OpenAPI Authentication

OpenAPI uses a digest based authentication scheme. The workflow is as follows:

1. Log into the Tetration UI Dashboard
2. Generate an API key and an API secret with the desired capabilities.
3. Use Tetration API sdk to send REST requests in json format.
4. To use python sdk, user would install the sdk using `pip install tetpyclient`.
5. Once python sdk is installed, here is some boilerplate code for instantiating the RestClient:

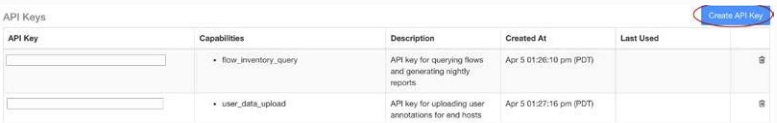
```
from tetpyclient import RestClient

API_ENDPOINT="https://<UI_VIP_OR_DNS_FOR_TETRATION_DASHBOARD>"


# ``verify`` is an optional param to disable SSL
server authentication.
# By default, Tetration appliance 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 Tetration appliance (from ``Settings
> Company`` Tab)
# which is signed by their enterprise CA, then server
side authentication
# should be enabled.
# 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=True)
# 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')
```

Generate API Key and Secret

In the UI dashboard, navigate to API Keys:

Click the Create API Key button



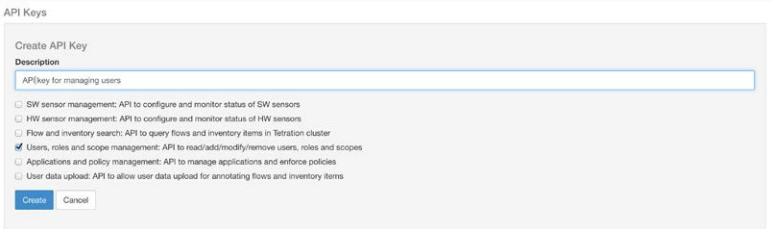
Specify the desired capabilities for the key and secret. User must choose the limited set of capabilities that they intend to use the API Key+Secret pair for. Note, the API capabilities available to the user varies based on user's roles, e.g. Site Admin users can generate keys to manage software agents but this capability is not available to not non Site Admin users.

List of API capabilities include:

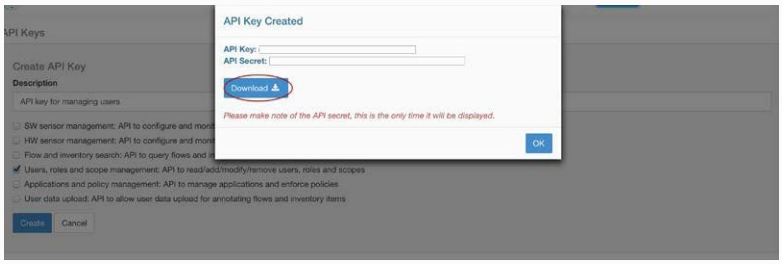
- SW agent management (`sensor_management`): able to configure and monitor status of SW agents (available only to Site Admin users)
- HW agent management (`hw_sensor_management`): able to configure and monitor status of HW agents (available only to Site Admin users)
- Flow and inventory search (`flow_inventory_query`) : able to query flows and inventory items in Tetration

cluster

- Users, roles and scope management (`user_role_scope_management`): able to read/add/modify/remove users, roles and scopes (available only to Site Admin users)
- User data upload (`user_data_upload`): allow user to upload data for annotating flows and inventory items (available only to Site Admin users)
- Applications and policy management (`app_policy_management`): able to manage applications and enforce policies



Copy and paste the key and secret and save it in a safe location. Alternatively, download the API Credentials file.



Previous

Next

Scopes

This set of APIs can be used to manage Scopes (or AppScopes) in Tetration cluster deployment. They require the `user_role_scope_management` capability associated with the API key. The API to get the list of scopes is also available to API keys with `app_policy_management` or `sensor_management` capability.

Scope object

The scope object attributes are described below:

Attribute	Type	Description
id	string	Unique identifier for the scope.
short_name	string	User specified name of the scope.
name	string	Fully qualified name of the scope. This is a fully qualified name, i.e. it has name of parent scopes (if applicable) all the way to the root scope.
description	string	User specified description of the scope.
short_query	JSON	Filter (or match criteria) associated with the scope.
		Filter (or match criteria)

query	JSON	associated with the scope in conjunction with the filters of the parent scopes (all the way to the root scope).
vrf_id	integer	ID of the VRF to which scope belongs to.
parent_app_scope_id	string	ID of the parent scope.
child_app_scope_ids	array	An array of scope children's ids.
policy_priority		Used to sort application priorities. See Semantics and Viewing.
dirty	bool	Indicates a child or parent query has been updated and that the changes need to be committed.
dirty_short_query	JSON	Non-null if the query for this scope has been updated but not yet committed.

Get scopes

This endpoint returns a list of scopes known to Tetration appliance. This API is available to API keys with either `app_policy_management` or `user_role_scope_management` capability.

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

Parameters: None

Returns a list of scope objects.

Create a scope

This endpoint is used to create new scopes.

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

Parameters:

Name	Type	Description
short_name	string	User specified name of the scope.
description	string	User specified description of the scope.
short_query	JSON	Filter (or match criteria) associated with the scope.
parent_app_scope_id	string	ID of the parent scope.

Sample python code

```
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))
```

Get specific scope

This endpoint returns an instance of a scope.

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

Returns the scope object associated with the specified ID.

Update a scope

This endpoint updates a scope. Changes to the `name` and `description` are applied immediately. Changes to the `short_query` mark the scope as 'dirty' and set the `dirty_short_query` attribute. Once all scope query changes, under a given root scope, are made, one needs to ping the Commit Scope Query Changes endpoint to commit all the required updates.

