventory/tags/%s' % root_app_scope_name,
json_body=json.dumps(req_payload))
```

上传标签

此终端用于上传包含标签的 CSV 文件，从而标记 Cisco Secure Workload 设备上根范围内的流和资产项目。名称为 `IP` 的列标题必须出现在 CSV 文件中。在其余列标题中，最多可使用 32 个列标题来注释流和资产项目。要在标签中使用非英文字符，上传的 csv 文件必须为 UTF-8 格式。

```
POST /openapi/v1/assets/cmdb/upload/{rootAppScopeName}
```

参数：

用户必须提供操作类型 (`X-Tetration-Oper`) 作为此 API 的参数。`X-Tetration-Oper` 可以是以下项之一：

- add：将标签附加到新的和现有的地址/子网。通过在现有标签上选择新标签来解决冲突。例如，如果数据库中地址的标签为 *{“foo”：“1”，“bar”：“2”}*，而 CSV 文件包含 *{“z”：“1”，“bar”：“3”}*，*add* 会为该地址将标签设置为 *{“foo”：“1”，“z”：“1”，“bar”：“3”}*。
- overwrite：为新地址/子网插入标签，并替换现有地址/子网的标签。例如，如果数据库中地址的标签为 *{“foo”：“1”，“bar”：“2”}*，而 CSV 文件包含 *{“z”：“1”，“bar”：“3”}*，*overwrite* 会为该地址将标签设置为 *{“z”：“1”，“bar”：“3”}*。
- merge：将标签合并到现有地址/子网。通过选择非空值而不是空值来解决冲突。例如，如果数据库中地址的标签为 *{“foo”：“1”，“bar”：“2”，“qux”：“”，“corge”：“4”}*，而 CSV 文件包含 *{“z”：“1”，“bar”：“”，“qux”：“3”，“corge”：“4-updated”}*，*merge* 会为该地址将标签设置为 *{“foo”：“1”，“z”：“1”，“bar”：“2”，“qux”：“3”，“corge”：“4-updated”}*。

Note 其中的“bar”值不会重置为“”（空），而是保留现有的“bar”=”2”值。

- delete：删除地址/子网的标签。

响应对象：

名称	类型	说明
warnings	JSON	包含设置标签时遇到的警告的键/值映射。

示例 python 代码

```
file_path = '/<path_to_file>/user_annotations.csv'
root_app_scope_name = 'Tetration'
```

```
req_payload = [tetpyclient.MultiPartOption(key='X-Tetration-Oper', val='add')]
restclient.upload(file_path, '/assets/cmdb/upload/%s' % root_app_scope_name, req_payload)
```

使用 JSON 格式上传标签

此终端用于上传标签，以便使用 JSON 格式在 Cisco Secure Workload 设备上标记流和资产项目。ip_tags 中的属性必须是标头的子集，最多可有 32 个标头用于注释流和资产项目。要在标签中使用非英文字符，json 负载必须是 UTF-8 格式。

```
POST /openapi/v1/multi_inventory/tags/{rootAppScopeName}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeName	字符串	根范围名称

Table 11: 负载参数

名称	类型	说明
headers	数组	指定标签名称的字符串数组
operation	字符串	**add** 或 **merge**，如未指定，则将其视为 **add**。
ip_tags	数组	ip_tags 对象数组

Table 12: ip_tags

名称	类型	说明
ip	字符串	有效的 IPV4/IPV6 地址或子网
attributes	JSON	标签、值字符串对的 JSON；标签必须是标头的子集。

Note

1. 如果记录标签不是给定标头的子集，则会生成警告，提醒用户给定标头与记录中的标签之间存在差异。

2. 此终端仅允许用户上传最多 95,000 个条目和 36 个属性。

- 对于 add 请求，*operation* 是可选的，如未指定，则将其视为 **add**。
- 对于 merge 请求，必须将 *operation* 指定为 **merge**。

3. 您最多可以上传 95,000 个条目和 36 个属性。

用于 add 请求的 Python 代码示例

```
  {
    "headers" : ["key1", "key2"],
    "operation": "add"
    "ip_tags" : [
        {
            "ip": "10.10.10.11",
            "attributes": {
                "key1": "val1",
                "key2": "val2"
            }
        },
        {
            "ip": "10.10.10.12",
            "attributes": {
                "key1": "val111",
                "key2": "val2"
            }
        }
    ]
}
```

用于 merge 请求的示例 Python 代码

```
  {{
    "headers" : ["key1", "key2"],
    "operation": "merge"
    "ip_tags" : [
        {
            "ip": "10.10.10.11",
            "attributes": {
                "key1": "val1",
                "key2": "val2"
            }
        },
        {
            "ip": "10.10.10.12",
            "attributes": {
                "key1": "val1",
                "key2": "val2"
            }
        }
    ]
}
resp = restclient.post('/multi_inventory/tags/%s' % rootAppScopeName,
json_body=json.dumps(req_payload))
```

下载用户标签

此终端会将 Cisco Secure Workload 设备上根范围的用户上传标签作为 CSV 文件返回。

```
GET /openapi/v1/assets/cmdb/download/{rootAppScopeName}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeName	字符串	根范围名称。

响应：

内容类型：*text/csv*

包含用户上传的范围标签的 CSV 文件。

示例 python 代码

```
file_path = '/<path_to_file>/output.csv'
root_app_scope_name = 'Tetration'
restclient.download(file_path, '/assets/cmdb/download/%s' % root_app_scope_name)
```

获取列标题

此终端会返回 Cisco Secure Workload 设备上根范围的列标题列表。

```
GET /openapi/v1/assets/cmdb/attributenames/{rootAppScopeName}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeName	字符串	根范围名称。

响应对象：可用于标签的分面数组。

示例 python 代码

```
root_app_scope_name = 'Tetration'
resp = restclient.get('/assets/cmdb/attributenames/%s' % root_app_scope_name)
```

删除列标题

此终端将删除 Cisco Secure Workload 设备上根范围内的列标题。删除列标题会将其从已标记分面列表中删除，并将其从现有标签中删除。

```
DELETE /openapi/v1/assets/cmdb/attributenames/{rootAppScopeName}/{attributeName}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeName	字符串	根范围名称。
attributeName	字符串	正在删除的属性。

响应对象：无

示例 python 代码

```
root_app_scope_name = 'Tetration'
attribute_name = 'column1'
resp = restclient.delete('/assets/cmdb/attributenames/%s/%s' % (root_app_scope_name,
attribute_name))
```

使用 JSON 格式删除标签

此终端使用 JSON 格式删除多个 IPv4/IPv6 地址或子网的标签。

```
DELETE /openapi/v1/multi_inventory/tags/{rootAppScopeName}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeName	字符串	根范围名称

Table 13: 负载参数

名称	类型	说明
headers	数组	（可选）指定标签名称的字符串数组
ip_tags	数组	ip_tags 对象数组

Table 14: ip_tags

名称	类型	说明
ip	字符串	有效的 IPV4/IPV6 地址或子网
attributes	JSON	（可选）标签、值字符串对的 JSON；标签必须是标头的子集。

示例 python 代码

```
  {
    "ip_tags" : [
         {
             "ip": "10.10.10.11",
           },
         {
             "ip": "10.10.10.12",
             "attributes": {
                 "key1": "val1",
                 "key2": "val2"
             }
         }
    ]
}
resp = restclient.delete('/multi_inventory/tags/%s' % rootAppScopeName,
json_body=json.dumps(req_payload))
```

获取已标记分面的列表

此终端会返回 Cisco Secure Workload 设备上根范围的已标记分面列表。带标签的分面是上传的 CSV 文件中列标题的子集，用于注释该范围内的流和资产项目。

```
GET /openapi/v1/assets/cmdb/annotations/{rootAppScopeName}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeName	字符串	根范围名称。

响应对象：根范围的已标记分面数组。

示例 python 代码

```
root_app_scope_name = 'Tetration'
resp = restclient.get('/assets/cmdb/annotations/%s' % root_app_scope_name)
```

更新已标记分面的列表

此终端更新用于在 Cisco Secure Workload 设备上的根范围内注释流和资产项目的分面列表。

```
PUT /openapi/v1/assets/cmdb/annotations/{rootAppScopeName}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeName	字符串	根范围名称。

响应对象：无

示例 python 代码

```
# the following list is a subset of column headers in the
# uploaded CSV file
req_payload = ['location', 'region', 'detail']
root_app_scope_name = 'Tetration'
restclient.put('/assets/cmdb/annotations/%s' % root_app_scope_name,
               json_body=json.dumps(req_payload))
```

刷新用户上传的标签

此终端会刷新 Cisco Secure Workload 设备上根范围内的流和资产项目的标签。更改会影响新数据；较旧的已标记数据保持不变。

```
POST /openapi/v1/assets/cmdb/flush/{rootAppScopeName}
```

参数：请求 URL 包含以下参数

名称	类型	说明
rootAppScopeName	字符串	根范围名称。

响应对象：无

示例 python 代码

```
root_app_scope_name = 'Tetration'
restclient.post('/assets/cmdb/flush/%s' % root_app_scope_name)
```

范围无关 API

以下 API 可以跨越 Cisco Secure Workload 设备上的多个范围。

Note 对于任何根范围，范围独立和相关的带注释的分面数量不得超过 32。

上传标签

此终端用于上传包含标签的 CSV 文件，以便标记 Cisco Secure Workload 设备上的流和资产项目。CSV 文件中必须显示名为 `IP` 和 `VRF` 的列标题，并且 `VRF` 必须与标签的根范围匹配。在其余列标题中，最多可使用 32 个列标题来注释流和资产项目。

```
POST /openapi/v1/assets/cmdb/upload
```

参数：

用户必须提供操作类型 (`X-Tetration-Oper`) 作为此 API 的参数，以指定要执行的操作。

响应对象：

名称	类型	说明
warnings	JSON	包含设置标签时遇到的警告的键或值映射。

示例 python 代码

```
file_path = '/<path_to_file>/user_annotations.csv'
req_payload = [tetpyclient.MultiPartOption(key='X-Tetration-Oper', val='add')]
restclient.upload(file_path, '/assets/cmdb/upload', req_payload)
```

下载用户标签

此终端会将 Cisco Secure Workload 设备上所有范围的用户上传的标签作为 CSV 文件返回。

```
GET /openapi/v1/assets/cmdb/download
```

参数：无

响应：

内容类型：*text/csv*

包含用户上传的范围标签的 CSV 文件。

示例 python 代码

```
file_path = '/<path_to_file>/output.csv'
restclient.download(file_path, '/assets/cmdb/download')
```

范围无关标签

这些标签与特定的根范围无关，适用于设备上的所有范围。

获取资产标签

这些终端会返回 Cisco Secure Workload 设备上 IPv4/IPv6 地址或子网的范围无关标签。用于查询此终端的地址或子网必须与用于上传标签的地址或子网完全匹配。

```
GET /openapi/v1/si_inventory/tags?ip={IPorSubnet}
```

参数：请求 URL 包含以下参数。

名称	类型	说明
IPorSubnet	字符串	IPv4/IPv6 地址或子网。

响应对象：

名称	类型	说明
attributes	JSON	用于标记匹配的流和资产项目的键或值映射

示例 python 代码

```
restclient.get('/si_inventory/tags', params={'ip': '10.1.1.1/24'})
```

搜索库存标签

此终端允许在标签中搜索 Cisco Secure Workload 设备上的 IPv4/IPv6 地址或子网。

```
GET /openapi/v1/si_inventory/tags/search?ip={IPorSubnet}
```

参数：请求 URL 包含以下参数

名称	类型	说明
IPorSubnet	字符串	IPv4/IPv6 地址或子网。

响应对象：此 API 返回以下格式的对象列表

名称	类型	说明
key	字符串	IPv4/IPv6 地址或子网。
updatedAt	整数	更新标签时的 Unix 时间戳。
value	JSON	键的属性的键或值映射。

示例 python 代码

```
restclient.get('/si_inventory/tags/search', params={'ip': '10.1.1.1/24'})
```

设置资产标签

此终端用于设置标签，以标记 Cisco Secure Workload 设备上的流和资产项目。

```
POST /openapi/v1/si_inventory/tags
```

参数：JSON 查询正文包含以下键

名称	类型	说明
ip	字符串	IPv4/IPv6 地址或子网。
attributes	JSON	用于标记匹配的流和资产项目的键或值映射。

响应对象：

名称	类型	说明
warnings	JSON	包含设置标签时遇到的警告的键或值映射。

示例 python 代码

```
req_payload = {'ip': '10.1.1.1/24', 'attributes': {'datacenter': 'SJC', 'location': 'CA'}}

restclient.post('/si_inventory/tags', json_body=json.dumps(req_payload))
```

删除资产标签

此终端会删除 Cisco Secure Workload 设备上 IPv4/IPv6 地址或子网的标签。

```
DELETE /openapi/v1/si_inventory/tags
```

参数：JSON 查询正文包含以下键

名称	类型	说明
ip	字符串	IPv4/IPv6 地址或子网

示例 python 代码

```
req_payload = {'ip': '10.1.1.1/24'}
restclient.delete('/si_inventory/tags, json_body=json.dumps(req_payload))
```

获取已标记分面的列表

此终端会返回 Cisco Secure Workload 设备上与范围无关的已标记分面列表。已标记分面是列标题的子集，用于在所有范围内注释流和资产项目。

Note 从请求 URL 中排除范围名称，以查看和更新带注释的范围无关分面列表。

```
GET /openapi/v1/assets/cmdb/annotations
```

响应对象：与范围无关的已标记分面数组。

示例 python 代码

```
resp = restclient.get('/assets/cmdb/annotations')
```

更新已标记分面的列表

此终端会更新用于注释 Cisco Secure Workload 设备上的流和资产项目的独立于范围的分面的列表。

```
PUT /openapi/v1/assets/cmdb/annotations
```

响应对象：无

示例 python 代码

```
# the following list is a subset of column headers in the
# uploaded CSV file
req_payload = ['location', 'region', 'detail']
restclient.put('/assets/cmdb/annotations',
               json_body=json.dumps(req_payload))
```

虚拟路由和转发

这组 API 可以管理虚拟路由和转发 (VRF)。

Note 这些 API 仅适用于站点管理员。

VRF 对象

VRF 对象属性如下所述：

属性	类型	说明
id	整数	VRF 的唯一标识符。
name	字符串	用户指定的 VRF 名称。
tenant_id	整数	父租户的 ID。
root_app_scope_id	字符串	关联根范围的 ID。
created_at	整数	创建 VRF 时的 Unix 时间戳。

属性	类型	说明
updated_at	整数	上次更新 VRF 时的 Unix 时间戳。

获取 VRF

此终端会返回 VRF 列表。此 API 可用于具有 `sensor_management or flow_inventory_query` 功能的 API 密钥。

```
GET /openapi/v1/vrfs
```

