
Aria Automation Config API Documentation

Release 8.14.1

VMware, Inc.

December 01, 2023

CONTENTS

1	API (RaaS) RPC endpoint documentation	3
1.1	Introduction	3
1.2	API syntax	4
1.3	Using the API (RaaS) with Curl	4
1.4	Permissions	4
1.4.1	API Permission values	4
1.5	Licensing	7
1.6	RPC client	7
1.7	HTTP Bridge	7
1.8	RPC Endpoints	8
1.8.1	admin interface	8
1.8.2	api interface	11
1.8.3	audit interface	12
1.8.4	auth interface	14
1.8.5	background_jobs interface	22
1.8.6	cmd interface	22
1.8.7	conf interface	37
1.8.8	fs interface	38
1.8.9	job interface	40
1.8.10	kv interface	44
1.8.11	license interface	45
1.8.12	master interface	50
1.8.13	masterfs interface	55
1.8.14	minions interface	56
1.8.15	pillar interface	60
1.8.16	ret interface	62
1.8.17	schedule interface	65
1.8.18	sec interface	69
1.8.19	settings interface	94
1.8.20	stats interface	98
1.8.21	test interface	100
1.8.22	tgt interface	101
1.8.23	vman interface	105

Contents:

API (RAAS) RPC ENDPOINT DOCUMENTATION

1.1 Introduction

In this document, the API (RaaS) refers to the application server that Aria Config clients connect to. These clients include the user interface component of Aria Config, properly configured Salt controllers (formerly called the Salt masters), and other users of RaaS. The application server is also known as the eAPI server.

RaaS is organized into modules (called “resources”) and functions (called “methods”). In the API call `test.echo('hello, world')`, the resource is `test` and the method is `echo()`. Arguments can be passed to methods by position or by keyword.

RaaS can be accessed in two ways: via an RPC client and via an HTTP (or HTTPS) bridge.

To connect to RaaS, set these options in `client = APIClient()`:

- `server`
- `username`
- `password`
- `config_name='internal'` Change to authentication backend name if using LDAP
- `timeout=60` # Take at most 60 seconds to perform any operation
- `ssl_key=None`
- `ssl_cert=None`
- `ssl_context=None`
- `ssl_validate_cert=True` = Set to `False` if using self signed certs

Example:

```
from sseapiclient import APIClient
client = APIClient('https://localhost', 'root', 'PASSWORD', ssl_validate_cert=False)
```

Note: In versions prior to 6.2, the API connection was made using `SyncClient.connect()`, which is still compatible with versions 6.2 and up. If you are using `SyncClient.connect()`, no changes are required.

1.2 API syntax

`client.api.<interface>.<method>(parameter=parameter_value)`

Example:

```
from sseapiclient import APIClient
client = APIClient('https://localhost', 'root', 'PASSWORD')
client.api.sec.download_content(auto_ingest=True)
```

1.3 Using the API (RaaS) with Curl

You can also use RaaS with directly with HTTP and JSON. The following is an example of using the API with `curl`. Note when `xsrif` is enabled you will need to obtain an `xsrif` cookie and token first. See [HTTP Bridge](#) on how to obtain and use an `xsrif` token and cookie. When using the Python API client, this is done automatically by default.

Example:

```
curl --user 'root:PASSWORD' --url 'https://localhost/rpc' \
  --data '{
    "resource": "sec",
    "method": "download_content",
    "kwarg": {"auto_ingest": true}
  }'
```

1.4 Permissions

You can assign permissions to a role or user in the API (RaaS) using `save_role(...)` or `save_user(...)` in the [API AUTH interface](#). For more on roles and permissions in Aria Config, see [roles-permissions](#).

1.4.1 API Permission values

You can assign permissions to a role or user in the API (RaaS) using `save_role(...)` or `save_user(...)` in the [auth interface](#).

Permission value syntax

Permission values in the API (RaaS) include a resource type and an action, based on the following syntax:

- resource-action

Some permission values include a qualifier as follows:

- resource-qualifier-action

For example, if you want to assign permission to run commands, you would use `cmd-run`. Whereas, to assign permission to run *wheel* commands, you would use `cmd-wheel-run`.

Note: The above syntax does not apply to the *Super user* permission, whose API value is `superuser`.

API Permission values by resource

The following list includes all resource types and permitted actions.

For a detailed explanation of each resource type, and of permissions in Aria Config, see roles-permissions.