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

Parameters:

Name	Type	Description
short_name	string	User specified name of the scope.
description	string	User specified description of the scope.
short_query	JSON	Filter (or match criteria) associated with the scope.

Returns the modified scope object associated with specified ID.

Delete specific scope

This endpoint deletes the specified scope.

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

Commit scope query changes

This endpoint triggers an asynchronous background job to update all ‘dirty’ children under a given root scope. This job updates scopes and applications, see Scopes for more details.

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

Parameters:

Name	Type	Description
root_app_scope_id	string	ID for a root scope for which all children will be updated.

Returns 201 to indicate the job has been enqueued. To check if the job has completed, poll the root scope’s ‘dirty’ attribute to see if it has been set to false.

Previous

Next

Roles

This set of APIs can be used to manage user roles. They require the `user_role_scope_management` capability associated with the API key.

Note

These APIs are only available to site admins and owners of root scopes.

Role object

The role object attributes are described below:

Attribute	Type	Description
id	string	Unique identifier for the role.
name	string	User specified name for the role.
description	string	User specified description for the role.

Get roles

This endpoint returns a list of roles accessible to the user. Roles can be filtered to a given root scope. If no scope if provided, all roles, for all scopes the user has access to, are returned. Service provider roles will only be returned if the user is a site admin.

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

Parameters:

Name	Type	Description
app_scope_id	string	(optional) ID of a root scope to return roles only assigned to that scope.

Returns a list of user role objects.

Create a role

This endpoint is used to create a new role.

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

Parameters:

Name	Type	Description
name	string	User specified name for the role.
description	string	User specified description for the role.
app_scope_id	string	(optional) The scope ID under which the role is created If no scope ID mentioned the role is considered as service provider role.

The requesting user must have access to the provided scope. A role without a scope is called a 'Service Provider Role' and only site admin may create them.

Sample python code

```
req_payload = {
      "name": "Role Name",
      "description": "Role Description",
      "app_scope_id": "App Scope Id"
}
resp = restclient.post('/roles',
json_body=json.dumps(req_payload))
```

Get specific role

This endpoint returns a specific role object.

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

Returns a role object associated with specified ID.

Give a role access to scope

This endpoint gives a role the specified access level to a scope.

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

Capabilities can only be added to the roles that the user has access to. If the roles is assigned to a scope, capabilities must correspond to that scope or its children. Service provider roles (those not assigned to a scope) can add capabilities for any scope.

Parameters:

Name	Type	Description
app_scope_id	string	ID of the scope to which access is provided.
ability	string	Possible values are `SCOPE_READ`, `SCOPE_WRITE`, `EXECUTE`, `ENFORCE`, `SCOPE_OWNER`

For more description of abilities, refer to Roles.

Delete specific role

This endpoint deletes the specified role.

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

Previous

Next

Users

This set of APIs manages users. They require the `user_role_scope_management` capability associated with the API key.

Note

These APIs are only available to site admins and owners of root scopes.

User object

The user object attributes are described below:

Attribute	Type	Description
id	string	Unique identifier for the user role.
email	string	Email associated with user account.
first_name	string	First name.
last_name	string	Last name.
role_ids	list	List of IDs of roles assigned to the user account.
deleted_at	integer	Unix timestamp of when the user has been disabled. Zero or null, otherwise.

Get users

This endpoint returns a list of user objects known to the Tetration appliance.

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

Parameters:

Name	Type	Description
include_disabled	boolean	To include disabled users, defaults to false.
app_scope_id	string	Return only users assigned to the provided scope.

Returns a list of user objects. Only site admins can see ‘Service provider users’, i.e. those not assigned to a scope.

Create a new user account

This endpoint is used to create a new user account.

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

Parameters:

Name	Type	Description
email	string	Email associated with user account.
first_name	string	First name.
last_name	string	Last name.
app_scope_id	string	(optional) Root scope to which user belongs.
role_ids	list	(optional) The list of roles that should be assigned to the user.

The app_scope_id is the ID of the root scope to which the user is to be assigned. If the `app_scope_id` is not present then the user is a ‘Service Provider user.’ Only

site admins can create service provider users. The `role_ids` are the ids of the roles that were created under the specified app scope.

Sample python code

```
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))
```

Get specific user

This endpoint returns specific user object.

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

Returns a user object associated with specified ID.

Add role to the user account

This endpoint is used to add a role to a user account.

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

Parameters:

Name	Type	Description
role_id	string	ID of the role object to be added.

Remove role from the user account

This endpoint is used to remove a role from a user

account.

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

Parameters:

Name	Type	Description
role_id	string	ID of the role object to be removed.

Delete specific user role

This endpoint deletes the specified user account.

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

Previous

Next

Inventory filters

Inventory filters encode the match criteria for inventory search queries. These set of APIs provide functionality similar to what is described in Inventory Filters. This set of APIs requires either `sensor_management` or `app_policy_management` capability associated with the API key.

Inventory Filter Object

The inventory filter JSON object is returned as a single object or an array of objects depending on the API endpoint. The object's attributes are described below:

Attribute	Type	Description
id	string	Unique identifier for the inventory filter.
name	string	User specified name of the inventory filter.
app_scope_id	string	ID of the scope associated with the filter.
short_query	JSON	Filter (or match criteria) associated with the filter.
primary	boolean	When 'true' it means inventory filter is restricted to ownership scope.
query	JSON	Filter (or match criteria) associated with the filter in conjunction with the filters of the parent scopes. These conjunctions take effect if 'restricted to

		ownership scope' checkbox is checked. If 'primary' field is false then query is same as short_query.