参数：无

响应对象：返回 VRF 对象列表。

示例 python 代码

```
resp = restclient.get('/vrfs')
```

创建 VRF

此终端用于创建新的 VRF。系统将使用与 VRF ID 匹配的查询来自动创建关联的根范围。此 API 可用于具有 `sensor_management` 功能的 API 密钥。

```
POST /openapi/v1/vrfs
```

参数：

名称	类型	说明
id	整数	（可选）VRF 的唯一标识符。如果未指定，Cisco Secure Workload 集群将为新创建的 VRF 生成唯一 ID。最佳实践是让 Cisco Secure Workload 生成这些 ID，而不是调用方明确指定唯一 ID。
tenant_id	整数	（可选）父租户的 ID。
name	字符串	用户指定的 VRF 名称。
apply_monitoring_rules	布尔值	（可选）是否应将收集规则应用于 VRF。默认值为“false”。

`tenant_id` 为可选。如果未提供，则 VRF 将使用与 VRF 相同的 ID 添加到租户，并在必要时自动创建。如果提供了 `tenant_id`，则不会自动创建租户；如果租户不存在，则会返回错误。

响应对象：返回新创建的 VRF 对象。

示例 python 代码

```
req_payload = {
    "tenant_id": <tenant_id>,
    "name": "Test",
    "apply_monitoring_rules": True
}
resp = restclient.post('/vrfs', json_body=json.dumps(req_payload))
```

获取特定 VRF

此终端会返回指定 VRF ID 的信息。此 API 可用于具有 `sensor_management or flow_inventory_query` 功能的 API 密钥。

```
GET /openapi/v1/vrfs/{vrf_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
vrf_id	整数	VRF 的唯一标识符。

响应对象：返回与指定 ID 关联的 VRF 对象。

示例 python 代码

```
vrf_id = 676767
resp = restclient.get('/vrfs/%d'% vrf_id)
```

更新 VRF

此终端会更新 VRF。此 API 可用于具有 `sensor_management` 功能的 API 密钥。

```
PUT /openapi/v1/vrfs/{vrf_id}
```

参数：请求 URL 包含以下参数

名称	类型	说明
vrf_id	整数	VRF 的唯一标识符。

JSON 请求正文包含以下参数

名称	类型	说明
name	字符串	用户指定的 VRF 名称。
apply_monitoring_rules	布尔值	（可选）是否应将收集规则应用于 VRF。

响应对象：返回与指定 ID 关联的已修改 VRF 对象。

示例 python 代码

```
vrf_id = 676767
req_payload = {
    "name": "Test",
    "apply_monitoring_rules": True
}
resp = restclient.put('/vrfs/%d'% vrf_id,
                      json_body=json.dumps(req_payload))
```

删除特定 VRF

此终端会删除 VRF。如果存在关联的根范围，则会失败。此 API 可用于具有 `sensor_management` 功能的 API 密钥。

```
DELETE /openapi/v1/vrfs/{vrf_id}
```

参数：以下参数是 URL 的一部分。

名称	类型	说明
vrf_id	整数	VRF 的唯一标识符。

示例 python 代码

```
vrf_id = 676767
resp = restclient.delete('/vrfs/%d'% vrf_id)
```

协调器

这组 API 可用于管理 Cisco Secure Workload 集群部署中的外部协调器资产学习。它们需要使用与 API 密钥关联的 `external_integration` 功能。

目前支持的协调器类型包括“vcenter”（vCenter 6.5 及更高版本）、“kubernetes”、“dns”、“f5”、“netscaler”、“infoblox”和“Cisco FMC”。支持的用户界面位于外部协调器。

协调器对象

协调器对象属性说明如下 - 某些字段仅适用于特定的协调器类型；下表中提到了相关限制。

属性	类型	说明
id	字符串	协调器的唯一标识符。
name	字符串	用户指定的协调器名称。
type	字符串	协调器类型 - 支持的值（*vcenter*、*kubernetes*、*f5*、*netscaler*、*infoblox*、*dns*）
description	字符串	用户指定的协调器说明。

属性	类型	说明
username	字符串	协调终端的用户名。（对于 *dns* 不需要）
password	字符串	协调终端的密码。（对于 *dns* 不需要）
certificate	字符串	用于身份验证的客户端证书（对于 *dns* 不需要）
key	字符串	与客户端证书对应的密钥（对于 *dns* 不需要）
ca_certificate	字符串	用于验证协调终端的 CA 证书（对于 *DNS*不需要）
auth_token	字符串	不透明身份验证令牌（持有者令牌）（仅适用于 *Kubernetes*）
insecure	布尔值	关闭严格 SSL 验证
delta_interval	整数	增量轮询间隔（以秒为单位）Cisco Secure Workload 资产管理器将每隔 delta_interval 秒对增量更改执行轮询。请注意，此参数不适用于 Infoblox 和 Cisco Secure Firewall Management Center。
full_snapshot_interval	整数	完整快照间隔（以秒为单位）Cisco Secure Workload 资产管理器将从协调器执行完整刷新轮询
verbose_tsdb_metrics	布尔值	每终端 TSDB 指标
hosts_list	数组	{“host_name”, port_number} 对的数组，指定 Cisco Secure Workload 必须如何连接到协调器
use_secureconnector_tunnel	布尔值	通过安全连接器隧道建立到此协调器主机的隧道连接
route_domain	整数	要在 F5 负载均衡器上轮询的路由域编号（仅适用于 *f5*）
dns_zones	数组	包含要从 DNS 服务器轮询的 DNS 区域的字符串数组（仅适用于 *dns*）。每个 DNS 区域条目必须以 . 结尾

属性	类型	说明
enable_enforcement	布尔值	仅适用于具有策略执行支持的外部协调器，例如防火墙和负载均衡器。示例包括 *Cisco Secure Firewall Management Center*、*F5 BIGIP* 和 *Citrix Netscaler*。默认情况下，此标志为 false（禁用策略实施）。如果为 true，则在对工作空间执行策略执行时，外部协调器会将策略部署到给定的负载均衡器设备。
ingress_controllers	对象	入口控制器对象数组。
fmc_enforcement_mode	字符串	仅适用于 *Cisco Secure Firewall Management Center* 外部协调器，并且必须为 *merge*（默认值）或 *override*。第一个实例指示 Cisco Secure Firewall Management Center 策略执行器将所有 Cisco Secure Workload 策略规则放在任何现有预过滤器规则之前，而后一个实例将删除用户创建的所有预过滤器规则。
infoblox_config	对象	仅适用于 *Infoblox* 外部协调器。Infoblox 配置记录类型选择器。

入口控制器

属性	类型	说明
pod_selector	对象	Pod 选择器
controller_config	对象	控制器配置

Pod 选择器

属性	类型	说明
namespace	字符串	运行入口控制器 Pod 的命名空间。

属性	类型	说明
labels	数组	由 {“key”，“value”} 对组成的数组，用于指定入口控制器 Pod 的标签。

控制器配置

属性	类型	说明
ingress_class	字符串	入口控制器满足的入口类的名称。
namespace	字符串	命名空间是入口控制器满足的命名空间的名称。
http_ports	数组	http 端口数组。
https_ports	数组	https 端口数组。

Infoblox 配置

enable_network_record	布尔值	默认值为 True。如果为 false，则禁用网络类型记录。
enable_host_record	布尔值	默认值为 True。如果为 false，则禁用主机类型记录。
enable_a_record	布尔值	默认值为 True。如果为 false，则禁用 *a* 类型的记录。
enable_aaaa_record	布尔值	默认值为 True。如果为 false，则禁用 *aaaa* 类型的记录。

协调器对象中的只读状态字段

属性	类型	说明
authentication_failure	布尔值	与 Cisco Secure Workload 协调器的连接状态 - *true* 表示已成功连接到协调器。如果此字段为 *false*，则 *authentication_failure_error* 字段将提供详细的错误消息，以说明失败的原因

属性	类型	说明
authentication_failure_error	字符串	详细的错误消息，可帮助调试协调器的连接或凭证故障
scope_id	字符串	将发布和显示资产的租户根范围 ID

获取协调器

此终端会返回 Cisco Secure Workload 设备已知的协调器列表。此 API 可用于具有 `external_integration` 功能的 API 密钥。

```
GET /openapi/v1/orchestrator/{scope}
```

参数：无

返回提供的根范围的协调器对象列表。范围必须是根范围 ID。

创建协调器

此终端将用于创建协调器。

```
POST /openapi/v1/orchestrator/{scope}
```

vCenter 协调器的示例 Python 代码

```
req_payload = {
    "name": "VCenter Orchestrator"
    "type": "vcenter",
    "hosts_list": [ { "host_name": "8.8.8.8", "port_number": 443}],
    "username":"admin",
    "password":"admin"
}
resp = restclient.post('/orchestrator/Default', json_body=json.dumps(req_payload))
```

DNS 协调器的示例 Python 代码

```
req_payload = {
    "name": "DNS Server"
    "type": "dns",
    "hosts_list": [ { "host_name": "8.8.8.8", "port_number": 53}],
    "dns_zones": [ "lab.corp.com.", "dev.corp.com." ]
}
resp = restclient.post('/orchestrator/Default', json_body=json.dumps(req_payload))
```

Kubernetes 协调器的示例 Python 代码

```
req_payload = {
    "name": "k8s"
    "type": "kubernetes",
    "hosts_list": [ { "host_name": "8.8.8.8", "port_number": 53}],
    "certificate": "",
    "key": "",
    "ca_certificate": "",
```

```
}
resp = restclient.post('/orchestrator/Default', json_body=json.dumps(req_payload))
```

具有入口控制器的 Kubernetes 协调器的示例 Python 代码

请参阅有关 Kubernetes/OpenShift 外部协调器的信息，以创建身份验证详细信息。

```
req_payload = {
    "name": "k8s",
    "type": "kubernetes",
    "hosts_list": [ { "host_name": "8.8.8.8", "port_number": 53}],
    "certificate": "",
    "key": "",
    "ca_certificate": "",
    "ingress_controllers": [
        {
            "pod_selector": {
                "namespace": "ingress-nginx",
                "labels": [{ "key": "app", "value": "nginx-ingress"}],
            }
        }
    ]
}
resp = restclient.post('/orchestrator/Default', json_body=json.dumps(req_payload))
```

具有多个入口控制器的 Kubernetes 协调器的示例 Python 代码

请参阅有关 Kubernetes/OpenShift 外部协调器的信息，以创建身份验证详细信息。

```
req_payload = {
    "name": "k8s",
    "type": "kubernetes",
    "hosts_list": [ { "host_name": "8.8.8.8", "port_number": 53}],
    "certificate": "",
    "key": "",
    "ca_certificate": "",
    "ingress_controllers": [
        {
            "pod_selector": {
                "namespace": "ingress-nginx",
                "labels": [{ "key": "app", "value": "nginx-ingress"}],
            },
            "controller_config": {
                "ingress_class": "nginx-class",
            }
        },
        {
            "pod_selector": {
                "namespace": "ingress-haproxy",
                "labels": [{ "key": "app", "value": "haproxy-ingress"}],
            },
            "controller_config": {
                "ingress_class": "haproxy-class",
                "http_ports": [8080],
                "https_ports": [8443],
                "namespace": "haproxy-watching-namespace"
            }
        }
    ],
}
resp = restclient.post('/orchestrator/Default', json_body=json.dumps(req_payload))

** Type AWS and EKS are no longer supported in external orchestrators. They have been
```

```
ported to
     connectors.
```

获取特定协调器

此终端节点会返回一个协调器实例。

```
GET /openapi/v1/orchestrator/{scope}/{orchestrator_id}
```

返回与指定 ID 相关联的协调器对象。

更新协调器

此终端会更新协调器。

```
PUT /openapi/v1/orchestrator/{scope}/{orchestrator_id}
```

参数：

与 POST 参数相同

删除特定协调器

此终端将删除指定的协调器。

```
DELETE /openapi/v1/orchestrator/{scope}/{orchestrator_id}
```

协调器黄金规则

这组 API 可用于管理外部 Kubernetes 协调器的黄金规则。黄金规则是在允许列表执行模式下确保 Kubernetes 控制平面连接的必要条件。它们需要使用与 API 密钥关联的 `external_integration` 功能。

目前支持的黄金规则协调器类型只有“kubernetes”。非 Kubernetes 协调器向此终端发出的请求将失败。

协调器黄金规则对象

协调器对象属性如下所述：

属性	类型	说明
kubelet_port	整数	Kubelet 节点本地 API 端口
services	数组	Kubernetes 服务对象数组

获取协调器黄金规则

此终端会返回与协调器关联的黄金规则。此 API 可用于具有 `external_integration` 功能的 API 密钥。

```
GET /openapi/v1/orchestrator/{scope}/{id}/gr
```

参数：无

返回单个黄金规则对象

创建或更新黄金规则

此终端用于为现有协调器创建或更新黄金规则。

```
POST /openapi/v1/orchestrator/{scope}/{id}/gr
```

参数：

属性	类型	说明
kubelet_port	整数	Kubelet 节点本地 API 端口
services	数组	Kubernetes 服务对象数组

示例 python 代码

```
  req_payload = {
              "kubelet_port":10255,
              "services": [
                          {
    "description": "kube-dns",
    "addresses":   [ "10.0.1.1:53/TCP", "10.0.1.1:53/UDP" ],
    "consumed_by":  [ "NODES", "PODS"],
   }
             ]
  }
  resp = restclient.post('/orchestrator/{scope_id}/{orchestrator_id}/gr',
json_body=json.dumps(req_payload))
```

FMC 协调器域

这组 API 可用于管理外部 FMC 协调器的域。需要 FMC 域才能在给定 FMC 域上启用执行。它们需要使用与 API 密钥关联的 `external_integration` 功能。

目前支持的 FMC 域协调器类型只有“fmc”。非 FMC 协调器向此终端发出的请求都会失败。

协调器 FMC 域对象

协调器对象属性如下所述：

属性	类型	说明
fmc_domains	数组	FMC 域对象数组

FMC 域对象属性如下所述：

属性	类型	说明
name	字符串	FMC 域的名称
enforcement_enabled	布尔值	此标志默认设置为 false。如果为 true，则在对工作空间执行策略执行时，外部协调器会将策略部署到与“name”匹配的域。

URL 属性如下所述：

属性	类型	说明
scope	字符串	将在其中发布和显示资产的租户根范围名称或 ID
orchestrator_id	字符串	FMC 协调器的协调器 ID

获取 FMC 域

此终端会返回在与 FMC 协调器关联的 FMC 上配置的 FMC 域。此 API 可用于具有 `external_integration` 功能的 API 密钥。

```
GET /openapi/v1/orchestrator/{scope}/{orchestrator_id}/fmcdomains
```

参数：无

返回具有 FMC 域对象属性列表的 JSON 对象。

更新 FMC 外部协调器的 FMC 域配置

此终端可更新现有 FMC 外部协调器的 FMC 域属性。