Commands

- cmd-delete
- cmd-read
- cmd-run
- cmd-write

Runner commands

- cmd-runner-run

SSH commands

- cmd-ssh-delete
- cmd-ssh-read
- cmd-ssh-run
- cmd-ssh-write

Wheel commands

- cmd-wheel-run

Formulas

- formula-delete
- formula-read
- formula-write

Filesystem

- fs-delete
- fs-read
- fs-write

Groups

- group-delete
- group-read
- group-write

Jobs

- job-delete
- job-read
- job-run
- job-write

License

- license-read

Salt controller

- master-delete
- master-read
- master-write

Salt controller configuration

- master-config-delete
- master-config-read
- master-config-write

Salt controller filesystem

- master-fs-delete
- master-fs-read
- master-fs-write

Minion

- minion-delete
- minion-read
- minion-write

Pillar

- pillar-delete
- pillar-read
- pillar-write

Returns

- returner-delete
- returner-read
- returner-write

Roles

- role-delete
- role-read
- role-write

Schedules

- schedule-delete
- schedule-read
- schedule-write

Super user

- superuser

Target

- target-delete

- target-read
- target-write
- target-allminions-run

Users

- user-delete
- user-read
- user-write

1.5 Licensing

The Aria Config web console displays notifications to warn when your license is close to expiring. As a RaaS user, you must use the License interface to track license status and ensure it stays active. If your license expires, the RaaS service stops. See [License interface](#).

1.6 RPC client

The programmatic RPC clients in the sseapiclient module work with Python version 2.7 and Python version 3.5 or later. The clients connect to SSE via HTTP or HTTPS and authenticate. Using the RPC client has the advantage of being somewhat easier to use than the HTTP bridge.

1.7 HTTP Bridge

The HTTP (or HTTPS) bridge accepts JSON payloads POSTed to an endpoint exposed by SSE, translates the payloads into RPC calls, then returns the result as JSON. The endpoint supports cookie-based authentication so that authentication credentials need to be passed only once per session. The bridge also allows sending multiple calls in a single payload.

If xsrf is enabled (default with state install) in the `/etc/raas/raas.conf` `tornado_xsrf_cookies_enabled: True` you will need to provide the X-XsrfToken: on the header of the rest call. The best way is to save a cookie with a get call then use the cookie to provide the header token. This cookie is saved in the \$HOME (users home) directory. The payload is a dictionary.

Example curl call with xsrf header:

```
curl -k -c $HOME/eAPICookie.txt -u root:PASSWORD 'https://localhost/account/login' >/dev/null
curl -k -u root:PASSWORD -b $HOME/eAPICookie.txt \
  -H 'X-XsrfToken: '$(grep -w '_xsrf' $HOME/eAPICookie.txt | cut -f7)'' \
  -X POST https://localhost/rpc \
  -d '{
    "resource": "sec",
    "method": "download_content",
    "kwarg": {"auto_ingest": true}
  }'
```

The samples assume:

- client=APIClient(<addr>,<user>,<pwd>)

- Default state based install of eAPI
- SSL enabled
- Import of sseapiclient. Example:

```
from sseapiclient import APIClient
```

1.8 RPC Endpoints

1.8.1 admin interface

The **LoadedMod** class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

delete_vra_params(...)

Returns <class 'uuid.UUID'>.

Delete VRA parameter record

Table 1: delete_vra_params() parameters

param_uuid		UUID of the parameter record to be deleted.
------------	--	---

Returns

UUID of the deleted parameter record.

Example using the Python client

```
from sseapiclient import APIClient
host = 'http://localhost'
user = 'root'
password = 'salt'
client = APIClient(host, user, password)
client.api.admin.delete_vra_params(
    param_uuid="f63e3ee3-2781-4e4d-a0ec-0ad2f1e9cc44")
```

Example response

```
RPCResponse(
  warnings=[],
  error=None,
  riq=139727129350384,
  ret="f63e3ee3-2781-4e4d-a0ec-0ad2f1e9cc44",
)
```

get_vra_params(...)

Returns Dict.

Get VRA parameters for Aria Config integration.

Table 2: get_vra_params() parameters

params_uuid		UUID of the parameter record. Optional. Can be passed to filter the response.
-------------	--	---

Returns

Dictionary of the parameter record.