Get inventory filters

This endpoint returns a list of inventory filters visible to the user.

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

Parameters: None

Create an inventory filter

This endpoint is used to create an inventory filter.

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

Parameters:

Name	Type	Description
name	string	User specified name of the application scope.
query	JSON	Filter (or match criteria) associated with the scope.
app_scope_id	string	ID of the parent application scope.
primary	boolean	When 'true' it means inventory filter is restricted to ownership scope.

Sample python code

```
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))
```

Get specific inventory filter

This endpoint returns an instance of inventory filter.

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

Returns an inventory filter object associated with specified ID.

Delete specific application scope

This endpoint deletes the specified instance of inventory filter.

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

Previous

Next

Flow Search

The flow search feature provides similar functionality as described in Flows. These set of APIs require the `flow_inventory_query` capability associated with the API key.

Query for flow dimensions

This endpoint returns the list of flow columns on which search criteria (or *filters*) can be specified for flow search queries (below).

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

Query for flow metrics

This endpoint returns the list of metrics (e.g. byte count, packet count) associated with flow observations.

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

Query for flows

This endpoint returns the list of flows matching the filter criteria. Each flow object in the result has attributes that are a union of flow dimensions (returned by the flow dimensions API above) as well as the flow metrics (returned by the flow metrics API above).

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

The list of columns that can be specified in the filter criteria can be obtained by `/openapi/v1/flowsearch/dimensions` API.

The query body consists of a JSON body with the following keys.

Name	Type	Description
t0	integer or string	Flow search start time (epoch or ISO 8601)
t1	integer or string	Flow search end time (epoch or ISO 8601)
filter	JSON	Query filter. If filter is empty (i.e. {}), then query matches all flows.
scopeName	string	(optional) Full name of the scope to which query is restricted.
dimensions	array	(optional) List of dimension names to be returned in the result of flowsearch API. This is an optional parameter. If unspecified, flowsearch results return all the available dimensions. This option is useful to specify a subset of the available dimensions when caller does not care about the rest of the dimensions.
metrics	array	(optional) List of metric names to be returned in the result of flowsearch API. This is an optional parameter. If unspecified, flowsearch results return all the available metrics. This option is useful to specify a subset of the available metrics when caller does not care about the rest of the metrics.

limit	integer	(optional) Number of response flows limit.
offset	integer	(optional) Offset object received from previous response.
descending	boolean	(optional) If this parameter is false or left unspecified, results are in ascending order of timestamps. If parameter value is true, results are in descending order of timestamps.

The body of the request should be a JSON formatted query. An example of a query body is shown below.

```
{
    "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>
}
```

Filters

The filter supports primitive filters and logical filters (“not”, “and”, “or”) comprised of one or more primitive filters.

Format of primitive filter is as follows:

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

For primitive filters, operator can be a comparison operator like `eq`, `ne`, `lt`, `lte`, `gt` or `gte`. Operator could also be `in`, `regex`, `subnet`, `contains` or `range`.

Some examples of primitive filters might include:

```
{"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]}
```

User can also specify complex filters using boolean operations like `not`, `and` or `or`. Following are some examples of these type of filters:

```
# "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"}
```

```
}
```

More formally, schema of `filter` in the flow search request is as follows:

Keys	Values
type	Filter type
field	Filter field column for primitive filters
filter	Filter object (only used for `not` filter type)
filters	List of filter objects (used for `and` and `or` filter types)
value	Value for primitive filters
values	List of values for primitive filters with filter type `in` or `range`

Primitive Filter Types

`eq`, `ne`
: Searches flows for equality or inequality respectively in column specified by `"field"` with value specified by `"value"`. Supports the following fields: `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`. These operators also work on user annotated columns.

`lt`, `lte`, `gt`, `gte`
: Searches flows where values of column specified by `"field"` are less than, less than equal to, greater than or greater than equal to (as applicable) the value specified by `"value"`. Supports the following fields: `[src_port, dst_port]`.

`range`
: Searches flows for values of column specified by `"field"` between range start and range end specified by `"values"` list (this list must be of size 2 for *"range"* filter type – first value is the range start and second is the range end). Supports the following fields: `[src_port, dst_port]`.

`in`
: Searches flows for membership in column

specified by `"field"` with membership list specified by `"values"`. Supports the following fields: `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`. This operator also works on user annotated columns.

`regex`, `contains`
: Searches flows for regex matches or containment matches respectively in column specified by `"field"` with regex specified by `"value"`. Supports the following fields: `src_hostname`, `dst_hostname`, `src_scope_name`, `dst_scope_name`, `vrf_name`, `src_enforcement_epg_name`, `dst_enforcement_epg_name`. These operators also work on user annotated columns. Filters with `regex` type must use Java style regex patterns as `"value"`.

`subnet`
: Searches flows for subnet membership specified by `"field"` as a string in CIDR notation. Supports the following fields: `["src_address", "dst_address"]`

Logical Filter Types

`not`
: Logical not filter of object specified by `"filter"`.

`and`
: Logical and filter of list of filter objects specified by `"filters"`.

`or`
: Logical or filter of list of filter objects specified by `"filters"`.

Response

The response is a JSON object in the body with the following properties.

Keys	Values
offset	Response offset to be passed for the next page of results
results	List of results

To generate the next page of results, take the object

received by the response in `offset` and pass it as the value for the `offset` of the next query.

Sample python code

```
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)
```

TopN query for flows

This endpoint returns a top n sorted list of values of specified dimension where rank in the list is determined by the aggregate of specified metric.