```
PUT /openapi/v1/orchestrator/{scope}/{orchestrator_id}/fmcdomains
```

参数：

属性	类型	说明
fmc_domains	数组	FMC 域对象数组

FMC 域对象属性如下所述：

属性	类型	说明
name	字符串	FMC 域的名称
enforcement_enabled	布尔值	此标志默认设置为 false。如果为 true，则在对工作空间执行策略执行时，外部协调器会将策略部署到与“name”匹配的域。

URL 属性如下所述：

属性	类型	说明
scope	字符串	将在其中发布和显示资产的租户根范围名称或 ID
orchestrator_id	字符串	FMC 协调器的协调器 ID

示例 python 代码

```
  req_payload = {
        "fmc_domains":    [
        {
            "enforcement_enabled": False,
            "name": "Global/Eng"
        },
        {
            "enforcement_enabled": True,
            "name": "Global/Prod"
        }
    ]
  }
  resp = restclient.put('/orchestrator/{scope}/{orchestrator_id}/fmcdomains',
json_body=json.dumps(req_payload))
```

RBAC（基于角色的访问控制）注意事项

要访问根范围下的协调器，要求用于请求的 API 密钥具有必要的权限。所有协调器 API 调用都有范围，并且始终需要将根范围 ID 作为 URL 的一部分。协调器始终位于根范围级别，并且无法在子范围下创建。特定租户根范围下创建的协调器（以及这些协调器了解到的资产）对其他租户是不可见的。

对于可能配置了多个路由域 (vrfs) 的 F5 负载均衡器，F5 路由域过滤逻辑会跨所有分区扫描 F5 上的所有实体，但会丢弃未评估的实体（服务、snat 池、池和后端）F5 协调器 *route_domain* 字段中指定的路由域。

高可用性和故障转移注意事项

hosts_list 参数允许为协调器配置多个服务器地址。在有多个服务器地址的情况下，每种协调器类型的 Cisco Secure Workload 服务器选择逻辑都不尽相同。

对于 *vCenter*、*Kubernetes*、*DNS*、*F5*、*Netscaler*、*Infoblox*，选择以第一个运行状况正常的终端为基础。连接并非持久（*Kubernetes* 除外），因此，在每个轮询期间，安全连接器协调器管理器都将扫描主机，并轮询 hosts_list 中遇到的第一个运行状况正常的终端。对于 *kubernetes*，将维护持久性事件通道，而在连接失败时，将使用下一个运行状况正常的终端执行所有主机扫描和后续完全轮询。

Kubernetes RBAC 资源注意事项

Kubernetes 客户端尝试 GET/LIST/WATCH 以下资源。

提供的 Kubernetes 身份验证凭证应具有对以下资源的最低权限集：

资源	动词
daemonsets	[get list watch]
deployments	[get list watch]
endpoints	[get list watch]
namespaces	[get list watch]
nodes	[get list watch]
pods	[get list watch]
replicasets	[get list watch]
replicationcontrollers	[get list watch]
services	[get list watch]
statefulsets	[get list watch]
daemonsets.apps	[get list watch]
deployments.apps	[get list watch]
endpoints.apps	[get list watch]
namespaces.apps	[get list watch]
nodes.apps	[get list watch]
pods.apps	[get list watch]
replicasets.apps	[get list watch]

资源	动词
replicationcontrollers.apps	[get list watch]
services.apps	[get list watch]
statefulsets.apps	[get list watch]
daemonsets.extensions	[get list watch]
deployments.extensions	[get list watch]
endpoints.extensions	[get list watch]
namespaces.extensions	[get list watch]
nodes.extensions	[get list watch]
pods.extensions	[get list watch]
replicasets.extensions	[get list watch]
replicationcontrollers.extensions	[get list watch]
services.extensions	[get list watch]
statefulsets.extensions	[get list watch]

站点信息

此 API 可用于获取集群信息，如集群状态、集群类型、外部 IP 和邮件。

Note 此 API 仅适用于站点管理员用户。

获取站点信息

此终端节点会返回包含集群站点信息的 JSON 对象。

```
GET /openapi/v1/site_infos
```

参数：无

响应对象：包含集群站点信息的 JSON 对象

示例 Python 代码

```
  resp = restclient.get('/site_infos')
```

响应示例

```
{
  "cluster_state": "Enabled till 2020-12-31 23:59:59 UTC",
  "cluster_uuid": "00000000-0000-0000-0000-000000000000",
  "site_bosun_email": "customer-support@company.com",
  "site_cluster_type": "physical",
  "site_external_ips": [
      "1.1.1.1",
      "1.1.1.2",
      ...
      "1.1.1.7"
  ],
  "site_name": "cluster_name",
  "site_sensor_vip_ip": "2.1.1.1",
  "site_ui_admin_email": "site-admin@company.com",
  "site_ui_fqdn": "cluster.company.com",
  "site_ui_primary_customer_support_email": "customer-support@company.com"
}
```

集群运行状况

此 API 可用于获取 Cisco Secure Workload 中所有物理服务器的状态。

Note 此 API 仅适用于站点管理员用户。

获取集群运行状况

此终端节点会返回包含集群运行状况信息的 JSON 对象。

```
GET /openapi/v1/cluster_nodes
```

参数：无

响应对象：包含集群运行状况信息的 JSON 对象

示例 Python 代码

```
resp = restclient.get('/cluster_nodes')
```

服务运行状况

此 API 可用于获取 Cisco Secure Workload 集群中使用的所有服务及其依赖关系的运行状况。

Note 此 API 仅适用于站点管理员用户。

获取服务运行状况

此终端会返回一个包含服务运行状况信息的 JSON 对象。

```
GET /openapi/v1/service_status
```

参数：无

响应对象：包含服务运行状况信息的 JSON 对象

示例 Python 代码

```
resp = restclient.get('/service_status')
```

安全连接器

OpenAPI 提供终端以管理安全连接器的功能。这些终端需要将 `external_integration` 功能与 API 密钥关联。

Note 安全连接器 API 不能在站点级别使用。它们只能在根范围级别使用。

获取状态

此终端会返回指定根范围的安全连接器隧道的当前状态。

```
GET /openapi/v1/secureconnector/name/{ROOT_SCOPE_NAME}/status
```

```
GET /openapi/v1/secureconnector/{ROOT_SCOPE_ID}/status
```

需要对指定根范围的 READ 权限。

返回的状态是具有以下架构的 json 对象：

键	类型	值
active	布尔值	安全连接器隧道当前处于活动状态
peer	字符串	隧道的安全连接器客户端的 <ip>:<port>
start_time	整数	启动隧道的时间戳（纪元时间，以秒为单位）
last_heartbeat	整数	来自客户端的最后一个心跳的时间戳（纪元时间，以秒为单位）

获取令牌

此终端会返回新的一次性限时令牌，用于引导指定根范围的安全连接器客户端。

```
GET /openapi/v1/secureconnector/name/{ROOT_SCOPE_NAME}/token
```

```
GET /openapi/v1/secureconnector/{ROOT_SCOPE_ID}/token
```

需要指定根范围的 OWNER 权限。

返回的令牌是一个字符串，其中包含有效期为一小时的加密签名令牌。有效令牌只能用于引导一次安全连接器客户端。

轮换证书

此终端会强制为指定的根范围创建新证书。新证书将由安全连接器服务器使用，并将用于对此根范围的客户端的证书签名请求进行签名。

```
POST /openapi/v1/secureconnector/name/{ROOT_SCOPE_NAME}/rotate_certs?invalidate_old=
  →{true|false}
```

```
POST /openapi/v1/secureconnector/{ROOT_SCOPE_ID}/rotate_certs?invalidate_old=   →{true|false}
```

需要指定根范围的 OWNER 权限。

调用此终端后，此根范围的客户端和服务器之间的通信会立即转换为使用新证书。

如果 *invalidate_old* 被设置为 false，则任何现有客户端都将自动创建新的公钥/私钥对，并使用其现有证书为新公钥签署新证书。

如果 *invalidate_old* 被设置为 true，则现有证书将立即失效。任何现有客户端都将无法连接到服务器，并且必须使用新令牌再次进行引导。有关详细信息，请参阅“安全连接器部署”。

Kubernetes 漏洞扫描

获取用于 Pod 漏洞扫描的 Kubernetes 注册表

此终端会返回给定 VRF 的 Cisco Secure Workload 集群中显示的所有 Kubernetes 注册表的列表。

```
GET /openapi/kubernetes/{root_scope_name_or_id}/vulnerability_scanning/registry
```

参数：JSON 查询正文包含以下键

名称	类型	说明
root_scope_name_or_id	字符串	根范围名称或 ID

响应对象：返回具有以下属性的注册表对象数组

属性	类型	说明
id	字符串	注册表的 ID
clusters	数组	使用注册表的 Kubernetes 集群对象数组
connection_status	字符串	定义注册表连接状态
credential_status	字符串	说明是否已提供凭证的状态
last_scanned	Int64	纪元中最后一个扫描程序
url	字符串	注册表的 URL

Kubernetes 集群对象

属性	类型	说明
id	字符串	Kubernetes 集群的 ID
connector_id	字符串	用于载入 Kubernetes 集群的连接器的 ID
connector_type	字符串	用于载入 Kubernetes 集群的连接器类型
name	字符串	Kubernetes 集群的名称。

示例 python 代码

```
root_app_scope_name = 'Tetration'
restclient.get('/kubernetes/%s/vulnerability_scanning/registry' % root_app_scope_name)
```

将凭证添加到 Kubernetes 注册表

此终端允许您为 Kubernetes 注册表添加凭证。接受的凭证基于注册表类型。

例如：

注册表类型：AWS；接受的凭证类型：aws_auth 凭证对象

注册表类型：其他；接受的凭证类型：basic_auth 凭证对象

```
PUT /openapi/kubernetes/{root_scope_name_or_id}/vulnerability_scanning/registry/{registry_id}
```

参数：JSON 查询正文包含以下键：

名称	类型	说明
root_scope_name_or_id	字符串	根范围名称或 ID
registry_id	字符串	Kubernetes 注册表 ID

名称	类型	说明
registry_credential	对象	凭证对象

凭证对象

属性	类型	说明
basic_auth	对象	基本身份验证凭证对象
aws_auth	对象	AWS 身份验证凭证对象
azure_auth	对象	Azure 身份验证凭证对象
gcp_auth	对象	GCP 身份验证凭证对象

basic_auth 对象

属性	类型	说明
username	字符串	用户名
password	字符串	密码

aws_auth 对象

属性	类型	说明
aws_access_key_id	字符串	AWS 凭证访问密钥
aws_secret_access_key	字符串	AWS 凭证访问秘密

azure_auth 对象：

属性	类型	说明
azure_tenant_id	字符串	Azure 租户 ID
azure_client_id	字符串	Azure 客户端 ID
azure_client_secret	字符串	Azure 客户端秘密

gcp_auth 对象：

属性	类型	说明
gcp_service_account	对象	GCP 服务帐户

示例 python 代码

```
root_app_scope_name = 'Tetration'
```

```
registry_id = '64cdc7a7362f57192dcc1625'
pay_load = {
        "registry_credential": {
            "basic_auth": {
                "username": "username",
                "password": "password",
            }
        }
    }
restclient.put('/kubernetes/%s/vulnerability_scanning/registry/%s' % root_app_scope_name,
registry_id, json_body=json.dumps(pay_load))
```

获取 Kubernetes Pod 扫描程序

此终端会返回给定 VRF 的 Cisco Secure Workload 集群中显示的所有 Kubernetes Pod 扫描程序的列表。

```
GET /openapi/kubernetes/{root_scope_name_or_id}/vulnerability_scanning/scanner
```

参数：JSON 查询正文包含以下键

名称	类型	说明
root_scope_name_or_id	字符串	根范围名称或 ID

响应对象：返回具有以下属性的注册表对象数组

属性	类型	说明
id	字符串	扫描程序的 ID
kubernetes_cluster	对象	将凭证添加到 Kubernetes 注册表
health_status	字符串	定义扫描程序运行状况
health_object	字符串	运行状况对象
scanner_action	字符串	通知扫描程序是 ENABLED 还是 DISABLED
name	字符串	扫描程序的名称

运行状况对象

属性	类型	说明
last_checkin	字符串	纪元中的上次报告时间
scanner_sensor_name	字符串	运行扫描程序的 Kubernetes 节点名称
scanner_sensor_uuid	字符串	在 Kubernetes 节点上运行的代理的 ID

属性	类型	说明
status	字符串	通知扫描程序是否报告运行状况

示例 python 代码

```
root_app_scope_name = 'Tetration'
restclient.get('/kubernetes/%s/vulnerability_scanning/scanner % root_app_scope_name)
```

编辑扫描程序过滤器查询和操作

此终端允许您编辑 Kubernetes 扫描程序过滤器查询和操作。

```
PUT /openapi/kubernetes/{root_scope_name_or_id}/vulnerability_scanning/scanner/{scanner_id}
```

参数：JSON 查询正文包含以下键

名称	类型	说明
rootAppScopeName	字符串	根范围名称或 ID
scanner_id	字符串	Kubernetes 扫描程序 ID
scanner_action	字符串	启用或禁用扫描程序。预期值为 ENABLED 或 DISABLED
filter_query	对象	验证资产过滤器查询，用于过滤用于漏洞扫描的 Pod。

示例 python 代码

```
root_app_scope_name = 'Tetration'
scanner_id = ‘64cdc7a7362f57192dcc1625’
pay_load = {
         “scanner_action” : “ENABLED”
        "filter_query": {
            "type": "contains",
            "field": "user_orchestrator_system/pod_name",
            "value": "pod"
        }
}
restclient.put('/kubernetes/%s/vulnerability_scanning/scanner/%s' % root_app_scope_name,
scanner_id, json_body=json.dumps(pay_load))
```

外部协调器的策略执行状态

这组 API 用于为负载均衡器外部协调器（例如 *F5 BIG-IP* 或 *Citrix Netscaler*）提供策略执行状态。

Note 要使用这些 API，您必须能够访问连接到 VRF 的范围。

获取所有外部协调器的策略执行状态

此终端会返回属于给定 VRF 的所有外部协调器的策略执行状态。此 API 可用于具有 `external_integration` 功能的 API 密钥。

```
GET /openapi/v1/tnp_policy_status/{vrfID}
```

参数：请求 URL 包含以下参数

名称	类型	说明
vrfID	整数	根范围的 VRF ID。

响应对象：返回状态为 `ENFORCED` 或 `FAILED` 或 `IGNORED` 的网络策略列表。

示例 python 代码

```
vrf_id = 676767
restclient.get('/tnp_policy_status/%d' % vrf_id)
```

获取外部协调器的策略执行状态

此终端会返回属于给定 VRF 的外部协调器的策略执行状态。此 API 可用于具有 `external_integration` 功能的 API 密钥。

```
GET /openapi/v1/tnp_policy_status/{vrfID}/{orchestratorID}
```

参数：请求 URL 包含以下参数

名称	类型	说明
vrfID	整数	根范围的 VRF ID。
orchestratorID	字符串	外部协调器 ID。

响应对象：返回状态为 `ENFORCED` 或 `FAILED` 或 `IGNORED` 的网络策略列表。

示例 python 代码