Example using the Python client

```
from sseapiclient import APIClient
host = 'http://localhost'
user = 'root'
password = 'salt'
client = APIClient(host, user, password)
client.api.admin.get_vra_params(
    param_uuid="f63e3ee3-2781-4e4d-a0ec-0ad2f1e9cc44")
```

Example response

```
RPCResponse(
  warnings=[],
  error=None,
  riq=139727129350384,
  ret={
    "count": 1,
    "results": [
      "url": "https://cloud.vmware.com",
      "extra_params": {
        "vra": 8.3,
      },
      "param_uuid": "f63e3ee3-2781-4e4d-a0ec-0ad2f1e9cc44",
    ]
  }
)
```

save_vra_params(...)

Returns <class 'uuid.UUID'>.

Save VRA parameters for Aria Config integration.

Table 3: save_vra_params() parameters

url		URL of the VRA instance.
extra_params		Additional parameters from VRA can be stored in Aria Config (for future use).
param_uuid		UUID of the parameter record. Optional. Can be passed to update the param record.

Returns

UUID of the parameter record.

Example using the Python client

```
from sseapiclient import APIClient
host = 'http://localhost'
user = 'root'
password = 'salt'
client = APIClient(host, user, password)
client.api.admin.save_vra_params(
    url='https://cloud.vmware.com', extra_params={'vra': 8.3})
```

Example response

```
RPCResponse(
  warnings=[],
  error=None,
  riq=139727129350384,
  ret="f63e3ee3-2781-4e4d-a0ec-0ad2f1e9cc44",
)
```

trim_database(...)

Returns Dict.

Trim records from various database tables. Returns number of records of each type that would be deleted from the database. When `test=True` these records will not be deleted. If `test` is `None` or `False` this call will delete the records as well.

Table 4: trim_database() parameters

audit		Number of days of data from the audit trail tables to retain.
events		Number of days of data from the event tables to retain.
jobs		Number of days of data from the jobs tables to retain.
schedule		Number of days of schedule history to retain.
test		When <code>test=True</code> no records will be deleted.

Returns

Dictionary with `audit`, `events`, `jobs`, and/or `schedule` keys, matching the arguments passed to the call. Each key is associated with the number of corresponding records removed from the database, except `jobs`, which is a dictionary containing the number of commands, minions-expected, returns, and jids entries that were deleted.

These results are also logged in the RaaS log at `info` level.

Example using the Python client

```
from sseapiclient import APIClient
host = 'http://localhost'
user = 'root'
password = 'salt'
client = APIClient(host, user, password)
```

(continues on next page)

(continued from previous page)

```
client.api.admin.trim_database(
    audit=30, events=30, jobs=30, schedule=30, test=True)
```

Additional connect arguments: If you have a large dataset and you want to wait longer than 15 seconds (default timeout) for the request. `timeout=None`

Needed sometimes if your SSL Certificate is self signed. `ssl_validate_cert=False`

Example response

```
RPCResponse(
  warnings=[],
  error=None,
  riq=139727129350384,
  ret={
    "test": True,
    "jobs": {
      "commands": 5394,
      "minions-expected": 231,
      "returns": 4403,
      "jids": 299
    },
    "events": 30,
    "audit": 123,
    "schedule": 48
  }
)
```

1.8.2 api interface

The **LoadedMod** class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

discover(...)

Returns Dict.

Return the RPC resources, their methods and the methods documentation. The RPC client calls this method on each connection and stores the result in the client object in `_discovered_api`.

param dict or list include filter document to only include listed sections

include: {'<section-key>': '<sub-section-key>'} or ['<section1-key>']

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.api.discover()

client.api.api.discover(include=['constants'])

client.api.api.discover(include={'constants': ['permissions', 'command-states']})
```

`get_versions(...)`

Returns Dict.

Return information on various software components used by RaaS.

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.get_versions()
```

```
RPCResponse(
  riq=4,
  ret={'python': '3.5.3 (default, Sep 14 2017, 22:58:41)',
      'opts': {'sql': {'dialect': 'postgresql',
                      'host': 'localhost', 'pool_timeout': 10,
                      'driver': 'psycopg2', 'pool_recycle': 3600},
              'customer_id': '43cab1f4-de60-4ab1-85b5-1d883c5c5d09'},
      'raas': {'raas': '5.2.0-762-g3a64d15' ...}
  error=None,
  warnings=[])
```

1.8.3 audit interface

The `LoadedMod` class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

`get_audit(...)`

Returns Dict.