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

The list of columns that can be specified in the filter criteria can be obtained by `/openapi/v1/flowsearch/dimensions` API. The body of the request should be a JSON formatted query. An example of a query body is shown below. Parameters `t0` and `t1` in the request body can be in epoch format or in iso8601 format. TopN API only allows querying maximum time range of 1 day. The

dimension on which the grouping has to be done should be specified through `dimension`. The metric by which top N results need to ranked should be specified in `metric` field in the JSON body. Users should specify a `threshold` with a minimum value of 1 which signifies the 'N' in 'TopN'. The maximum value of this `threshold` is 1000. Even if the user specifies more than 1000 the API returns only a maximum of 1000 results. User can also specify an optional parameter called `scopeName` which is the full name of the application scope to which user wants to restrict the search. If this parameter is not specified, topN request applies to all scopes to which user has read access to. The `filter` is same as that of filter of Flow Search Filters. If the `filter` is not mentioned, then the topN is applied on all the flow entries.

```
{
  "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"    #optional
}
```

The query body consists of a JSON body with the following keys.

Keys	Values
t0	Start time of the Flow (epoch or ISO 8601)
t1	End time of the Flow (epoch or ISO 8601)
filter	Query filter. If filter is empty (i.e. {}),or filter is absent (optional) then topN query is applied on all flow entries

scopeName	Full name of the scope to which query is restricted to (optional)
dimension	The dimension is a field on which we are grouping.
metric	The metric is the total count of values of the dimension.
threshold	Threshold is ‘N’ in the topN.

Response

The response is a JSON object in the body with the following properties.

Keys	Values
result	Array of the top N entries

Sample python code to access topN API

```
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)
```

Sample response of topN API

```
[
  { "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}
    ]
  }
]
```

Flow Count

This endpoint returns the number of flow observations matching the specified criteria.

```
POST /openapi/v1/flowsearch/count
```

The body of the request should be a JSON formatted query. An example of a query body is shown below. Parameters `t0` and `t1` in the request body can be in epoch format or in iso8601 format. This API only allows querying maximum time range of 1 day. User can also specify an optional parameter called `scopeName` which is the full name of the application scope to which user wants to restrict the search. If this parameter is not specified, flow observation count API request applies to all scopes to which user has read access to. The `filter` is same as that of filter of Flow Search Filters.

```
{
  "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"    #optional
}
```

The query body consists of a JSON body with the following keys.

Keys	Values
t0	Start time of the Flow (epoch or ISO 8601)
t1	End time of the Flow (epoch or ISO 8601)
filter	Query filter. If filter is empty (i.e. {}), then query matches all flows.
scopeName	Full name of the scope to which query is restricted to (optional)

Response

The response is a JSON object in the body with the following properties.

Keys	Values
count	The number of flow observations matching flow search criteria.

Sample python code for flow count API

```
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)
```

Sample response of count API

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

Inventory

The inventory search APIs provide similar functionality as described in Inventory Search. These set of APIs require the `flow_inventory_query` capability associated with the API key.

Query for inventory dimensions

This endpoint returns the list of inventory columns on which search criteria (or *filters*) can be specified for inventory search queries.

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

Inventory search

This endpoint returns the list of inventory items matching the specified criteria.

```
POST /openapi/v1/inventory/search
```

The list of columns that can be specified in the filter criteria can be obtained with the `/openapi/v1/inventory/search/dimensions` API.

Parameters:

Name	Type	Description
filter	JSON	A filter query.
scopeName	string	(optional) Name of the scope by which to limit

		results.
limit	integer	(optional) Max number of results to return.
offset	integer	(optional) Offset from the previous request to get the next page.

The body of the request must be a JSON formatted query. An example of a query body is shown below.

```
{
    "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
    "limit": 100,
    "offset": <offset-object>               # optional
}
```

The query body consists of a JSON body with the following keys.

Keys	Values
filter	Query filter. If filter is empty (i.e. {}), then query matches all inventory items.
scopeName	Full name of the scope to which query is restricted to (optional)
dimensions	List of dimension names to be returned in the result of inventory search API. This is an optional parameter. If unspecified, results return all the available dimensions.

	This option is useful to specify a subset of the available dimensions when caller does not care about the rest of the dimensions.
limit	Number of response items limit (optional)
offset	Offset object received from previous response (optional)

Response

The response is a JSON object in the body with the following properties.

Name	Type	Description
offset	integer	Response offset to be passed for the next page of results.
results	array of objects	List of results.

The response may contain an `offset` field for paginated responses. Users will need to specify the same offset in the subsequent request to get the next set of results.

Sample Python code

```
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)
```

Inventory Statistics

This endpoint returns statistics for inventory items.

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

Path Parameter	Description
id	Inventory item id as `{ip}-{vrf_id}` such as `1.1.1.1-123`

Query Parameter	Description
t0	Start time for statistics in epoch time
t1	End time for statistics in epoch time
td	Granularity for statistic aggregations. An integer specifies number of seconds. Strings may be passed such as “minute”, “hour”, and “day”.

Sample Python code

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

Inventory count

This endpoint returns the count of inventory items matching the specified criteria.

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

The list of columns that can be specified in the filter criteria can be obtained with the `/openapi/v1/inventory/search/dimensions` API.

Parameters:

Name	Type	Description
filter	JSON	A filter query.
scopeName	string	(optional) Name of the scope by which to limit results.

The body of the request must be a JSON formatted query. An example of a query body is shown below.

```
{
    "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
}
```

Response

The response is a JSON object in the body with the following properties.

Keys	Values

count	Number of inventory items matching the filter Criteria

Sample python code

```
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)
```

Previous

Next

Applications

Application workspaces are the containers for defining, analyzing and enforcing policies for a particular application. For more information about how they work see the Application Workspaces documentation. This set of APIs requires the `app_policy_management` capability associated with the API key.