```
vrf_id = 676767
orchestrator_id = '5ee3c991497d4f3b00f1ee07'
restclient.get('/tnp_policy_status/%d/%s' % (vrf_id, orchestrator_id))
```

下载托管数据分流和数据接收器的证书

这组 API 用于下载托管数据分流和数据接收器的证书。

Note 要使用这些 API，您必须能够访问连接到 VRF 的范围。

获取给定 VRF ID 的托管数据分流器列表。

此终端会返回给定 VRF 中托管数据分流的列表。此 API 可用于具有 `external_integration` 功能的 API 密钥。

```
GET /openapi/v1/mdt/{vrfID}
```

参数：无

返回具有托管数据分流 ID 等属性的托管数据分流的列表。

下载给定 MDT ID 的托管数据分流证书

此终端用于下载给定托管数据分流 ID 的证书。可以通过使用 **/openapi/v1/mdt/{vrfID}** 终端来获取 MDT ID，如上述文档中所述。此 API 可用于具有 `external_integration` 功能的 API 密钥。

```
GET /openapi/v1/mdt/{vrfID}/{mdtID}/certs
```

参数：

名称	类型	说明
format	字符串	密钥库和信任存储区格式。值：`jks`（默认值）或 `cert`

返回包含以下文件的 tar.gz 文件：

对于 `jks` 格式：**truststore.jks**、**topic.txt**、**passphrase.txt**、**keystone.jks**、**kafkaBrokerIps.txt**、**consumer_name.txt**、**consumer_group_id.txt**。。

对于 `jks` 格式：**KafkaConsumerCA.cert**、**KafkaConsumerPrivateKey.key**、**kafkaCA.cert**、**kafkaBrokerIps.txt**、**topic.txt**

KafkaConsumerCA.cert 是公共证书文件，**KafkaConsumerPrivateKey.key** 文件包含私钥。**kafkaCA.cert** 包含 CA 证书，**kafkaBrokerIps.txt** 包含 Kafka 代理 IP 地址和端口列表。topic.txt 文件包含应用于从 MDT 获取数据的主题的名称。**truststore.jks** 和 **keystone.jks** 是 Java 密钥库文件。

获取给定 VRF ID 的 DataSink 列表

此终端会返回给定 VRF 中的 DataSink 列表。此 API 可用于具有 `external_integration` 功能的 API 密钥。

```
GET /openapi/v1/datasinks/{vrfID}
```

参数：无

返回包含 DataSink ID 等属性的 DataSink 列表。

下载给定 DataSink ID 的 DataSink 证书

此终端用于下载给定 DataSink ID 的证书。可以通过使用 **/openapi/v1/datasinks/{vrfID}** 终端来获取 DataSink ID，如上述文档中所述。此 API 可用于具有 `external_integration` 功能的 API 密钥。

```
GET /openapi/v1/datasinks/{vrfID}/{dsID}/certs
```

参数：无

返回包含以下文件的 tar.gz 文件：- **userCA.cert, userPrivateKey.key, intermediateCA.cert, kafkaCA.cert, kafkaBrokerIps.txt, topic.txt**。

userCA.cert 是公共证书文件，**KafkaConsumerPrivateKey.key** 文件包含私钥。**intermediateCA. cert** 和 **kafkaCA.cert** 分别包含中间 CA 和根 CA 的 CA 证书。**kafkaBrokerIps.txt** 包含 Kafka 代理 IP 地址和端口的列表。**topic.txt** 文件包含应用于从数据接收器获取数据的主题的名称。

变更日志

此 API 会提供对更改日志项的读取访问权限。此 API 需要使用与 API 密钥关联的 `user_role_scope_management` 功能。

Note 此 API 仅对网站管理员和根范围的所有者开放。

变更日志对象

变更日志对象属性的说明如下：

属性	类型	说明
id	字符串	更改日志项的唯一标识符。
association_chain	对象数组	与此更改关联的名称和 ID 的列表。

属性	类型	说明
scope	字符串	变更范围（不同于 Cisco Secure Workload 范围）。
action	字符串	更改操作。
details	字符串	更多操作详细信息（如果可用）。
created_at	整数	创建变更日志项时的 Unix 时间戳。
modifier	对象	负责更改的用户。
modified	对象	修改的字段和值。
original	对象	修改前的字段和值。
version	整数	版本标识符。

搜索

此终端会返回与指定条件匹配的变更日志项目列表。

```
GET /openapi/v1/change_logs
```

参数：请求 URL 包含以下参数

名称	类型	说明
root_app_scope_id	字符串	（可选）对于根范围所有者为必填。按根范围过滤结果。
association_name	字符串	（可选）对于根范围所有者为必填。要返回的项目类型。例如：“H4Users”
history_action	字符串	（可选）更改操作。例如：“update”
details	字符串	（可选）操作详细信息。例如：“soft-delete”
before_epoch	整数	（可选）包括在此 Unix 时间戳之前创建的结果。
after_epoch	整数	（可选）包括在此 Unix 时间戳之后创建的结果。

名称	类型	说明
offset	整数	（可选）要跳过的结果数。
limit	整数	（可选）限制结果数量，默认为1000。

响应对象：返回变更日志对象列表。

响应

响应是正文中具有以下属性的 JSON 对象。

名称	类型	说明
total_count	整数	在应用偏移或限制之前匹配的项目总数。
items	对象数组	结果列表。

示例 python 代码

获取给定根范围在过去一天内最近 100 次范围对象更改。

```
root_app_scope_id = '5ce480db497d4f1ca1fc2b2b'
one_day_ago = int(time.time() - 24*60*60)
resp = restclient.get('/change_logs', params={'root_app_scope_id': root_app_scope_id,
                                              'association_name': 'AppScope',
                                              'after_epoch': one_day_ago,
                                              'limit': 100})
```

获取第二千个范围对象更改。

```
root_app_scope_id = '5ce480db497d4f1ca1fc2b2b'
resp = restclient.get('/change_logs', params={'root_app_scope_id': root_app_scope_id,
                                              'association_name': 'AppScope',
                                              'offset': 1000})
```

进一步细化这些结果，以仅显示新的范围创建。

```
root_app_scope_id = '5ce480db497d4f1ca1fc2b2b'
one_day_ago = int(time.time() - 24 * 60 * 60)
resp = restclient.get('/change_logs', params={'root_app_scope_id': root_app_scope_id,
                                              'association_name': 'AppScope',
                                              'history_action': 'create',
                                              'after_epoch': one_day_ago,
                                              'limit': 100})
```

站点管理员可以使用 limit 和 offset 来迭代获取所有范围内的所有更改。

```
resp = restclient.get('/change_logs', params={'offset': 100, 'limit': 100})
```

不可路由终端

以下 API 用于管理不可路由终端，将 IP 或子网标记为不可路由，或获取用户标记的不可路由终端的列表，或者将 IP 或子网取消标记为不可路由终端。需要使用与 API 密钥关联的 `user_data_upload` 功能。

不可路由终端对象

不可路由终端对象属性的说明如下：

属性	类型	说明
id	字符串	不可路由终端的唯一标识符。
name	字符串	用户指定的不可路由终端的名称。
subnet	字符串	IPv4/IPv6 子网。
vrf_id	长度	不可路由终端所属的 VRF 的 ID。
address_type	字符串	基于子网地址类型的 IPV4/IPV6
host_uuid	字符串	代理的唯一 ID
description	字符串	用户指定的不可路由终端的说明。

GET 不可路由终端

此终端会返回给定租户中不可路由终端列表。

```
GET /openapi/v1/non_routable_endpoints/{rootScopeName}
```

参数：无

创建不可路由终端

此终端用于创建不可路由终端。

```
POST /openapi/v1/non_routable_endpoints/{rootScopeName}
```

参数：

属性	类型	说明
name	字符串	用户指定的不可路由终端的名称。
subnet	字符串	IPv4 或 IPv6 子网。
address_type(optional)	字符串	IPv4 或 IPv6（基于子网地址类型）
host_uuid(optional)	字符串	代理的唯一 ID
description(optional)	字符串	用户指定的不可路由终端的说明。

*如果未指定可选字段，则填充空值。

示例 python 代码

```
  req_payload = {
    "name": "nre-1",
    "subnet": "1.1.1.1/30",
    "address_type": IPV4,
    "description": "sample parameters test"
  }
  resp = restclient.post('/openapi/v1/non_routable_endpoints/Default',
json_body=json.dumps(req_payload))
```

获取具有名称的特定不可路由终端

此终端会返回指定名称的不可路由终端。

```
GET /openapi/v1/non_routable_endpoints/{rootScopeName}/name/{name}
```

参数：无

获取具有 ID 的特定不可路由终端

此终端会返回指定 ID 的不可路由终端。

```
GET /openapi/v1/non_routable_endpoints/{rootScopeName}/id/{id}
```

参数：无

更新特定的不可路由终端名称

此终端用于更新不可路由终端。它使用现有不可路由终端的 ID 或名称来更新其名称。

```
PUT /openapi/v1/non_routable_endpoints/{rootScopeName}
```

参数：

属性	类型	说明
id	字符串	不可路由终端的唯一标识符。
name	字符串	用户指定的不可路由终端的名称。
new_name	字符串	要更新的新名称

示例 python 代码

```
  req_payload = {
    "name": "nre-1",
    "new_name": "nre-updated",
  }
  resp = restclient.put('/openapi/v1/non_routable_endpoints/Default',
json_body=json.dumps(req_payload))


  req_payload = {
    "id": "5f706964a5b5f16ed4b0aacb",
    "new_name": "nre-updated",
  }
  resp = restclient.put('/openapi/v1/non_routable_endpoints/Default',
json_body=json.dumps(req_payload))
```

删除具有名称的特定不可路由终端

此终端将删除特定的不可路由终端。

```
DELETE /openapi/v1/non_routable_endpoints/{rootScopeName}/name/{name}
```

删除具有 ID 的特定不可路由终端

此终端将删除特定的不可路由终端。

```
DELETE /openapi/v1/non_routable_endpoints/{rootScopeName}/id/{id}
```

外部设备和连接器的配置和命令架构

配置组 API

配置组 API 可为设备和连接器 API 提供配置模式。这些 API 需要使用与 API 密钥关联的 `sensor_management` 或 `external_integration` 功能。

用于获取配置架构的 API

此终端会返回所选类型/配置组的静态配置架构。

```
GET /openapi/v1/config_groups/schema/<type>
```

其中 <type> 是设备配置类型。

参数：请求 URL 包含以下参数

名称	类型	说明
type	字符串	从 "VM1" "VM3" "NTP" "LOG" "LDAP" "NETFLOW" "IPFIX" "NETSCALER" "F5" "AWS" "ENDPOINT" "SLACK_NOTIFIER" "GCP_CONNECTOR" "PAGERDUTY_NOTIFIER" "SYSLOG_NOTIFIER" "KINESIS_NOTIFIER" "EMAIL_NOTIFIER" "ISE" "MERAKI" "SLACK_NOTIFIER_OVERRIDE" "PAGERDUTY_NOTIFIER_OVERRIDE" "SYSLOG_NOTIFER_OVERRIDE" "KINESIS_NOTIFER_OVERRIDE" "AZURE_CONNECTOR" "EMAIL_NOTIFIER_OVERRIDE" "SYSLOG_SEVERITY_MAPPING" "SERVICENOW" "SYNC_INTERVAL" "ALERT" "VM3_ERSPAN" "AWS_CONNECTOR" "VM0" 中指定配置类型

响应对象：返回所选配置类型的配置架构。

响应示例

```
resp = restclient.get('/config_groups/schema/LOG')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "type": "LOG",
    "name": "Log",
    "mode": "TEST",
    "config": {
      "secured": {},
      "unsecured": {
        "log-level": "info",
        "max-log-size": 10,
        "max-log-age": 30,
        "max-log-backups": 20 }
    },
    "fill_ins": [
      {
        "field": "log-level",
        "label": "Logging Level",
        "placeholder": "info",
        "type": "user_fill_in",
        "input_type": "dropdown",
        "possible_values": [
          "trace",
          "debug",
          "info",
          "warn",
          "error"
        ]
    },
      {
```

```
        "field": "max-log-size",
        "label": "Max Log File Size (in MB)",
        "placeholder": 10,
        "type": "user_fill_in",
        "input_type": "number",
        "min": 1,
        "max": 10240
      },
      {"field": "max-log-age",
        "label": "Log Rotation (in days)",
        "placeholder": 30,
        "type": "user_fill_in",
        "input_type": "number",
        "min": 1,
        "max": 365
      },
      {
        "field": "max-log-backups",
        "label": "Log Rotation (in instances)",
        "placeholder": 20,
        "type": "user_fill_in",
        "input_type": "number",
        "min": 1,
        "max": 100
}
    ]
  }
```

用于获取故障排除命令架构的 API

此终端会为选定的故障排除命令类型返回静态故障排除命令配置架构。

```
GET /openapi/v1/config_groups/command_schema/<type>
```

在请求负载中，<type> 是故障排除命令配置类型。

参数：

请求 URL 包含以下参数

名称	类型	说明
type	字符串	从 "SHOW LOG" "SHOW_SERVICE_LOG" "SHOW_RUNNING_CONF" "SHOW_SERVICE_RUNNING_CONF" "SHOW_SYS_COMMANDS" "SHOW_DOCKER_COMMANDS" "SHOW_DOCKER_INSTANCE_COMMANDS" "OPER_DOCKER_INSTANCE_COMMANDS" "SHOW_SUPERVISOR_COMMANDS" "SHOW_SUPERVISOR_SERVICE_COMMANDS" "OPER_SUPERVISOR_SERVICE_COMMANDS" "NETWORK_CONNECTIVITY_COMMANDS" "LIST_FILES" "LIST_SERVICE_FILES" "PACKET_CAPTURE" "SHOW_DATA_EXPORT_LGO" "SHOW_DATAEXPORT_RUNNING_CONF" "SHOW_DATA_EXPORT_SYS_COMMANDS" "UPDATE_LISTENING_PORT" "UPDATE_TAN_LOG_CONF" "SNAPSHOT_APPLIANCE" "SNAPSHOT_CONNECTOR" "SHOW_AWS_DOWNLOADER_LOG" "CONTROLLER_PROFILING" "SERVICE_PROFILING" "RESTART_CONNECTOR_CONTAINER" "RESTART_CONNECTOR_SERVICE" "CONNECTOR_ALERT_INTERVAL_APPLIANCE" "CONNECTOR_ALERT_INTERVAL_CONNECTOR" "EXEC_SCRIPT" "SHOW_SEGMENTATION_POLICIES" 中指定命令类型

响应对象：返回所选配置类型的配置架构。

响应示例