Return a list of audit records matching search criteria.

Table 5: get_audit() parameters

account_uuid		Account UUID to match on audit records. Cannot be combined with <code>username</code>
username		Username to match on audit records. Cannot be combined with <code>account_uuid</code>
session_uuid		Session UUID, to match retrieve audit records from a particular client session
event_type		'system', 'auth', 'rpc', or 'task'
event_name		For system events, 'startup' or 'shutdown'. For auth events, 'login' or 'logout'. For rpc events, '<resource>.<method>' of the API call. For task events, the name of the background task.
failed		True to match failed events, False to match successful events
daterange		List of two date strings in ISO 8601 format
elapsed_msec_min		Minimum event duration, in milliseconds
elapsed_msec_max		Maximum event duration, in milliseconds
page		Which page of the records to return (offset = page * limit)
limit		How many records to return at a time
sort_by		Field to sort by ('username', 'event_type', 'event_name', 'failed', 'start_time', 'end_time', 'elapsed_msec')
reverse		True to reverse sort order

```

client.api.audit.get_audit(username='root', event_type='auth', failed=True)

RPCResponse(
  riq=4,
  ret={'count': 1,
      'results': [
        {'id': 175,
         'start_time': '2019-06-05T21:12:53.190506+00:00',
         'end_time': '2019-06-05T21:13:24.729561+00:00',
         'event_type': 'auth',
         'event_name': 'login',
         'username': 'root',
         'account_uuid': None,
         'session_uuid': '4642ef92-d5ac-4ea2-b95d-2055b43f2193',
         'failed': True,
         'event_data': {
           'note': 'User root: authentication failed: invalid basic-auth❌
→credentials',
           'status': 401,
           'headers': {
             'Host': 'localhost:8080',
             'Accept': '*/*',
             'User-Agent': 'curl/7.65.0',
             'Content-Type': 'application/x-www-form-urlencoded',
             'Authorization': 'Basic cm9vdDpzYWx0MQ==',
             'Content-Length': '41'}}}},
  error=None,
  warnings=[])

```

1.8.4 auth interface

The `LoadedMod` class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

`change_password(...)`

Returns None.

Update a password for an account_uuid.

`delete_group(...)`

Returns None.

Delete a group from the provided named configuration

Table 6: `delete_group()` parameters

<code>config_name</code>		Name of the authentication backend to which this group belongs
<code>group_name</code>		Name of the group to be deleted.
<code>group_uuid</code>		UUID for this group. When calling this endpoint, use either <code>group_uuid</code> or <code>config_name</code> and <code>group_name</code> to uniquely identify a group.

`delete_group_link(...)`

Returns None.

Delete the link between an external auth backend group and an Aria Automation Config (AAC) internal group.

Table 7: `delete_group_link()` parameters

<code>group_name</code>		Name of the AAC group to be linked to the external group
<code>group_uuid</code>		UUID of the AAC group. Pass either <code>group_uuid</code> or <code>group_name</code> to uniquely identify the AAC group.
<code>config_name</code>		Name of the authentication backend of the external group
<code>ext_group_name</code>		Name of the external group to be linked to the AAC group
<code>ext_group_uid</code>		Unique ID of the external group. Pass <code>config_name</code> and either <code>ext_group_name</code> or <code>ext_group_uid</code> to uniquely identify an external group.

delete_role(...)

Returns None.

Delete a role from the system.

Table 8: delete_role() parameters

role_name		Name of the group to be deleted.
role_uuid		UUID for this group. When calling this endpoint, use either role_uuid or role_name to uniquely identify a role.

delete_user(...)

Returns None.

Delete a user account.

Table 9: delete_user() parameters

config_name		Name of the authentication backend to which this user account belongs
username		User's login name.
account_uuid		UUID for this account. When calling this endpoint, use either account_uuid or config_name and username to uniquely identify an account.

get_all_groups(...)

Returns Dict.