Application Object

The application JSON object is returned as a single object or an array of objects depending on the API endpoint. The object's attributes are described below:

Attribute	Type	Description
id	string	A unique identifier for the application.
name	string	User specified name of the application.
description	string	User specified description of the application.
app_scope_id	string	ID of the scope assigned to the application.
author	string	First and last name of the user who created the application.
primary	boolean	Indicates if the application is primary for its scope.
created_at	integer	Unix timestamp of when the application was created.

version	integer	The latest version of the application policies

List Applications

This endpoint will return an array of applications that are visible to the users.

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

Retrieve a single Application

This endpoint will return the requested application as a single JSON object.

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

Create an Application

This endpoint creates an Application. It is possible to define policies by posting a JSON body containing the cluster and policy definitions.

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

Parameters:

Name	Type	Description
app_scope_id	string	The scope ID to assign to the application.
name	string	(optional) A name for the application.
description	string	(optional) A description for the application.
primary	string	(optional) Set to ‘true’ to indication this application should be primary for the given scope.

Extra optional parameters may be included describing policies to be created within the application.

> **Note**
>
> The scheme corresponds to that returned during export from the UI and the **Details** endpoint.

Name	Type	Description
clusters	array of clusters	Groups of nodes to be used to define policies.
inventory_filters	array of inventory filters	Filters on datacenter assets.
absolute_policies	array of policies	Ordered policies to be created with the absolute rank.
default_policies	array of policies	Ordered policies to be created with the default rank.
catch_all_action	string	“ALLOW” or “DENY”

Cluster object attributes:

Name	Type	Description
id	string	Unique identifier to be used with policies.
name	string	Displayed name of the cluster.
nodes	array of nodes	Nodes or endpoints that are part of the cluster.

Node object attributes:

Name	Type	Description
ip	string	IP or subnet of the node. eg

		10.0.0.0/8 or 1.2.3.4
name	string	Displayed name of the cluster.

Inventory Filter object attributes:

Name	Type	Description
id	string	Unique identifier to be used with policies.
name	string	Displayed name of the cluster.
Query	object	JSON object representation of an inventory filter query.

Policy object attributes:

Name	Type	Description
consumer_filter_id	string	ID of a defined cluster.
provider_filter_id	string	ID of a defined cluster.
action	string	“ALLOW” or “DENY”
l4_params	array of l4details	List of allowed ports and protocols.

L4Details object attributes:

Name	Type	Description
proto	integer	Protocol Integer value (NULL means all protocols).
port	array	Inclusive range of ports. eg [80, 80] or [5000, 6000].

Delete an Application

Remove an application.

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

Enforcement must be disabled on the application before it can be deleted.

Retrieve Application Details

This endpoint returns a full export JSON file for the Application. This will include policy and cluster definitions.

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

Returns the latest version of the application. This is not necessarily the latest enforced or analyzed version.

Enforce a single Application

Enable enforcement on the latest set of policies in the Application.

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

Warning

New host firewall rules will be inserted and any existing rules will be deleted on the relevant hosts.

Parameters:

Name	Type	Description
application_id	string	The unique identifier for the application.
version	integer	(optional) The policy version to enforce.

If a `version` is not provided the latest policies of the application will be enforced.

Response object Attributes:

Name	Type	Description
epoch	string	Unique identifier for the latest enforcement profile.

Disable a single Application

Disable enforcement on the application.

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

Warning

New host firewall rules will be inserted and any existing rules will be deleted on the relevant hosts.

Response object Attributes:

Name	Type	Description
epoch	string	Unique identifier for the latest enforcement profile.

Previous

Next

Enforcement

Policy enforcement is the feature where generated policies are pushed to the assets in the scope of an application and new firewall rules are written. More information can be found in the Enforcement documentation. This set of APIs requires the `app_policy_management` capability associated with the API key.

Agent Network Policy Config

This endpoint returns an Agent object according to the agent ID. It is useful for fetching the network policy, agent configuration, its version, etc.

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

Parameters:

Name	Type	Description
aid	string	Agent UUID for network policy config.

Response Object:

Attribute	Type	Description
desired_network_policy_config	object	Network Policy Configuration
network_policy	object	Network Policy. The network policies or

		concrete policies configured on the agent.
concrete_policy_id	string	Used for the statistics endpoint.

Concrete Policy Statistics

This endpoint returns statistics for concrete policies given the agent ID and the concrete policy ID. The endpoint returns an array of Timeseries Concrete Policy Result objects.

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

Path Parameters:

Name	Type	Description
aid	string	Agent UUID for statistics.
cid	string	Concrete Policy UUID for statistics.

Query Parameters:

Name	Type	Description
t0	integer	Start time for statistics in epoch time
t1	integer	End time for statistics in epoch time
td	integer or string	Granularity for statistic aggregations. An integer specifies number of seconds. Strings may be passed such as “minute”, “hour”, and “day”.

JSON Object Definitions

Agent

Attribute	Type	Description
agent_uuid	string	A unique identifier for the agent
desired_network_policy_config	object	The Network Policy Configuration on the agent
desired_policy_update_timestamp	int	Timestamp when agent updated with policy

Network Policy Configuration

Attribute	Type	Description
version	string	Version number
network_policy	array	Array of Network Policy objects
address_sets	array	Array of Address Set objects for IP set feature

Network Policy

Attribute	Type	Description
priority	string	Priority of concrete policy
enforcement_intent_id	string	Intent ID of enforcement intent
concrete_policy_id	string	Concrete Policy ID
match	object	Match criteria for policy. Exactly one of match or match_set will be present.
action	object	Action for policy match

workspace_id	string	ID for ADM/enforcement workspace
policy_intent_group_id	string	Policy intent group ID
match_set	object	:ref`MatchSet` object for IP set support. Exactly one of match or match_set will be present.