```
resp = restclient.get('/config_groups/command_schema/SHOW_LOG')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "name": "Show logs",
    "desc": "Show the contents of a log file",
   "long_desc": "Show the contents of a log file and optionally grep the file for a specified
 pattern. The output is tailed for the last 5000 lines.",
    "valid_appliances": [
      "TETRATION_DATA_INGEST",
      "TETRATION_EDGE",
      "TETRATION_EXPORT"
    ],
    "valid_connectors": [
      "netflow",
```

```
      "netscaler",
      "f5",
      "aws",
      "anyconnect",
      "slack",
      "kinesis",
      "syslog",
      "email",
      "pagerduty",
      "ise",
      "asa",
      "meraki",
      "servicenow",
      "wad"
    ],
    "arg_fillins": [
      {
        "field": "pattern",
        "label": "Grep Pattern",
        "input_type": "text",
        "optional": true
      }
    ],
    "output_type": "FILE",
    "output_ext": "LOG"
  }
```

外部设备

外部设备 API

外部设备 API 与管理 SecureWorkload 外部设备相关联。这些 API 集需要使用与 API 密钥关联的 `sensor_management` 或 `external_integration` 功能。

用于获取设备列表的 API

此终端会返回设备列表。

```
GET /openapi/v1/ext_appliances?root_scope_id=<root_scope_id>&type=<type>
```

其中 <root_scope_id> 是可从 获取范围 API 获取的 root_ scope_id，而 <type> 是用于决定设备类型的字符串。

参数：请求 URL 包含以下参数

名称	类型	说明
root_scope_id	字符串	指定根范围
type	字符串	指定设备类型，值可以是“TETRATION_EDGE”、“TETRATION_DATA_INGEST”、“TETRATION_EXPORT”、“TETRATION_ERSPAN”或“TETRATION_INTERNAL”

响应对象：返回设备列表。

响应示例

```
resp =
restclient.get('/ext_appliances?root_scope_id=63bf8d2f497d4f7287dbd335&type=TETRATION_INTERNAL')

  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
[
    {
      "root_scope_id": "63bf8d2f497d4f7287dbd335",
      "type": "tetration_internal",
      "status": {
        "state": "active",
        "controller_state": "up",
        "message": "",
        "display_state": "active"
      },
      "auto_upgrade": true,
      "created_at": 1673498141,
      "updated_at": 1673498141,
      "registered_at": 1673498141,
      "last_checkin_at": 0,
      "last_rpm_sent_at": 0,
      "upgrade_attempts": 0,
      "delete_attempts": 0,
      "last_delete_msg_sent_at": 0,
      "taas": false,
      "deleted": false,
      "deleted_at": 0,
      "connector_limit": 5000,
      "available_slots": 5000,
      "internal": true,
      "id": "63bf8e1d6419d06bef39bc85",
      "ha_peer_appliance_id": "",
      "display_type": "Tetration Internal"
    }
  ]
```

用于创建设备的 API

此终端会创建一个设备。

```
POST /openapi/v1/ext_appliances
```

在请求负载中，要获取 <config> 架构，从 用于获取设备架构的 API，第 193 页 响应中选择“valid_config”之一，将“valid_config”应用于 用于获取配置架构的 API，第 179 页。

参数：请求 URL 包含以下参数

名称	类型	说明
name	字符串	指定名称
root_scope_id	字符串	指定根范围

名称	类型	说明
type	字符串	指定设备类型，值可以是“TETRATION_EDGE”、“TETRATION_DATA_INGEST”、“TETRATION_EXPORT”、“TETRATION_ERSPAN”或“TETRATION_INTERNAL”
config	set	以 JSON 格式提供填充的配置架构
taas	布尔值	指明设备是否适用于TAAS环境
version	字符串	指定版本

响应对象：返回创建的设备。

响应示例

```
req_payload = {
    "name":"Data Ingest Appliance",
    "type":"tetration_data_ingest",
    "root_scope_id":"63c41ff2497d4f5f5be73662",
    "config":{
      "VM3":{
        "secured":{},
        "unsecured":{
          "cidr":[
            "172.26.231.141/23",
            "172.26.231.142/23",
            "172.26.231.143/23"
          ],
          "gateway":[
            "172.26.231.140",
            "172.26.231.140",
            "172.26.231.140"
          ],
          "cidr_v6":[],
          "gateway_v6":[],
          "dns":[
            "testserver"
          ],
          "search_domains":[],
          "hostname":"",
          "use_proxy_for_tetration":false,
          "https_proxy":"",
          "no_proxy":[],
          "docker_subnet":""
        }
      }
    }
  }
  resp = restclient.post('/ext_appliances', json_body=json.dumps(req_payload))
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
   "root_scope_id":"63c41ff2497d4f5f5be73662",
   "type":"tetration_data_ingest",
   "status":{
     "state":"pending_dio",
     "controller_state":"down",
     "message":"Setting up appliance",
     "display_state":"preparing"
   },
   "auto_upgrade":true,
   "created_at":1674183549,
   "updated_at":1674183549,
   "registered_at":0,
   "last_checkin_at":0,
   "last_rpm_sent_at":0,
   "upgrade_attempts":0,
   "delete_attempts":0,
   "last_delete_msg_sent_at":0,
   "name":"Data Ingest Appliance",
   "taas":false,
   "deleted":false,
   "deleted_at":0,
   "connector_limit":3,
   "available_slots":0,
   "internal":false,
   "id":"63ca037dbca44e263daeb5d0",
   "ha_peer_appliance_id":"",
   "display_type":"Tetration Data Ingest"
 }
```


用于删除设备的 API

此终端将删除给定设备。

```
DELETE /openapi/v1/ext_appliances/<id>
```

其中 <id> 是可以从 用于获取设备列表的 API，第 183 页 获取的 appliance_id。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定设备 ID

响应对象：返回已删除设备的状态。

响应示例

```
resp = restclient.delete('/ext_appliances/63be3b1ade36423c12bff6e1')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例


```
 {
    "status": 200,
    "code": 1000,
    "message": "deleted"
  }
```

按 ID 获取设备的 API

此终端会获取具有给定 ID 的设备。

```
GET /openapi/v1/ext_appliances/<id>
```

其中 <id> 是可以从 用于获取设备列表的 API，第 183 页 获取的 appliance_id。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定设备 ID

响应对象：返回具有给定 ID 的设备。

响应示例

```
resp = restclient.get('/ext_appliances/63bf8e1d6419d06bef39bc85')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "root_scope_id": "63bf8d2f497d4f7287dbd335",
    "type": "tetration_internal",
    "status": {
      "state": "active",
      "controller_state": "up",
      "message": "",
      "display_state": "active"
    },
    "auto_upgrade": true,
    "created_at": 1673498141,
    "updated_at": 1673498141,
    "registered_at": 1673498141,
    "last_checkin_at": 0,
    "last_rpm_sent_at": 0,
    "upgrade_attempts": 0,
    "delete_attempts": 0,
    "last_delete_msg_sent_at": 0,
    "taas": false,
    "deleted": false,
    "deleted_at": 0,
    "connector_limit": 5000,
    "available_slots": 5000,
    "internal": true,
    "id": "63bf8e1d6419d06bef39bc85",
    "ha_peer_appliance_id": "",
    "display_type": "Tetration Internal"
  }
```

用于重命名设备的 API

此终端会重命名具有给定 ID 的设备。

```
PUT /openapi/v1/ext_appliances/<id>
```

其中 <id> 是可以从 用于获取设备列表的 API，第 183 页 获取的 appliance_id。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定设备 ID
name	字符串	指定设备的新名称

响应对象：返回具有给定 ID 和新名称的设备。

响应示例

```
req_payload = {
    "name": "new_name",
  }
  resp = restclient.put('/ext_appliances/63bf8e1d6419d06bef39bc85',
json_body=json.dumps(req_payload))
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "root_scope_id": "63bf8d2f497d4f7287dbd335",
    "type": "tetration_internal",
    "status": {
      "state": "active",
      "controller_state": "up",
      "message": "",
      "display_state": "active"
    },
    "auto_upgrade": true,
    "created_at": 1673498141,
    "updated_at": 1673498141,
    "registered_at": 1673498141,
    "last_checkin_at": 0,
    "last_rpm_sent_at": 0,
    "upgrade_attempts": 0,
     "delete_attempts": 0,
    "last_delete_msg_sent_at": 0,
    "name": "new_name",
    "taas": false,
    "deleted": false,
    "deleted_at": 0,
    "connector_limit": 5000,
    "available_slots": 5000,
    "internal": true,
    "id": "63bf8e1d6419d06bef39bc85",
    "ha_peer_appliance_id": "",
    "display_type": "Tetration Internal"
  }
```

用于获取有关配置类型的配置的 API

此终端会获取具有给定 ID 和配置类型的设备的配置。

```
GET /openapi/v1/ext_appliances/<id>/config?config_type=<config_type>
```

其中 <id> 是可以从 用于获取设备列表的 API，第 183 页 获取的 appliance_id，而 <config_type> 是可从 用于获取设备架构的 API，第 193 页 获取的“valid_config”。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定设备 ID
config_type	字符串	指定配置类型。有关说明下列出的所有可能值，请参阅用于获取配置架构的 API，第 179 页

响应对象：返回具有给定设备 ID 和配置类型的配置

响应示例

```
resp =
restclient.get('/ext_appliances/63c1272039042a1c0ddd3e20/config?config_type=VM3_ERSPAN')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例


```
[
    {
      "mode": "TEST_AND_APPLY",
      "name": "VM3_ERSPAN",
      "root_scope_id": "63bf8d2f497d4f7287dbd335",
      "vrf_id": 1,
      "appliance_id": "63c1272039042a1c0ddd3e20",
      "deleted": false,
      "type": "VM3_ERSPAN",
      "state": "COMMIT",
      "attempts": 0,
      "config": {
        "secured": {},
        "unsecured": {
          "cidr": [
            "172.26.231.141/23",
            "172.26.231.142/23",
            "172.26.231.143/23"
          ],
          "gateway": [
            "172.26.231.140",
            "172.26.231.140",
            "172.26.231.140"
          ],
          "cidr_v6": [],
          "gateway_v6": [],
          "dns": [
            "test"
          ],
          "search_domains": [],
          "hostname": "hjtest",
          "https_proxy": "",
          "no_proxy": []
        }
      },
      "push_to_dio_at": 0,
      "created_at": 1673602848,
      "updated_at": 1673602848,
      "discovery_completed_at": 0,
      "committed_at": 0,
      "delete_at": 0,
      "error_at": 0,
```

```
    "hidden": false,
    "discovery_phase": 0,
    "internal": false,
    "id": "63c1272039042a1c0ddd3e21"
  }
]
```

用于将新配置添加到外部设备的 API

此终端可向具有给定 ID 的设备添加新配置

```
POST /openapi/v1/ext_appliances/<id>/config
```

其中 <id> 是可以从用于获取设备列表的 API，第 183 页 获取的 appliance_id。在请求负载中，<type> 是可从 用于获取设备架构的 API，第 193 页 获取的“valid_config”。要获取 <config> 架构，从 用于获取设备架构的 API，第 193 页 响应中选择“valid_config”之一，将“valid_config”应用于 用于获取配置架构的 API，第 179 页。

参数：请求 URL 包含以下参数

名称	类型	说明
name	字符串	指定配置名称
type	字符串	指定配置类型。有关说明下列出的所有可能值，请参阅用于获取配置架构的 API，第 179 页
config	set	以 JSON 格式提供填充的配置架构

响应对象：返回更新后的配置。

响应示例

```
req_payload = {
    "name": "new_config",
    "type": "VM3_ERSPAN",
    "config": {}
  }
  resp = restclient.post('/ext_appliances/63c1272039042a1c0ddd3e20/config',
json_body=json.dumps(req_payload))
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "prev_id": "63c1272039042a1c0ddd3e21",
    "mode": "TEST_AND_APPLY",
    "name": "new_config",
    "root_scope_id": "63bf8d2f497d4f7287dbd335",
    "vrf_id": 1,
    "appliance_id": "63c1272039042a1c0ddd3e20",
    "deleted": false,
    "type": "VM3_ERSPAN",
    "state": "COMMIT",
    "attempts": 0,
    "config": {
```

```
    "secured": {},
    "unsecured": null
  },
  "push_to_dio_at": 0,
  "created_at": 1673661042,
  "updated_at": 1673661042,
  "discovery_completed_at": 0,
  "committed_at": 0,
  "delete_at": 0,
  "error_at": 0,
  "hidden": false,
  "discovery_phase": 0,
  "internal": false,
  "id": "63c20a7239042a0991b871b7"
}
```

用于删除配置的 API

此终端会从给定设备中删除配置。

```
DELETE /openapi/v1/ext_appliances/<id>/config/<config_id>
```

其中 <id> 是可以从用于获取设备列表的 API，第 183 页获取的 appliance_id，而 <config_id> 是可从用于获取有关配置类型的配置的 API，第 188 页获取的 ID。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定设备 ID
config_id	字符串	指定配置 ID

响应对象：返回给定设备的已删除配置状态。

响应示例

```
resp =
restclient.delete('/ext_appliances/63c1272039042a1c0ddd3e20/config/63c1272039042a1c0ddd3e21')

  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "status": 200,
    "code": 1000,
    "message": "deleted"
  }
```

用于获取配置的 API

此终端获取具有给定 ID 的配置

```
GET /openapi/v1/ext_appliances/<id>/config/<config_id>
```

其中 <id> 是可以从用于获取设备列表的 API，第 183 页获取的 appliance_id，而 <config_id> 是可从用于获取有关配置类型的配置的 API，第 188 页获取的 ID。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定设备 ID
config_id	字符串	指定配置 ID

响应对象：返回具有给定 ID 的配置。

响应示例

```
resp =
restclient.get('/ext_appliances/63c1272039042a1c0ddd3e20/config/63c1272039042a1c0ddd3e21')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
b
```

响应示例

```
{
    "mode": "TEST_AND_APPLY",
    "name": "VM3_ERSPAN",
    "root_scope_id": "63bf8d2f497d4f7287dbd335",
    "vrf_id": 1,
    "appliance_id": "63c1272039042a1c0ddd3e20",
    "deleted": false,
    "type": "VM3_ERSPAN",
    "state": "COMMIT",
    "attempts": 0,
    "config": {
      "secured": {},
      "unsecured": {
        "cidr": [
          "172.26.231.141/23",
          "172.26.231.142/23",
          "172.26.231.143/23"
        ],
        "gateway": [
          "172.26.231.140",
          "172.26.231.140",
          "172.26.231.140"
        ],
        "cidr_v6": [],
        "gateway_v6": [],
        "dns": [
          "test"
        ],
        "search_domains": [],
        "hostname": "hjtest",
        "https_proxy": "",
        "no_proxy": []
      }
    },
    "push_to_dio_at": 0,
    "created_at": 1673602848,
    "updated_at": 1673602848,
    "discovery_completed_at": 0,
    "committed_at": 0,
    "delete_at": 0,
    "error_at": 0,
    "hidden": false,
```