Return all groups from the provided config_name or all groups from all authentication configurations if config_name is None

Table 10: get_all_groups() parameters

config_name		Authentication backend name (internal, ldap, etc.)
include_users		Include users belonging to each group
include_custom_data		Include any custom data assigned to each group
sort_by		Sort by this field, currently only 'name' is supported
reverse		Pass True to sort results in descending order
limit		Limit results to this many users (default is 50, pass 0 for unlimited)
page		Return users from this page (offset = page * limit)

```
client.api.auth.get_all_groups(config_name='internal', include_users=True)
```

```
RPCResponse(riq=4,
             ret={'count': 2,
                  'results': [{ 'uuid': '9e0fb921-14fd-45ab-9def-b777711d5cfc',
                                'config_name': 'internal',
                                'remote_uuid': None,
```

(continues on next page)

(continued from previous page)

```

        'name': 'group1',
        'desc': None,
        'users': [{ 'uuid': 'f6cdb715-2e83-455c-ba83-8c5059f5ed41'
                    'config_name': 'internal',
                    'username': 'user1',
                    'email': None,
                    'remote_uid': None,
                    'perms': [],
                    'roles': ['User'],
                    'groups': ['group1']}],
        'roles': ['role1']},
    { 'uuid': None,
      'config_name': None,
      'remote_uid': None,
      'name': None,
      'desc': None,
      'users': [{ 'uuid': '8025afa2-929f-4860-b378-658bac410abb'
                  'config_name': 'internal',
                  'username': 'deleted',
                  'email': None,
                  'remote_uid': None,
                  'perms': [],
                  'roles': ['User'],
                  'groups': []}],
      { 'uuid': '80c67364-cb31-4f4b-972a-e7ea3f752bb8'
        'config_name': 'internal',
        'username': 'root',
        'email': None,
        'remote_uid': None,
        'perms': [],
        'roles': ['User', 'Superuser'],
        'groups': []}],
      { 'uuid': 'ad3d6d8f-06b6-42b4-80d2-af3af32b0db0'
        'config_name': 'internal',
        'username': 'master_master1',
        'email': None,
        'remote_uid': None,
        'perms': [],
        'roles': ['Salt Master', 'User'],
        'groups': [{}]}]}

    error=None,
    warnings=[])

```

get_all_roles(...)

Returns Dict.

Retrieve details about all roles in the system.

Example:

get_all_users(...)

Returns Dict.

Get all users for the provided `config_name` or all users of all authentication configurations if `config_name` is `None`

For more information on users, see the Aria Automation Config documentation on VMware's Doc Center.

Table 11: `get_all_users()` parameters

<code>config_name</code>		Authentication backend name (<code>internal</code> , <code>ldap</code> , etc.)
<code>include_roles</code>		Include roles assigned to this user
<code>include_inherited_roles</code>		Include roles inherited via group membership
<code>include_perms</code>		Include permissions assigned to this user
<code>include_groups</code>		Include groups to which this user belongs
<code>include_custom_data</code>		Include any custom data assigned to this user
<code>include_deleted_user</code>		Include the user account representing deleted users
<code>sort_by</code>		Sort by this field, either 'username' or 'email'
<code>reverse</code>		Pass True to sort results in descending order
<code>limit</code>		Limit results to this many users (default is 50, pass 0 for unlimited)
<code>page</code>		Return users from this page (offset = page * limit)

```
client.api.auth.get_all_users(config_name='internal', include_roles=True)

RPCResponse(riq=4,
  ret={'count': 3,
    'results': [{ 'uuid': 'ad3d6d8f-06b6-42b4-80d2-af3af32b0db0',
      'config_name': 'internal',
      'username': 'master_master1',
      'email': None,
      'remote_uid': None,
      'roles': ['User', 'Salt Master']
    },
    { 'uuid': '80c67364-cb31-4f4b-972a-e7ea3f752bb8',
      'config_name': 'internal',
      'username': 'root',
      'email': None,
      'remote_uid': None,
      'roles': ['User', 'Superuser']
    },
    { 'uuid': 'f6cdb715-2e83-455c-ba83-8c5059f5ed41',
      'config_name': 'internal',
      'username': 'user1',
      'email': None,
      'remote_uid': None,
```

(continues on next page)

(continued from previous page)

```
        'roles': ['User', 'role1']
    }
]
},
error=None,
warnings=[])
```

get_auth_pubkey(...)

Returns Dict.

Get the public key used for key authentication to this raas instance.

get_group(...)

Returns Dict.

Retrieve information about a group.

Table 12: get_group() parameters

config_name		Name of the authentication backend to which this group account belongs
group_name		Name of the group
group_uuid		UUID for this group. When calling this endpoint, use either group_uuid or config_name and group_name to uniquely identify a group.
include_users		If this parameter is true, include group members in the return.

get_group_links(...)

Returns List[Dict].