Match

Attribute	Type	Description
src_addr	object	Subnet object for source address
dst_addr	object	Subnet object for destination address
src_port_range_start	int	Source port range start
src_port_range_end	int	Source port range end
dst_port_range_start	int	Destination port range start
dst_port_range_end	int	Destination port range end
ip_protocol	string	IP Protocol
address_family	string	IPv4 or IPv6 address family
direction	string	Direction of match, INGRESS or EGRESS
src_addr_range	object	Address Range object for source address
dst_add_range	object	Address Range object for destination address

Action

Attribute	Type	Description
type	string	Action type

Match Set

Attribute	Type	Description
src_set_id	string	Source set ID of Address Set object in the Network Policy Configuration `address_sets` array
dst_set_id	string	Destination set ID of Address Set object in the Network Policy Configuration `address_sets` array
src_ports	array	Array of Port Range objects for source ports
dst_ports	array	Array of Port Range objects for destination ports
ip_protocol	string	IP Protocol
address_family	string	IPv4 or IPv6 address family
direction	string	Direction of match, INGRESS or EGRESS

Address Set

Attribute	Type	Description
set_id	string	Address set ID
addr_ranges	array	Array of Address Range objects
subnets	array	Array of Subnet objects
addr_family	string	IPv4 or IPv6 address family

Subnet

Attribute	Type	Description
ip_addr	string	IP address
prefix_length	int	Prefix length for subnet

Address Range

Attribute	Type	Description
start_ip_addr	string	Start IP address for range
end_ip_addr	string	End IP address for range

Port Range

Attribute	Type	Description
start_port	int	Start port for range
end_port	int	End port for range

Concrete Policy Result

Attribute	Type	Description
byte_count	int	Byte count for concrete policy hits
pkt_count	int	Packet count for concrete policy hits

Timeseries Concrete Policy Result

Attribute	Type	Description
timestamp	string	Timestamp string for aggregation of results
result	object	Concrete Policy Result

Software Agents

Agent APIs

The software agents APIs are associated with managing Tetration software agents. These set of APIs require the `sensor_management` capability associated with the API key.

Note

These APIs are only available to site admin users.

Get software agents

This endpoint returns a list of software agents.

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

Parameters:

Name	Type	Description
limit	integer	Limits the number of results returned (optional)
offset	string	Offset is used for paginated requests. If response returns offset then subsequent request must use the same offset to get more results in the next page. (optional)

Deleting software agent

This endpoint is used to decommission a software agent given its UUID. This API must be used with

caution; once a sensor is deleted, it does not show up in the Tetration dashboard and if the sensor is active, flow exports from the sensor are not allowed in Tetration.

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

Software agent configuration using Intents

This API workflow uses few REST endpoints defined below.

Creating an inventory filter

This endpoint is used to specify criteria that match agent hosts on which user wants to configure software agents.

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

Parameters:

Name	Type	Description
app_scope_id	string	The scope ID to assign to the inventory filter.
name	string	A name for the inventory filter.
query	json	Filter or match criteria for agent host.

Sample python code

```
# 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.
```

Creating a software agent configuration profile

This endpoint is used to specify the set of configuration options to apply to target set of software agents.

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

Following configuration options can be specified as part of agent configuration profile:

- data_plane_disabled: if true, agent stops reporting flows to Tetration.
- enable_pid_lookup: if true, agent tries to attach process information to flows. Note this config option uses more CPU on the end host.
- cpu_quota_mode & cpu_quota_usec: these options are used to police the amount of CPU quota to give to agent on the end host.
- auto_upgrade_opt_out: if true, agents are not auto-upgraded during upgrade of Tetration cluster.
- enforcement_disabled: can be used to disable enforcement on hosts running enforcement agents.
- preserve_existing_rules: option to specify whether to preserve existing iptable rules.

For more details about the configuration options, refer to Software Agent Config

Sample python code

```
# 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)
```

Creating a software agent configuration intent

This endpoint is used to specify the intent to apply set of configuration options to specified set of software agents. This will create the intent and updates the intent order by adding the newly created intent to the order.

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

Sample python code

```
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.
```

Specifying order of intents

This endpoint is used to specify the ordering of various software agent configuration intents. For example, there could be two intents – one to enable process ID

lookup on development machines and second one to disable process ID lookup on windows machines. If the first intent has higher priority, then development windows machines will have process ID lookup enabled. NOTE: By default, when intent is created, it is added to the beginning of intent orders list. This endpoint is only to be used if end user needs to modify the existing order of intents.

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

Sample python code

```
# 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))
```

VRF configuration for agents behind NAT

Following set of APIs are useful to specify policies to assign VRFs to agents behind NAT boxes. These set of APIs require the `sensor_management` capability associated with the API key and are only available to site admin users.

List VRF configuration rules for agents behind NAT

This endpoint returns a list of VRF configuration rules applicable to agents behind NAT.

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

Create a new VRF configuration applicable to agents behind NAT

This endpoint is used to specify criteria for VRF tagging for hosts based on their source IP and source port as seen by Tetration appliance.

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

Parameters:

Name	Type	Description
src_subnet	string	Subnet to which source IP can belong to (CIDR notation).
src_port_range_start	integer	Lower bound of source port range (0-65535).
src_port_range_end	integer	Upper bound of source port range (0-65535).
vrf_id	integer	VRF ID to use for tagging flows for agents whose source address and port falls in the above specified range.