```
    "discovery_phase": 0,
    "internal": false,
    "id": "63c1272039042a1c0ddd3e21"
  }
```

用于获取设备架构的 API

此终端会获取具有给定类型的设备架构

```
GET /openapi/v1/ext_appliances/schema/<type>
```

其中 <type> 是用于决定设备类型的字符串。

参数：请求 URL 包含以下参数

名称	类型	说明
type	字符串	指定设备类型，值可以是“TETRATION_EDGE”、“TETRATION_DATA_INGEST”、“TETRATION_EXPORT”、“TETRATION_ERSPAN”或“TETRATION_INTERNAL”

响应对象：返回配置架构。

响应示例

```
resp = restclient.get('/ext_appliances/schema/TETRATION_ERSPAN')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
 {
    "name": "Data Ingest for ERSPAN",
    "type": "tetration_erspan",
    "desc": "Data Ingest Appliance for ERSPAN",
    "long_desc": "Data Ingest appliance for ERSPAN is a software appliance that can export
 flow data from ERSPAN appliance.",
    "valid_config": [
      "VM3_ERSPAN"
    ],
    "deploy_config": [
      "VM3_ERSPAN"
    ],
    "connectors": [
      "ERSPAN"
    ],
    "limit_connectors_per_appliance": 0,
    "limit_per_rootscope": 8,
    "limit_per_rootscope_taas": 4,
    "limit_per_cluster": 150,
    "cco_url":
"https://software.cisco.com/download/home/286309796/type/286309874/release/3.7.1.26.devel"
  }
```

用于列出可用于设备的故障排除命令的 API

此终端会返回可用于给定设备的故障排除命令列表。

```
GET /openapi/v1/ext_appliances/<id>/commands/available
```

其中 <id> 是可以从 用于获取设备列表的 API，第 183 页 获取的 appliance_id。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定设备 ID

响应对象：返回可用于给定设备的故障排除命令列表。

响应示例

```
resp = restclient.get('/ext_appliances/63c6ef42bca44e2b5e729191/commands/available')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
[
    {
      "type": "UPDATE_LISTENING_PORT",
      "name": "Update the listening port on a connector"
    },
    {
      "type": "SNAPSHOT_APPLIANCE",
      "name": "Collect Snapshot from Appliance"
    },
    {
      "type": "LIST_FILES",
      "name": "List a directory"
    },
    {
      "type": "CONTROLLER_PROFILING",
      "name": "Collect controller profile"
    },
    {
      "type": "SHOW_LOG",
      "name": "Show logs"
    },
    {
      "type": "SHOW_SUPERVISOR_COMMANDS",
      "name": "Execute supervisorctl command"
    },
    {
      "type": "PACKET_CAPTURE",
      "name": "Packet capture"
    },
    {
      "type": "NETWORK_CONNECTIVITY_COMMANDS",
      "name": "Test network connectivity"
    },
    {
      "type": "SHOW_DOCKER_COMMANDS",
      "name": "Execute docker command"
    },
    {
```

```
    "type": "CONNECTOR_ALERT_INTERVAL_APPLIANCE",
    "name": "Override connector alert interval for Appliance"
  },
  {
    "type": "SHOW_RUNNING_CONF",
    "name": "Show running configuration"
  },
  {
    "type": "SHOW_SYS_COMMANDS",
    "name": "Execute system command"
  }
]
```

用于列出故障排除命令的 API

此终端会返回为给定设备启用的故障排除命令的列表。

```
GET /openapi/v1/ext_appliances/<id>/commands
```

其中 <id> 是可以从 用于获取设备列表的 API，第 183 页 获取的 appliance_id。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定设备 ID

响应对象：返回为给定设备启用的故障排除命令的列表。

响应示例

```
resp = restclient.get('/ext_appliances/63be3b1ade36423c12bff6e1/commands')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
[
    {
      "appliance_id": "63be3b1ade36423c12bff6e1",
      "state": "pending",
      "level": "APPLIANCE",
      "command": "SHOW_LOG",
      "arg_string": "",
      "args": {},
      "tailed": false,
      "rc": 0,
      "push_to_dio_at": 0,
      "attempts": 0,
      "deleted": false,
      "deleted_at": 0,
      "created_at": 1673595392,
      "total_chunk": 0,
      "id": "63c10a0039042a6aee1b008c"
    }
  ]
```

用于创建故障排除命令的 API

此终端创建可用于给定设备的故障排除命令。

```
POST /openapi/v1/ext_appliances/<id>/commands
```

其中 <id> 是可以从 用于获取设备列表的 API，第 183 页 获取的 appliance_id。在请求负载中，<command> 是故障排除命令类型，可从 用于获取故障排除命令架构的 API，第 181 页 响应“valid_appliances”字段中获取。<arguments> 是命令架构的填充 JSON 对象，可从用于获取故障排除命令架构的 API，第 181 页中获取。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定设备 ID
command	字符串	指定命令类型
arguments	set	以 JSON 格式提供填充的命令架构

响应对象：返回为给定设备创建的故障排除命令。

响应示例

```
req_payload = {
    "command": "SHOW_LOG",
    "arguments": {}
  }
  resp = restclient.post('/ext_appliances/63be3b1ade36423c12bff6e1/commands',
json_body=json.dumps(req_payload))
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
 {
    "appliance_id": "63be3b1ade36423c12bff6e1",
    "state": "pending",
    "level": "APPLIANCE",
    "command": "SHOW_LOG",
    "args": {},
    "tailed": false,
    "rc": 0,
    "push_to_dio_at": 0,
    "attempts": 0,
    "deleted": false,
    "deleted_at": 0,
    "created_at": 1673595392,
    "total_chunk": 0,
    "id": "63c10a0039042a6aee1b008c"
  }
```

-

用于删除故障排除命令的 API

此终端删除为给定设备启用的故障排除命令。

```
DELETE /openapi/v1/ext_appliances/<id>/commands/<command_id>
```

其中 <id> 是可以从 用于获取设备列表的 API，第 183 页 获取的 appliance_id，而 <command_id> 是可从 用于列出故障排除命令的 API，第 195 页 获取的 ID。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定设备 ID
command_id	字符串	指定命令 ID

响应对象：返回为给定设备删除的故障排除命令的状态。

响应示例

```
resp =
restclient.delete('/ext_appliances/63be3b1ade36423c12bff6e1/commands/63c10a0039042a6aee1b008c')

  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
   "status": 200,
   "code": 1000,
   "message": "deleted"
 }
```

用于返回故障排除命令的 API

此终端会返回为给定设备选择的故障排除命令。

```
GET /openapi/v1/ext_appliances/<id>/commands/<command_id>
```

其中 <id> 是从 用于获取设备列表的 API，第 183 页 获取的 appliance_id，而 <command_id> 是从 用于列出故障排除命令的 API，第 195 页 获取的 ID。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定设备 ID
command_id	字符串	指定命令 ID

响应对象：返回为给定设备选择的故障排除命令。

响应示例

```
resp =
restclient.get('/ext_appliances/63be3b1ade36423c12bff6e1/commands/63c10fd139042a1c0ddd3e1f')

  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "appliance_id": "63be3b1ade36423c12bff6e1",
    "state": "pending",
    "level": "APPLIANCE",
    "command": "SHOW_LOG",
    "arg_string": "",
    "args": {},
    "tailed": false,
    "rc": 0,
    "push_to_dio_at": 0,
    "attempts": 0,
    "deleted": false,
    "deleted_at": 0,
    "created_at": 1673596881,
    "total_chunk": 0,
    "id": "63c10fd139042a1c0ddd3e1f"
  }
```


用于将设备命令的输出下载为文件的 API

此终端会将命令的输出下载为文件。

```
GET /openapi/v1/appliances/<id>/commands/{command_id}/download
```

其中 <id> 是可以从 用于获取设备列表的 API，第 183 页 获取的 appliance_id，而 <command_id> 是可从 用于列出故障排除命令的 API，第 195 页 获取的 ID。并非所有命令都有可下载的输出，请检查 用于获取故障排除命令架构的 API，第 181 页 提供的命令架构，其中 "output_type": "FILE" 表示其具有可下载内容，"output_ext" 表示文件类型。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定设备 ID
command_id	字符串	指定命令 ID

响应对象：将命令输出作为文件下载。

响应示例

```
resp = restclient.download('downloadFile',
'/appliances/63c6ef42bca44e2b5e729191/commands/63cace941a49bd4c0e0cf45a/download')
```

连接器

连接器 API

连接器 API 与管理 Cisco Secure Workload Connector 相关联。这些 API 集需要使用与 API 密钥关联的 `sensor_management` 或 `external_integration` 功能。

用于获取所有类型的连接器的 API

此终端节点获取给定根范围的所有类型的连接器。

```
GET /openapi/v1/connectors/cards?root_scope_id=<root_scope_id>
```

其中 <root_scope_id> 是可从 获取范围，第 57 页 API 获取的 root_ scope_id。

参数：请求 URL 包含以下参数

名称	类型	说明
root_scope_id	字符串	指定根范围

响应对象：返回具有给定根范围 ID 的所有类型的连接器。

响应示例

```
resp = restclient.get('/connectors/cards?root_scope_id=63bf8d2f497d4f7287dbd335')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
[{
    "type": "NETFLOW",
    "name": "NetFlow",
    "desc": "Collect telemetry from network devices",
    "long_desc": "Collect NetFlow V9 and/or IPFIX telemetry from network devices such as
routers and switches.",
    "group": "Flow Ingest",
    "index": 0,
    "appliance_type": "tetration_data_ingest",
    "state": "disabled",
    "limit_per_appliance": 3,
    "limit_per_rootscope": 10,
    "limit_per_cluster": 100,
    "config": [
      "LOG",
      "ALERT"
    ],
    "max_instances": 0,
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "Flow Visibility"
    ],
    "connector_count": 0,
    "group_order": 0
  },
  {
    "type": "NETSCALER",
    "name": "NetScaler",
    "desc": "Collect telemetry from Citrix ADC",
   "long_desc": "Collect telemetry from Citrix ADC, stitch client and server side flows.",

    "group": "Flow Ingest",
    "index": 2,
    "appliance_type": "tetration_data_ingest",
    "state": "disabled",
    "limit_per_appliance": 3,
    "limit_per_rootscope": 10,
    "limit_per_cluster": 100,
    "config": [
      "LOG",
      "ALERT"
    ],
    "max_instances": 0,
```

```
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "Flow Visibility",
      "Flow Stitching",
      "ADM"
    ],
    "connector_count": 0,
    "group_order": 0
  },
  {
    "type": "F5",
    "name": "F5",
    "desc": "Collect telemetry from F5 BIG-IP",
    "long_desc": "Collect telemetry from F5 BIG-IP, stitch client and server side flows,
enrich client inventory with user attributes.",
    "group": "Flow Ingest",
    "index": 1,
    "appliance_type": "tetration_data_ingest",
    "state": "disabled",
    "limit_per_appliance": 3,
    "limit_per_rootscope": 10,
    "limit_per_cluster": 100,
    "config": [
      "LDAP",
      "LOG",
      "ALERT"
    ],
    "max_instances": 0,
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "Flow Visibility",
      "User Insights",
      "Flow Stitching",
      "ADM"
    ],
    "connector_count": 0,
    "group_order": 0
  },
  {
    "type": "ANYCONNECT",
    "name": "AnyConnect",
    "desc": "Collect telemetry from AnyConnect NVM",
    "long_desc": "Collect telemetry data from Cisco AnyConnect Network Visibility Module
(NVM) and enrich endpoint inventories with user attributes",
    "group": "Endpoints",
    "index": 0,
    "appliance_type": "tetration_data_ingest",
    "state": "disabled",
    "limit_per_appliance": 1,
    "limit_per_rootscope": 50,
    "limit_per_cluster": 500,
    "config": [
      "ENDPOINT",
      "LDAP",
      "LOG",
      "ALERT"
    ],
    "max_instances": 0,
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "Flow Visibility",
```

```
      "Process Annotation",
      "User Insights",
      "Endpoint Insights",
      "Inventory Enrichment"
    ],
    "connector_count": 0,
    "group_order": 1
  },
  {
    "type": "ASA",
    "name": "Cisco Secure Firewall",
    "desc": "Collect telemetry from Cisco Secure Firewall",
    "long_desc": "Collect telemetry from Cisco Secure Firewall, stitch client and server
side flows. Recommended usage with ISE connector for flow visibility.",
    "group": "Flow Ingest",
    "index": 3,
    "appliance_type": "tetration_data_ingest",
    "state": "disabled",
    "limit_per_appliance": 1,
    "limit_per_rootscope": 10,
    "limit_per_cluster": 100,
    "config": [
      "LOG",
      "ALERT"
    ],
    "max_instances": 0,
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "Flow Visibility",
      "Flow Stitching",
      "ADM"
    ],
    "connector_count": 0,
    "group_order": 0
  },
  {
    "type": "SLACK",
    "name": "Slack",
    "desc": "Send alerts to Slack",
    "long_desc": "Send Tetration Alerts to Slack.",
    "group": "Alerts",
    "index": 2,
    "appliance_type": "tetration_edge",
    "state": "disabled",
    "limit_per_appliance": 1,
    "limit_per_rootscope": 1,
    "limit_per_cluster": 150,
    "config": [
      "SLACK_NOTIFIER",
      "ALERT"
    ],
    "max_instances": 0,
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "Alert Destination"
    ],
    "connector_count": 0,
    "group_order": 3
  },
  {
    "type": "KINESIS",
    "name": "Kinesis",
```

```
    "desc": "Send alerts to Kinesis",
    "long_desc": "Send Tetration Alerts to Kinesis.",
    "group": "Alerts",
    "index": 4,
    "appliance_type": "tetration_edge",
    "state": "disabled",
    "limit_per_appliance": 1,
    "limit_per_rootscope": 1,
    "limit_per_cluster": 150,
    "config": [
      "KINESIS_NOTIFIER",
      "ALERT"
    ],
    "max_instances": 0,
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "Alert Destination"
    ],
    "connector_count": 0,
    "group_order": 3
  },
  {
    "type": "SYSLOG",
    "name": "Syslog",
    "desc": "Send alerts to Syslog server",
    "long_desc": "Send Tetration Alerts to Syslog server.",
    "group": "Alerts",
    "index": 0,
    "appliance_type": "tetration_edge",
    "state": "disabled",
    "limit_per_appliance": 1,
    "limit_per_rootscope": 1,
    "limit_per_cluster": 150,
    "config": [
      "SYSLOG_NOTIFIER",
      "SYSLOG_SEVERITY_MAPPING",
      "ALERT"
    ],
    "max_instances": 0,
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "Alert Destination"
    ],
    "connector_count": 0,
    "group_order": 3
  },
  {
    "type": "EMAIL",
    "name": "Email",
    "desc": "Send alerts to Email",
    "long_desc": "Send Tetration Alerts to Email.",
    "group": "Alerts",
    "index": 1,
    "appliance_type": "tetration_edge",
    "state": "disabled",
    "limit_per_appliance": 1,
    "limit_per_rootscope": 1,
    "limit_per_cluster": 150,
    "config": [
      "EMAIL_NOTIFIER",
      "ALERT"
    ],
```