Get information on links between external auth backend groups and Aria Automation Config (AAC) internal groups.

Table 13: get_group_links() parameters

group_name		AAC group name (substring match)
group_uuid		AAC group UUID
config_name		Name of the authentication backend (substring match)
ext_group_name		External group name (substring match)
ext_group_uid		External group unique ID

get_jwt(...)

Returns Dict.

Get a JSON Web Token for the current user.

get_role(...)

Returns Dict.

Retrieve details about a particular role. For more information on roles and how they function, see the Aria Automation Config documentation on VMware's Doc Center.

Table 14: get_role() parameters

role_name		Name of the role to retrieve.
role_uuid		UUID of the role to retrieve. Use role_name or role_uuid but not both.

Example:

get_user(...)

Returns Dict.

Get details for a user account. For more information on users, see the Aria Automation Config documentation on VMware's Doc Center.

Table 15: get_user() parameters

account_uuid		UUID of the user for which the call should retrieve details. This parameter cannot be combined with username
username		Username of the account for which the call should retrieve details. This parameter cannot be combined with account_uuid, and requires that config_name also be passed
config_name		Authentication/Authorization backend name (internal, ldap, etc.)

```
client.api.auth.get_user(account_uuid='80c67364-cb31-4f4b-972a-e7ea3f752bb8')
RPCResponse(riq=12,
  ret={'uuid': '80c67364-cb31-4f4b-972a-e7ea3f52bb8',
    'groups': [],
    'perms': ['cmd-read', 'master-config-read',
      'job-run', 'fs-read', 'cloud-read',
      'job-read', 'superuser', 'returner-read',
      'metadata-auth-read', 'target-read',
      'license-read', 'master-fs-read',
      'minion-read', 'master-read'],
    'config_name': 'internal',
    'custom_data': None,
    'roles': ['Superuser', 'User'],
    'username': 'root'}, error=None, warnings=[])
```

get_ws_ticket(...)

Returns <class 'str'>.

Get a ticket for establishing a websocket connection.

save_group_link(...)

Returns None.

Link an external auth backend group to an Aria Automation Config (AAC) internal group. Users in the external group will inherit the roles assigned to the AAC group.

Table 16: save_group_link() parameters

group_name		Name of the AAC group to be linked to the external group
group_uuid		UUID of the AAC group. Pass either <code>group_uuid</code> or <code>group_name</code> to uniquely identify the AAC group.
config_name		Name of the authentication backend of the external group
ext_group_name		Name of the external group to be linked to the AAC group
ext_group_uuid		Unique ID of the external group. Pass <code>config_name</code> and either <code>ext_group_name</code> or <code>ext_group_uuid</code> to uniquely identify an external group.

save_group(...)

Returns ['None', 'typing.Dict'].

Save a group. This call supports both creating groups and updating group information.

Table 17: save_group() parameters

config_name		Name of the authentication backend to which this group belongs
group_name		Name of this group. Changing a group name is only supported for the <code>internal</code> authentication backend.
group_uuid		UUID for this account. When calling this endpoint, use either <code>group_uuid</code> or <code>config_name</code> and <code>group_name</code> to uniquely identify a group.
description		Text describing the group purpose.
roles		A list of roles to assign to this group.
custom_data		A dictionary containing arbitrary data. Authentication backends can use this to store information to assist in aligning groups in RaaS with groups in the backend. For example, the Active Directory backend uses it to store a group's Distinguished Name.
remote_uuid		Externally defined group unique id.

save_role(...)

Returns None.

Update an existing or create a new role.

For more information on roles and how they function, see the Aria Config Config documentation on VMware's Doc Center.

Table 18: save_role() parameters

role_name		Name of the role in question.
role_uuid		UUID for this role.
perms		A list of permissions to assign to this role.
description		Text to describe nature and purpose of the role.

save_user_link(...)

Returns None.

Create a link between an external user and an internal RaaS user. Will create an internal user record if necessary.

Table 19: save_user_link() parameters

config_name		Name of the authentication backend (substring match)
username		Username to link
user_dn		DistinguishedName in the external directory

save_user(...)

Returns <class 'uuid.UUID'>.

Save a user account. This call supports both creating accounts and updating user account information.

For more information on users, see the Aria Automation Config documentation on VMware's Doc Center.