Sample python code

```
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
```

Delete existing VRF configuration

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

Previous

Next

Switches

The switch related APIs are associated with managing Tetration hardware agents. These set of APIs require the `hw_sensor_management` capability associated with the API key.

Note

These APIs are only available to site admin users.

Get switches

This endpoint returns a list of switches known to Tetration appliance.

```
GET /openapi/v1/switches
```

Parameters: None

Configure switch

This endpoint is used to configure a switch given its serial number.

```
PUT /openapi/v1/switches/{serial}
```

This API can be used to configure one or more of the following configuration options for a switch with specified serial number.

The query body consists of a json body with the following keys.

Keys	Values
datapath_disabled	Optional parameter. If true, switch stops reporting flows to Tetration
export_interval_ms	Optional parameter. Export interval to Tetration cluster
catchall_vrf_id	Optional parameter. Default Catch All Vrf Id

Sample python code

```
req_payload = {'export_interval_ms': 60000}
resp = restclient.put('/switches/%s' % switch_serial,
     json_body=json.dumps(req_payload))
print resp.status_code
```

Previous

Next

Collection Rules

These set of APIs can be used to manage collection rules. Collection rules in Tetration appliance are means for user to specify what IP addresses or subnets are interesting for their deployment. If the deployment has any switches that support Tetration analytics, then these collection rules are sent to the switches (user needs to check the 'Apply to switches' checkbox on the dashboard). On receiving these collection rules, switches only extract tarffic signals for IP addresses that match these sets of collection rules. These APIs require the `hw_sensor_management` capability associated with the API key.

Note

These APIs are only available to site admin users.

Collection rule object

The collection rule object attributes are described below:

Attribute	Type	Description
subnet	string	Subnet or IP address in CIDR format.
action	string	Possible values are 'include' or 'exclude'.

Update new collection rules for a VRF

This endpoint can be used to update the ordered list of collection rules for the specified VRF. Note, the list of

collection rules in the POST request is treated as an ordered list.

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

Parameters

Ordered list of collection rule objects in the POST body. **The last two rules must be catch all rules for IPv4 and IPv6.** The rules may specify the subnets 0.0.0.0/0 and ::/0 respectively, similar to the example below.

Sample python code

```
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))
```

Get collection rules for a VRF

This endpoint returns an ordered list of collection rules for a specified VRF.

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

Parameters: None

Previous

Next

User defined annotations

These APIs are used to add or remove user defined annotations that tag flows and inventory items on the Tetration appliance. To call these APIs, use an API key with the `user_data_upload` capability.

Note

Refer to Inventory > Uploads for instructions on accessing this functionality via the UI.

Scope dependent APIs

The following APIs are available to scope owners and site admins.

Upload annotations

This endpoint is used to upload a CSV file with annotations for tagging flows and inventory items in a scope on the Tetration appliance. A column header with name `IP` must appear in the CSV file. Of the remaining column headers, up to 32 can be used to annotate flows and inventory items.

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

Users needs to provide an operation type (`X-Tetration-Oper`) as a parameter to this API. `X-Tetration-Oper` can be one of `add` (for adding annotations) and `delete` (for removing annotations).

Sample python code

```
file_path = '/<path_to_file>/user_annotations.csv'
app_scope = 'Tetration'
req_payload = [tetpyclient.MultiPartOption(key='X-
Tetration-Oper', val='add')]
restclient.upload(file_path, '/assets/cmdb/upload/' +
appScope, req_payload)
```

Download user annotations

This endpoint returns the user uploaded annotations for a scope on the Tetration appliance.

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

Sample python code

```
file_path = '/<path_to_file>/output.csv'
app_scope = 'Tetration'
restclient.download(file_path, '/assets/cmdb/download'
+ app_scope)
```

Update list of annotated facets

This endpoint updates list of facets used for annotating flows and inventory items in a scope on the Tetration appliance.

```
PUT /openapi/v1/assets/cmdb/annotations/{appScope}
```

Sample python code

```
# the following list is a subset of column headers in
the
# uploaded CSV file
req_payload = ['location', 'region', 'detail']
app_scope = 'Tetration'
restclient.put('/assets/cmdb/annotations/' +
app_scope,
               json_body=json.dumps(req_payload))
```

Flush user uploaded annotations

This endpoint flushes annotations for flows and inventory items in a scope on the Tetration appliance. The changes affect new data; older annotated data remains unaltered.

```
POST /openapi/v1/assets/cmdb/flush/{appScope}
```

Sample python code

```
restclient.post('/assets/cmdb/flush/' + app_scope)
```

The following APIs are available to users with read access to a scope, scope owners and site admins:

Get list of annotated facets

This endpoint returns a list of annotated facets for a scope on the Tetration appliance. Annotated facets are a subset of column headers in the uploaded CSV file used for annotating flows and inventory items in that scope.

```
GET /openapi/v1/assets/cmdb/annotations/{appScope}
```

Sample python code

```
resp =
restclient.get('/assets/cmdb/annotations/{appScope}')
print resp.text
```

Scope independent APIs

The following APIs are only available to site admins.

Upload annotations

This endpoint is used to upload a CSV file with annotations for tagging flows and inventory items on the Tetration appliance. Column headers with names `IP` and `VRF` must appear in the CSV file. Of the remaining column headers, up to 32 can be used to annotate flows and inventory items.

```
POST /openapi/v1/assets/cmdb/upload
```

Users need to provide an operation type (`X-Tetration-Oper`) as a parameter to this API. The possible values for `X-Tetration-Oper` are `add` (for adding annotations) and `delete` (for removing annotations).