```
    "max_instances": 0,
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "Alert Destination"
    ],
    "connector_count": 0,
    "group_order": 3
  },
  {
    "type": "PAGERDUTY",
    "name": "Pager Duty",
    "desc": "Send alerts to Pager Duty",
    "long_desc": "Send Tetration Alerts to Pager Duty",
    "group": "Alerts",
    "index": 3,
    "appliance_type": "tetration_edge",
    "state": "disabled",
    "limit_per_appliance": 1,
    "limit_per_rootscope": 1,
    "limit_per_cluster": 150,
    "config": [
      "PAGERDUTY_NOTIFIER",
      "ALERT"
    ],
    "max_instances": 0,
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "Alert Destination"
    ],
    "connector_count": 0,
    "group_order": 3
  },
  {
    "type": "ISE",
    "name": "ISE",
    "desc": "Collect endpoints and inventories from Cisco ISE",
    "long_desc": "Collect information about endpoints and inventories managed by Cisco ISE
 appliances and enrich endpoint inventories with user attributes and secure group tags
(SGT). Recommended usage with Cisco Secure Firewall connector for flow visibility.",
    "group": "Endpoints",
    "index": 1,
    "appliance_type": "tetration_edge",
    "state": "disabled",
    "limit_per_appliance": 1,
    "limit_per_rootscope": 1,
    "limit_per_cluster": 150,
    "config": [
      "LDAP",
      "LOG",
      "ALERT"
    ],
    "instance_spec": "ISE",
    "max_instances": 20,
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "User Insights",
      "Inventory Enrichment",
      "Endpoint Insights",
      "Software Compliance Posture",
      "MDM Insights"
    ],
```

```
    "connector_count": 0,
    "group_order": 1
  },
  {
    "type": "MERAKI",
    "name": "Meraki",
    "desc": "Collect telemetry from Meraki firewalls",
    "long_desc": "Collect telemetry data from Meraki firewalls.",
    "group": "Flow Ingest",
    "index": 5,
    "appliance_type": "tetration_data_ingest",
    "state": "disabled",
    "limit_per_appliance": 1,
    "limit_per_rootscope": 10,
    "limit_per_cluster": 100,
    "config": [
      "LOG",
      "ALERT"
    ],
    "max_instances": 0,
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "Flow Visibility"
    ],
    "connector_count": 0,
    "group_order": 0
  },
  {
    "type": "SERVICENOW",
    "name": "ServiceNow",
    "desc": "Collect ServiceNow CMDB records for inventories",
   "long_desc": "Collect CMDB information and service records from ServiceNow instance and
enriches endpoint inventories with cmdb attributes.",
    "group": "Inventory Enrichment",
    "index": 1,
    "appliance_type": "tetration_edge",
    "state": "disabled",
    "limit_per_appliance": 1,
    "limit_per_rootscope": 1,
    "limit_per_cluster": 150,
    "config": [
      "SERVICENOW",
      "SYNC_INTERVAL",
      "LOG",
      "ALERT"
    ],
    "instance_spec": "SERVICENOW",
    "max_instances": 10,
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "User Insights",
      "Inventory Enrichment",
      "Endpoint Insights",
      "Software Compliance Posture"
    ],
    "connector_count": 0,
    "group_order": 2
  },
  {
    "type": "ERSPAN",
    "name": "ERSPAN",
    "desc": "Collect ERSPAN traffic",
```

```
  "long_desc": "",
  "group": "Flow Ingest",
  "index": 7,
  "appliance_type": "tetration_erspan",
  "state": "enabled",
  "limit_per_appliance": 3,
  "limit_per_rootscope": 24,
  "limit_per_cluster": 450,
  "config": [],
  "max_instances": 0,
  "noteworthy": false,
  "hidden": false,
  "capabilities": [],
  "connector_count": 3,
  "group_order": 0
},
{
  "type": "AWS_CONNECTOR",
  "name": "AWS",
  "desc": "AWS Connector",
  "long_desc": "",
  "group": "Cloud",
  "index": 0,
  "appliance_type": "tetration_internal",
  "state": "disabled",
  "limit_per_appliance": 5000,
  "limit_per_rootscope": 5000,
  "limit_per_cluster": 100000,
  "config": [
    "AWS_CONNECTOR"
  ],
  "max_instances": 0,
  "noteworthy": true,
  "pre_release_tag": "BETA",
  "hidden": false,
  "capabilities": [
    "Flow Visibility",
    "Segmentation",
    "Managed K8s",
    "Inventory Enrichment"
  ],
  "connector_count": 0,
  "group_order": 5
},
{
  "type": "AZURE_CONNECTOR",
  "name": "Azure",
  "desc": "Azure Connector",
  "long_desc": "",
  "group": "Cloud",
  "index": 1,
  "appliance_type": "tetration_internal",
  "state": "disabled",
  "limit_per_appliance": 5000,
  "limit_per_rootscope": 5000,
  "limit_per_cluster": 100000,
  "config": [
    "AZURE_CONNECTOR"
  ],
  "max_instances": 0,
  "noteworthy": true,
  "pre_release_tag": "BETA",
  "hidden": false,
  "capabilities": [
```

```
      "Flow Visibility",
      "Segmentation",
      "Managed K8s",
      "Inventory Enrichment"
    ],
    "connector_count": 0,
    "group_order": 5
  },
  {
    "type": "GCP_CONNECTOR",
    "name": "GCP",
    "desc": "GCP Connector",
    "long_desc": "",
    "group": "Cloud",
    "index": 2,
    "appliance_type": "tetration_internal",
    "state": "disabled",
    "limit_per_appliance": 5000,
    "limit_per_rootscope": 5000,
    "limit_per_cluster": 100000,
    "config": [
      "GCP_CONNECTOR"
    ],
    "max_instances": 0,
    "noteworthy": true,
    "pre_release_tag": "BETA",
    "hidden": false,
    "capabilities": [
      "Managed K8s"
    ],
    "connector_count": 0,
    "group_order": 5
  }]
```

用于删除连接器的 API

此终端将删除给定连接器。

```
DELETE /openapi/v1/connectors/<id>
```

其中 <id> 是可从 用于获取连接器的 API，第 209 页 获取的 ID。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定连接器 ID

响应对象：返回已删除连接器的状态。

响应示例

```
resp = restclient.delete('/connectors/63c12e316419d0131767e21c')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
 {
    "status": 200,
    "code": 1000,
```

```
    "message": "deleted"
}
```

用于按 ID 获取连接器的 API

此终端会获取具有给定 ID 的连接器。

```
GET /openapi/v1/connectors/<id>
```

其中 <id> 是可从 用于获取连接器的 API，第 209 页 获取的 ID。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定连接器 ID

响应对象：返回具有给定 ID 的连接器。

响应示例

```
resp = restclient.get('/connectors/63c12e316419d0131767e21b')
 if resp.status_code == 200:
   parsed_resp = json.loads(resp.content)
   print json.dumps(parsed_resp)
```

响应示例

```
{
   "name": "ERSPAN",
   "updated_at": 0,
   "created_at": 1673604657,
   "appliance_id": "63c1272039042a1c0ddd3e20",
   "root_scope_id": "63bf8d2f497d4f7287dbd335",
   "vrf_id": 1,
   "type": "ERSPAN",
   "ip_bindings": [],
   "internal": false,
   "id": "63c12e316419d0131767e21b"
}
```

用于重命名连接器的 API

此终端会重命名具有给定 ID 的连接器。

```
PUT /openapi/v1/connectors/<id>
```

其中 <id> 是可从 用于获取连接器的 API，第 209 页 获取的 ID。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定连接器 ID
name	字符串	指定连接器的新名称

响应对象：返回具有给定 ID 和新名称的连接器。

响应示例

```
req_payload = {
    "name": "ERSPAN2",
  }
  resp = restclient.put('/ext_appliances/63c12e316419d0131767e21b',
json_body=json.dumps(req_payload))
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "name": "ERSPAN2",
    "updated_at": 0,
    "created_at": 1673604657,
    "appliance_id": "63c1272039042a1c0ddd3e20",
    "root_scope_id": "63bf8d2f497d4f7287dbd335",
    "vrf_id": 1,
    "type": "ERSPAN",
    "ip_bindings": [],
    "internal": false,
    "id": "63c12e316419d0131767e21b"
  }
```

用于获取连接器信息及详细信息的 API

此终端可获取连接器信息及详细信息。

```
GET /openapi/v1/connectors/cards/<type>?root_scope_id=<root_scope_id>
```

其中 <root_scope_id> 是可从 获取范围，第 57 页 API 获取的 root_ scope_id。在请求负载中，<type> 是用于决定连接器类型的字符串。

参数：请求 URL 包含以下参数

名称	类型	说明
root_scope_id	字符串	指定根范围
type	字符串	指定连接器类型，值可以是 "NETFLOW"、"NETSCALER" "F5" "AWS" "ANYCONNECT" "ASA" "SLACK" "KINESIS" "SYSLOG" "EMAIL" "MERAKI" "PAGERDUTY" "ISE" "SERVICENOW" "ERSPAN" "AWS_CONNECTOR" "AZURE_CONNECTOR" "GCP_CONNECTOR"

响应对象：返回给定范围内的连接器。

响应示例

```
resp = restclient.get('/connectors/cards/NETFLOW?root_scope_id=63bf8d2f497d4f7287dbd335')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "type": "NETFLOW",
    "name": "NetFlow",
    "desc": "Collect telemetry from network devices",
    "long_desc": "Collect NetFlow V9 and/or IPFIX telemetry from network devices such as
routers and switches.",
    "group": "Flow Ingest",
    "index": 0,
    "appliance_type": "tetration_data_ingest",
    "state": "disabled",
    "limit_per_appliance": 3,
    "limit_per_rootscope": 10,
    "limit_per_cluster": 100,
    "config": [
      "LOG",
      "ALERT"
    ],
    "max_instances": 0,
    "noteworthy": false,
    "hidden": false,
    "capabilities": [
      "Flow Visibility"
    ],
    "connector_count": 0,
    "info": {
      "help": "NetFlow connector collects telemetry data from various network devices (such
 as routers, switches).<br> It supports ingest of telemetry data in IPFIX and NetFlow V9
protocols. This connector can be used to discover inventory as it provides a network context.
 The connector helps convert data from flow exports and send them securely as Tetration
Flow records into an instance of Tetration. <br><br><b> Connector Alerts: </b><br> When
Connector Alerts are enabled, you may receive the following alerts: <br>   1. NetFlow
Connector is down (due to missing heartbeats). <br>   2. Informational alert on high
CPU/Memory usage."
    },
    "group_order": 0
  }
```


用于获取连接器的 API

此终端会返回给定设备的所有连接器。

```
GET
/openapi/v1/connectors?root_scope_id=<root_scope_id>&appliance_id=<appliance_id>&type=<type>&state=<state>
```

其中 <root_scope_id> 是可从 获取范围，第 57 页 API 获取的 root_ scope_id，<appliance_id> 是可以从 用于获取设备列表的 API，第 183 页 获取的 appliance_id，<type> 是用于确定连接器类型的字符串（可从 用于获取设备架构的 API，第 193 页 的“connectors”字段中获取），而 <state> 是连接器状态的过滤器。

参数：请求 URL 包含以下参数

名称	类型	说明
root_scope_id	字符串	指定根范围
appliance_id	字符串	指定设备 ID

名称	类型	说明
type	字符串	指定连接器类型，值可以是 “NETFLOW”、 “NETSCALER” “F5” “AWS” “ANYCONNECT” “ASA” “SLACK” “KINESIS” “SYSLOG” “EMAIL” “MERAKI” “PAGERDUTY” “ISE” “SERVICENOW” “ERSPAN” “AWS_CONNECTOR” “AZURE_CONNECTOR” “GCP_CONNECTOR”
state	字符串	过滤连接器状态，值可以是 “ENABLED”、 “DISABLED”或 “UNAVAILABLE”

响应对象：返回给定设备的所有连接器。

响应示例

```
resp =
restclient.get('root_scope_id=63bf8d2f497d4f7287dbd335&appliance_id=63c1272039042a1c0ddd3e20&type=ERSPAN&state=ENABLED')

  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例


```
[
    {
      "name": "ERSPAN",
      "updated_at": 0,
      "created_at": 1673604657,
      "appliance_id": "63c1272039042a1c0ddd3e20",
      "root_scope_id": "63bf8d2f497d4f7287dbd335",
      "vrf_id": 1,
      "type": "ERSPAN",
      "ip_bindings": [],
      "state": "enabled",
      "internal": false,
      "id": "63c12e316419d0131767e21b"
    },
    {
      "name": "ERSPAN",
      "updated_at": 0,
      "created_at": 1673604657,
      "appliance_id": "63c1272039042a1c0ddd3e20",
      "root_scope_id": "63bf8d2f497d4f7287dbd335",
      "vrf_id": 1,
      "type": "ERSPAN",
      "ip_bindings": [],
```

```
    "state": "enabled",
    "internal": false,
    "id": "63c12e316419d0131767e21c"
  },
  {
    "name": "ERSPAN",
    "updated_at": 0,
    "created_at": 1673604657,
    "appliance_id": "63c1272039042a1c0ddd3e20",
    "root_scope_id": "63bf8d2f497d4f7287dbd335",
    "vrf_id": 1,
    "type": "ERSPAN",
    "ip_bindings": [],
    "state": "enabled",
    "internal": false,
    "id": "63c12e316419d0131767e21d"
  }
]
```


用于创建连接器的 API

此终端会为给定设备创建连接器。

```
POST /openapi/v1/connectors
```

在请求负载中，<root_scope_id> 是可从 获取范围，第 57 页 API 获取的 root_scope_id，<appliance_id> 是可以从 用于获取设备列表的 API，第 183 页 获取的 appliance_id，而 <type> 是用于决定连接器类型的字符串（可从 用于获取设备架构的 API，第 193 页 的“connectors”字段中获取）。

参数：请求 URL 包含以下参数

名称	类型	说明
name	字符串	指定名称
root_scope_id	字符串	指定根范围
appliance_id	字符串	指定设备 ID
type	字符串	指定连接器类型，值可以是 “NETFLOW”、 “NETSCALER” “F5” “AWS” “ANYCONNECT” “ASA” “SLACK” “KINESIS” “SYSLOG” “EMAIL” “MERAKI” “PAGERDUTY” “ISE” “SERVICENOW” “ERSPAN” “AWS_CONNECTOR” “AZURE_CONNECTOR” “GCP_CONNECTOR”
version	字符串	指定版本

响应对象：返回已创建的连接器。

响应示例

```
req_payload = {
    "root_scope_id": "63c41ff2497d4f5f5be73662",
    "appliance_id": "63c6ef42bca44e2b5e729191",
    "type": "NETFLOW",
    "name": "netflowtest",
    "version": "1.1.1"
  }
  resp = restclient.post('/connectors', json_body=json.dumps(req_payload))
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "name": "netflowtest",
    "updated_at": 0,
    "created_at": 1674187875,
    "appliance_id": "63c6ef42bca44e2b5e729191",
    "root_scope_id": "63c41ff2497d4f5f5be73662",
    "vrf_id": 1,
    "type": "NETFLOW",
    "ip_bindings": [],
    "sources": [],
    "internal": false,
    "id": "63ca14631a49bd4c0e0cefa2"
  }
```

用于获取连接器配置类型上的配置的 API

此终端可获取具有给定 ID 的连接器的配置。

```
GET /openapi/v1/connectors/<id>/config?config_type=<config_type>
```

其中 <id> 是可从 用于获取设备列表的 API，第 183 页 获取的 ID。<config_type> 是可从 用于获取设备架构的 API，第 193 页 获取的“valid_config”。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定连接器 ID
config_type	字符串	指定配置类型。有关说明下列出的所有可能值，请参阅用于获取配置架构的 API，第 179 页

响应对象：返回具有给定连接器 ID 和配置类型的配置。

响应示例

```
resp = restclient.get('/connectors/63db5418e6ee1167a4c0986c/config?config_type=LOG')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
[{
   "mode": "TEST_AND_APPLY",
   "name": "Log instance 2/1/23 22:29",
   "root_scope_id": "63d98f45497d4f53005b24fa",
   "vrf_id": 1,
   "appliance_id": "63dad690e6ee1131f255e985",
   "connector_id": "63db5418e6ee1167a4c0986c",
   "service_id": "63db5418e6ee1167a4c0986d",
   "deleted": false,
   "type": "LOG",
   "state": "COMMIT",
   "error_code": "NO_ERROR",
   "error_text": "",
   "attempts": 1,
   "config": {
     "secured": {},
     "unsecured": {
       "log-level": "info",
       "max-log-size": 10,
       "max-log-age": 30,
       "max-log-backups": 20
     }
   },
   "push_to_dio_at": 1675319360,
   "created_at": 1675319360,
   "updated_at": 1675319364,
   "discovery_completed_at": 0,
   "committed_at": 1675319364,
   "delete_at": 0,
   "error_at": 0,
   "hidden": false,
   "discovery_phase": 0,
   "internal": false,
   "id": "63db5840f029813659f9fcf5"
 }]
```


用于将新配置添加到连接器的 API

此终端会向具有给定 ID 的连接器添加新配置

```
POST /openapi/v1/connectors/<id>/config
```

其中 <id> 是可从 用于获取连接器的 API，第 209 页 获取的 ID。<config_type> 是可从 用于获取设备架构的 API，第 193 页 获取的“valid_config”。要获取 <config> 架构，请从 用于获取所有类型的连接器的 API，第 198 页 响应中选择一个“config”，然后将“config”应用于 用于获取配置架构的 API，第 179 页。

参数：请求 URL 包含以下参数

名称	类型	说明
name	字符串	指定配置名称
type	字符串	指定配置类型。有关说明下列出的所有可能值，请参阅用于获取配置架构的 API，第 179 页
config	set	以 JSON 格式提供填充的配置架构

响应对象：返回更新后的配置。

响应示例

```
req_payload = {
    "name": "Log instance 2/1/23 22:29",
    "type": "LOG",
    "config": {
      "secured": {},
      "unsecured": {
        "log-level": "info",
        "max-log-size": 10,
        "max-log-age": 30,
        "max-log-backups": 20
      }
    }
  }
  resp = restclient.post('/connectors/63db5418e6ee1167a4c0986c/config',
json_body=json.dumps(req_payload))
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "mode": "TEST_AND_APPLY",
    "name": "Log instance 2/1/23 11:29",
    "root_scope_id": "63d98f45497d4f53005b24fa",
    "vrf_id": 1,
    "appliance_id": "63dad690e6ee1131f255e985",
    "connector_id": "63db5418e6ee1167a4c0986c",
    "deleted": false,
    "type": "LOG",
    "state": "PENDING",
    "attempts": 0,
    "config": {
      "secured": {},
      "unsecured": {
        "log-level": "info",
        "max-log-size": 10,
        "max-log-age": 30,
        "max-log-backups": 20
      }
    },
    "push_to_dio_at": 0,
    "created_at": 1675322272,
    "updated_at": 1675322272,
    "discovery_completed_at": 0,
    "committed_at": 0,
    "delete_at": 0,
    "error_at": 0,
    "hidden": false,
    "discovery_phase": 0,
    "internal": false,
    "id": "63db63a0f029813659f9fcf7"
  }
```

用于删除配置的 API

此终端会从给定连接器删除配置。

```
DELETE /openapi/v1/connectors/<id>/config/<config_id>
```

其中 <id> 是可从 用于获取连接器的 API，第 209 页 获取的 ID，而 <config_id> 是可从 用于获取连接器配置类型上的配置的 API，第 212 页 获取的 ID。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定连接器 ID
config_id	字符串	指定配置 ID

响应对象：返回为给定连接器删除的配置的状态。

响应示例

```
resp =
restclient.delete('/connectors/63c1272039042a1c0ddd3e20/config/63c1272039042a1c0ddd3e21')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "status": 200,
    "code": 1000,
    "message": "deleted"
  }
```

-

用于获取配置的 API

此终端获取具有给定 ID 的配置

```
GET /openapi/v1/connectors/<id>/config/<config_id>
```

其中 <id> 是可从 用于获取连接器的 API，第 209 页 获取的 ID，而 <config_id> 是可从 用于获取连接器配置类型上的配置的 API，第 212 页 获取的 ID。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定设备 ID
config_id	字符串	指定配置 ID

响应对象：返回具有给定 ID 的配置。

响应示例

```
resp = restclient.get('/connectors/63db5418e6ee1167a4c0986c/config/63db5840f029813659f9fcf5')

  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "mode": "TEST_AND_APPLY",
    "name": "Log instance 2/1/23 22:29",
    "root_scope_id": "63d98f45497d4f53005b24fa",
    "vrf_id": 1,
    "appliance_id": "63dad690e6ee1131f255e985",
    "connector_id": "63db5418e6ee1167a4c0986c",
    "service_id": "63db5418e6ee1167a4c0986d",
    "deleted": false,
    "type": "LOG",
    "state": "COMMIT",
    "error_code": "NO_ERROR",
    "error_text": "",
    "attempts": 1,
    "config": {
      "secured": {},
      "unsecured": {
        "log-level": "info",
        "max-log-size": 10,
        "max-log-age": 30,
        "max-log-backups": 20
      }
    },
    "push_to_dio_at": 1675319360,
    "created_at": 1675319360,
    "updated_at": 1675319364,
    "discovery_completed_at": 0,
    "committed_at": 1675319364,
    "delete_at": 0,
    "error_at": 0,
    "hidden": false,
    "discovery_phase": 0,
    "internal": false,
    "id": "63db5840f029813659f9fcf5"
  }
```


用于列出可用于连接器的故障排除命令的 API

此终端会返回可用于给定连接器的故障排除命令列表。

```
GET /openapi/v1/connectors/<id>/commands/available
```

其中 <id> 是可从 用于获取连接器的 API，第 209 页 获取的 ID。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定连接器 ID

响应对象：返回可用于给定连接器的故障排除命令列表。

响应示例

```
resp = restclient.get('/connectors/63c6f2701a49bd2bb0696cab/commands/available')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例


```
[
    {
```

```
    "type": "LIST_SERVICE_FILES",
    "name": "List a directory in a service"
  },
  {
    "type": "CONTROLLER_PROFILING",
    "name": "Collect controller profile"
  },
  {
    "type": "SHOW_LOG",
    "name": "Show logs"
  },
  {
    "type": "SHOW_SERVICE_LOG",
    "name": "Show service logs"
  },
  {
    "type": "RESTART_CONNECTOR_CONTAINER",
    "name": "Restart the connector docker container"
  },
  {
    "type": "SHOW_SUPERVISOR_COMMANDS",
    "name": "Execute supervisorctl command"
  },
  {
    "type": "CONNECTOR_ALERT_INTERVAL_CONNECTOR",
    "name": "Override connector alert interval for Connector"
  },
  {
    "type": "SERVICE_PROFILING",
    "name": "Collect connector profile"
  },
  {
    "type": "SNAPSHOT_CONNECTOR",
    "name": "Collect Snapshot from a connector"
  },
  {
    "type": "PACKET_CAPTURE",
    "name": "Packet capture"
  },
  {
    "type": "NETWORK_CONNECTIVITY_COMMANDS",
    "name": "Test network connectivity"
  },
  {
    "type": "SHOW_SERVICE_RUNNING_CONF",
    "name": "Show running configuration of a service"
  },
  {
    "type": "RESTART_CONNECTOR_SERVICE",
    "name": "Restart the connector service"
  },
  {
    "type": "SHOW_SYS_COMMANDS",
    "name": "Execute system command"
  }
]
```


用于列出故障排除命令的 API

此终端会返回为给定连接器启用的故障排除命令的列表。

```
GET /openapi/v1/connectors/<id>/commands
```

其中 <id> 是可从 用于获取连接器的 API，第 209 页 获取的 ID。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定连接器 ID

响应对象：返回为给定连接器启用的故障排除命令的列表。

响应示例

```
resp = restclient.get('/connectors/63db5418e6ee1167a4c0986c/commands')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
[
    {
      "appliance_id": "63dad690e6ee1131f255e985",
      "connector_id": "63db5418e6ee1167a4c0986c",
      "service_id": "63db5418e6ee1167a4c0986d",
      "state": "success",
      "level": "SERVICE",
      "command": "SHOW_LOG",
      "arg_string": "",
      "args": {
        "pattern": "info"
      },
      "tailed": false,
      "rc": 0,
      "push_to_dio_at": 1675319615,
      "attempts": 1,
      "error_code": "NO_ERROR",
      "error_text": "",
      "deleted": false,
      "deleted_at": 0,
      "created_at": 1675319613,
      "total_chunk": 0,
      "id": "63db593df029813659f9fcf6"
    }
  ]
```

用于创建故障排除命令的 API

此终端创建可用于给定连接器的故障排除命令。

```
POST /openapi/v1/connectors/<id>/commands
```

此处的 <id> 是可从 用于获取连接器的 API，第 209 页 获取的 ID。在请求负载中，<command> 是故障排除命令类型，可从 用于列出可用于连接器的故障排除命令的 API，第 216 页 获取。<arguments> 是命令架构的填充 JSON 对象，可从 用于获取故障排除命令架构的 API，第 181 页 中获取。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定连接器 ID
command	字符串	指定命令类型

名称	类型	说明
arguments	set	以 JSON 格式提供填充的命令架构

响应对象：返回为给定设备创建的故障排除命令。

响应示例

```
req_payload = {
    "command": "SHOW_LOG",
    "arguments": {
      "pattern": "info"
    }
  }
  resp = restclient.post('/connectors/63db5418e6ee1167a4c0986c/commands',
json_body=json.dumps(req_payload))
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "appliance_id": "63dad690e6ee1131f255e985",
    "connector_id": "63db5418e6ee1167a4c0986c",
    "service_id": "63db5418e6ee1167a4c0986d",
    "state": "pending",
    "level": "SERVICE",
    "command": "SHOW_LOG",
    "args": {
      "pattern": "info"
    },
    "tailed": false,
    "rc": 0,
    "push_to_dio_at": 0,
    "attempts": 0,
    "deleted": false,
    "deleted_at": 0,
    "created_at": 1675319613,
    "total_chunk": 0,
    "id": "63db593df029813659f9fcf6"
  }
```

用于删除故障排除命令的 API

此终端删除可用于给定连接器的故障排除命令。

```
DELETE /openapi/v1/connectors/<id>/commands/<command_id>
```

其中 <id> 是可从 用于获取连接器的 API，第 209 页 获取的 ID，而 <command_id> 是可从 用于列出故障排除命令的 API，第 195 页 获取的 ID。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定连接器 ID
command_id	字符串	指定命令 ID

响应对象：返回为给定连接器删除的故障排除命令的状态。

响应示例

```
resp =
restclient.delete('/connectors/63c12e316419d0131767e21c/commands/63c10a0039042a6aee1b008c')

  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "status": 200,
    "code": 1000,
    "message": "deleted"
  }
```

用于返回故障排除命令的 API

此终端会返回给定连接器的选定故障排除命令。

```
GET /openapi/v1/connectors/<id>/commands/<command_id>
```

其中 <id> 是可从 用于获取连接器的 API，第 209 页 获取的 ID，而 <command_id> 是可从 用于列出故障排除命令的 API，第 195 页 获取的 ID。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定连接器 ID
command_id	字符串	指定命令 ID

响应对象：返回为给定连接器选择的故障排除命令。

响应示例

```
resp =
restclient.get('/connectors/63db5418e6ee1167a4c0986c/commands/63db593df029813659f9fcf6')
  if resp.status_code == 200:
    parsed_resp = json.loads(resp.content)
    print json.dumps(parsed_resp)
```

响应示例

```
{
    "appliance_id": "63dad690e6ee1131f255e985",
    "connector_id": "63db5418e6ee1167a4c0986c",
    "service_id": "63db5418e6ee1167a4c0986d",
    "state": "success",
    "level": "SERVICE",
    "command": "SHOW_LOG",
    "arg_string": "",
    "args": {
      "pattern": "info"
    },
    "tailed": false,
    "rc": 0,
    "push_to_dio_at": 1675319615,
    "attempts": 1,
```

```
  "error_code": "NO_ERROR",
  "error_text": "",
  "deleted": false,
  "deleted_at": 0,
  "created_at": 1675319613,
  "total_chunk": 0,
  "id": "63db593df029813659f9fcf6"
}
```

用于将连接器命令的输出下载为文件的 API

此终端会将命令的输出下载为文件。

```
GET /openapi/v1/connectors/<id>/commands/{command_id}/download
```

其中 <id> 是可从 用于获取连接器的 API，第 209 页 获取的 ID，而 <command_id> 是可从 用于列出故障排除命令的 API，第 195 页 获取的 ID。并非所有命令都有可下载的输出，请检查 用于获取故障排除命令架构的 API，第 181 页 提供的命令架构，其中 "output_type": "FILE" 表示其具有可下载内容，"output_ext" 表示文件类型。

参数：请求 URL 包含以下参数

名称	类型	说明
id	字符串	指定连接器 ID
command_id	字符串	指定命令 ID

响应对象：返回为给定连接器选择的故障排除命令。

响应示例

```
resp = restclient.download('downloadFile',
'/connectors/63c6ef42bca44e2b5e729191/commands/63cace941a49bd4c0e0cf45a/download')
```

当地语言翻译版本说明

思科可能会在某些地方提供本内容的当地语言翻译版本。请注意，翻译版本仅供参考，如有任何不一致之处，以本内容的英文版本为准。