Sample python code

```
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)
```

Download user annotations

This endpoint returns the user uploaded annotations for all scopes on the Tetration appliance.

```
GET /openapi/v1/assets/cmdb/download
```

Sample python code

```
file_path = '/<path_to_file>/output.csv'
restclient.download(file_path, '/assets/cmdb/download')
```

Previous Next

VRFs

This set of APIs manages VRFs.

Note

These APIs are only available to site admins.

VRF object

The VRF object attributes are described below:

Attribute	Type	Description
id	string	Unique identifier for the VRF.
name	string	User specified name of the VRF.
tenant_id	string	ID of parent tenant.
root_app_scope_id	string	ID of associated root scope.
created_at	integer	Unix timestamp when the VRF was created.
updated_at	integer	Unix timestamp when the VRF was last updated.

Get VRFs

This endpoints returns a list of VRFs. This API is available to API keys with `sensor_management`, `flow_inventory_query` or `hw_sensor_management` capability.

```
GET /openapi/v1/vrfs
```

Parameters: None

Returns a list of VRF objects.

Create a VRF

This endpoint is used to create new VRFs. An associated root scope will automatically be created with a query matching the VRF ID. This API is available to API keys with `sensor_management` capability.

```
POST /openapi/v1/vrfs
```

Parameters:

Name	Type	Descrption
id	string	Unique identifier for the VRF.
tenant_id	string	ID of parent tenant.
name	string	User specified name of the VRF.
apply_monitoring_rules	boolean	Whether or not collection rules should be applied for the VRF. See Collection Rules for more information.

Returns the newly created VRF object.

Sample python code

```
req_payload = {
    "id": <vrf_id>,
    "tenant_id": <tenant_id>,
```

```
    "name": "Test",
    "apply_monitoring_rules": True
}
resp = restclient.post('/vrfs',
json_body=json.dumps(req_payload))
```

Update a VRF

This endpoint updates a VRF. This API is available to API keys with `sensor_management` capability.

```
PUT /openapi/v1/vrfs/{vrf_id}
```

Parameters:

Name	Type	Description
name	string	User specified name of the VRF.
apply_monitoring_rules	boolean	Whether or not collection rules should be applied to the VRF.

Returns the modified VRF object associated with specified ID.

Delete specific VRF

This endpoint deletes a VRF. It will fail if there are is an associated root scope. This API is available to API keys with `sensor_management` capability.

```
DELETE /openapi/v1/vrfs/{vrf_id}
```

Previous

Next

Orchestrators

This set of APIs can be used to manage external Orchestrator inventory learning in Tetration cluster deployment. They require the `external_integration` capability associated with the API key.

Currently supported Orchestrator types are 'vcenter' (VCenter 6.5 and later) and 'aws'.

Experimental Feature

This feature and its APIs are in **ALPHA** and are subject to changes and enhancements in future releases.

Orchestrator Object

The orchestrator object attributes are described below:

Attribute	Type	Description
id	string	Unique identifier for the orchestrator.
name	string	User specified name of the orchestrator.
type	string	Type of orchestrator - currently supported values `vcenter` and `aws`
description	string	User specified description of

		the orchestrator.
aws_region	string	AWS Region in which the cluster resides .
username	string	Username for the orchestration endpoint.
password	string	Password for the orchestration endpoint.
certificate	string	Client certificate used for authentication
key	string	Key corresponding to client certificate
ca_certificate	string	CA Certificate to validate orchestration endpoint
aws_access_key_id	string	AWS Access Key ID
aws_secret_access_key	string	AWS Secret Access Key
insecure	boolean	Turn off strict SSL verification
delta_interval	integer	Delta polling interval
full_snapshot_interval	integer	Full snapshot interval
verbose_tsdb_metrics	boolean	Per-Endpoint TSDB metrics
hosts_list	Array	Array of {host_name, port_number} to connect to the

		orchestrator

Get orchestrators

This endpoint returns a list of orchestrators known to Tetration appliance. This API is available to API keys with the `external_integration` capability.

```
GET /openapi/v1/orchestrator/{scope}
```

Parameters: None

Returns a list of orchestrator objects.

Create a orchestrator

This endpoint is used to create new orchestrators.

```
POST /openapi/v1/orchestrator/{scope}
```

Parameters:

Attribute	Type	Description
id	string	Unique identifier for the orchestrator.
name	string	User specified name of the orchestrator.
type	string	Type of orchestrator - currently supported values `vcenter` and `aws`
description	string	User specified description of the orchestrator.

aws_region	string	AWS Region in which the cluster resides .
username	string	Username for the orchestration endpoint.
password	string	Password for the orchestration endpoint.
certificate	string	Client certificate used for authentication
key	string	Key corresponding to client certificate
ca_certificate	string	CA Certificate to validate orchestration endpoint
aws_access_key_id	string	AWS Access Key ID
aws_secret_access_key	string	AWS Secret Access Key
insecure	boolean	Turn off strict SSL verification
delta_interval	integer	Delta polling interval
full_snapshot_interval	integer	Full snapshot interval
verbose_tsdb_metrics	boolean	Per-Endpoint TSDB metrics
hosts_list	Array	Array of {host_name, port_number} to connect to the orchestrator

Sample python code

```
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))
```

Get specific orchestrator

This endpoint returns an instance of a orchestrator.

```
GET /openapi/v1/orchestrator/{scope}/{orchestrator_id}
```

Returns the orchestrator object associated with the specified ID.

Update an orchestrator

This endpoint updates a orchestrator.

```
PUT /openapi/v1/orchestrator/{scope}/{orchestrator_id}
```

Parameters:

Same as POST parameters

Returns the modified orchestrator object associated with specified ID.

Delete specific orchestrator

This endpoint deletes the specified orchestrator.

```
DELETE
/openapi/v1/orchestrator/{scope}/{orchestrator_id}
```

Previous

Next