Table 20: save_user() parameters

config_name		Name of the authentication backend to which this user account belongs
username		User's login name. Changing a username is only supported for the <code>internal</code> authentication backend.
password		Account password. (account creation only)
account_uuid		UUID for this account. When calling this endpoint, use either <code>account_uuid</code> or <code>config_name</code> and <code>username</code> to uniquely identify an account.
perms		A list of permissions to assign to this account.
roles		A list of roles to assign to this account.
groups		A list of groups to assign to this account
custom_data		A dictionary containing arbitrary data. Authentication backends can use this to store information to assist in aligning users in RaaS with users in the backend. For example, the Active Directory backend uses it to store the users Distinguished Name.

`transfer_resources(...)`

Returns Dict.

Transfer resources that belong to one user to another.

Table 21: `transfer_resources()` parameters

<code>from_user_uuid</code>		UUID of the user whose resources are to be transferred.
<code>to_user_uuid</code>		UUID of the user to whom the resources are to be transferred.
<code>resource_types</code>		The type of the resource that is to be transferred (Optional). <code>target_groups</code> , <code>jobs</code> , <code>files</code> , <code>pillars</code> , <code>auth_configs</code> , <code>formulas</code> are valid inputs.
<code>resource_uuids</code>		The UUID of the resources that are to be transferred. (Optional).
<code>revoke_access</code>		Revoke access on the resource after transfer.
<code>test</code>		Return the number of resources to be updated without updating ownership.

1.8.5 `background_jobs` interface

The `LoadedMod` class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

`get_background_job(...)`

Returns Dict.

Return the status information about a background running job

`get_task_status(...)`

Returns List.

Get the status of background tasks

`task_ids` list of task IDs

1.8.6 `cmd` interface

The `LoadedMod` class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

`get_cmd_details(...)`

Returns Dict.

Query commands based on `jid` and retrieve more detailed data.

jid A job ID to match against.

minion_id_exact Filter by a `minion_id` that match exactly. Also full return data will be returned.

minion_id Filter by a `minion_id` search.

master_id Filter by a `master_id` search.

has_errors Filter by whether there were errors, `True|False`.

has_return Filter by whether return data was recieved, `True|False`.

sort_by Sort by the specified field, such as `'alter_time'`.

reverse Sort ascending (`False`) or descending (`True`).

page This specifies which page, after the first, of results to return. Often used with `limit`.

limit This is how many records to return at a time. The default is 50. Often used with `page`.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.cmd.get_cmd_details(
    jid="20190726204734431402"
)
```

`get_cmd_status(...)`

Returns List.

Get the status of a command

jids list of JIDs

`get_cmds(...)`

Returns Dict.

Query commands based on a number of properties.

For more information on commands and jobs, see the Aria Automation Config documentation on VMware's Doc Center.

Table 22: get_cmds() parameters

cmd		A command name to match against.
daterange		A comma-separated list of two date strings in ISO 8601 format.
filter_find_job		Exclude commands referring to saltutil.find_job.
filter_refresh_grains		Exclude commands referring to saltutil.refresh_grains.
fun		Command (function) name that was called for this job.
highstate		True if the job submitted was call to state.highstate
include_adhoc		When this is true, you can search jobs that are running (in-flight), but were not scheduled via the scheduler API (ad-hoc). This parameter is deprecated. Use include_adhoc_scheduled instead. Until the removal of this parameter, it will take precedence over include_adhoc_scheduled if it is present. If it is not present or is passed a value of None, the value of include_adhoc_scheduled will be honored.
include_adhoc_scheduled		One of all, scheduled, or adhoc. Filters jobs based on if they are scheduled, unscheduled (aka “on the fly” or “ad-hoc”, or show all jobs regardless of what started them.)
limit		This is how many records to return at a time. The default is 50. Often used with page.
jid		A job ID to match against.
job_names		A list of job names to match against.
job_uuid		Every job gets a jid and a UUID. This is the uuid for the job.
master_id		Master ID
page		This specifies which page, after the first, of results to return. Often used with limit.
reverse		Sort ascending (False) or descending (True).
sched_names		A list of schedule names to match against.
sched_uuid		A schedule UUID to match against.
sort_by		Sort by the specified field, such as ‘start_time’.
state_invert		Return commands that do not have the specified state.
state		Find commands in one of these states, 'new', 'retrieved', 'skipped', 'pausing', 'paused', 'resuming', 'resumed', 'completed_missing_returns', 'completed_failures', 'completed_all_successful', 'stopping', 'stopped', or 'disabled'. Note that state may contain completed when no minions have returned.
tgt_names		A list of target names to match against.
tgt_uuid		Commands can be targeted directly or via a saved target group. This is the UUID of the target group used to submit the command.
users		A list of users to match against.

Example:

```
client.api.cmd.get_cmds()

RPCResponse(riq=5,
  ret={'count': 1,
    'results': [
      {'cmd': 'local',
        'duration': None,
        'expected': 2,
        'fun': 'disk.usage',
        'is_highstate': False,
        'jid': '20190416202619938862',
        'job_desc': None,
        'job_name': None,
        'job_source': 'raas',
        'job_uuid': None,
        'masters_done': ['master1'],
        'masters_to': ['master1'],
        'origination': 'Ad-Hoc',
        'returned': 1,
        'returned_failed': 0,
        'returned_good': 1,
        'sched_name': None,
        'sched_uuid': None,
        'start_time': '2019-04-16T20:26:19.93886Z',
        'state': 'retrieved',
        'tgt_desc': None,
        'tgt_name': None,
        'tgt_uuid': None,
        'user': 'root',
        'user_uuid': '80c67364-cb31-4f4b-972a-e7ea3f752bb8'}]},
  error=None,
  warnings=[])
```

get_highstate_minion_details(...)

Returns Dict.

Get the most recent highstate details for a particular minion

Table 23: get_highstate_minion_details() parameters

param minion_id		Get highstate details for this minion (required, exact match)
param state		Filter results by matching states (substring match)
param sls		Filter results by matching SLS files (substring match)
param saltenv		Filter results by matching saltenv (substring match)
param jid		Filter results by jid (substring match)
param result		Report only successful (True) or failed (False) states
param changes		Report only states that resulted in changes (True) or no changes (False)
param comment		Report only states with matching comments (substring match)
param test		Report only results from dry runs (True), default is to exclude
param filter_summary		Pass True to filter summary data by query args (default False)
param sort_by: One of		state, sls, saltenv, jid, run_num, result, changes, duration, comment
param reverse		Pass True to sort descending
param page		Specify which page of results to return, default is 0 (first page)
param limit		How many records to return per call, default is 50

Example:

```
client.api.cmd.get_highstate_minion_details(minion_id='dev01', limit=3)
```

```
RPCResponse(riq=7,
  ret={
    'summary': {
      'total_states': 9,
      'states_failed': 0,
      'states_succeeded': 9,
      'states_changed': 0,
      'states_unchanged': 9,
      'total_duration': 63264.21
    },
    'count': 9,
    'results': [
      {
        'state': 'install_apache',
        'sls': 'web',
        'saltenv': 'pre',
        'jid': '20220828231050673988',
        'run_num': 0,
        'result': True,
        'duration': 11043.26,
        'num_changes': 0,
        'changes': {},
        'comment': 'No changes made.'
      },
      {
        'state': 'ensure_service_running',
        'sls': 'web',
```

(continues on next page)

(continued from previous page)

```
        'saltenv': 'pre',
        'jid': '20220828231050673988',
        'run_num': 1,
        'result': True,
        'duration': 10009.25,
        'num_changes': 0,
        'changes': {},
        'comment': 'No changes made.'
    },
    {
        'state': 'default_html_page',
        'sls': 'web',
        'saltenv': 'pre',
        'jid': '20220828231050673988',
        'run_num': 2,
        'result': True,
        'duration': 9023.58,
        'num_changes': 0,
        'changes': {},
        'comment': 'No changes made.'
    }
]
},
error=None,
warnings=[])
```

get_highstate_minions(...)

Returns Dict.

Get minion information from the most recent highstate runs

Table 24: get_highstate_minions() parameters

param tgt_uuid		Report minions in this target group
param master_id		Report minions on this master (substring match)
param minion_id		Report minions with matching ids (substring match)
param saltenv		Report minions with results from this saltenv (substring match)
param jid		Report minions that reported results for this jid (substring match)
param result		Report only successful (True) or failed (False) minions
param changes		Match only minions reporting changes (True) or no changes (False)
param test		Report only results from dry runs (True), default is to exclude
param age_limit		Ignore data older than this (minutes), default is no age limit
param filter_summary		Pass True to filter summary data by query args (default False)
param sort_by: One of		minion_id, total_states, states_failed, states_succeeded, states_changed, states_unchanged, total_duration
param reverse		Pass True to sort descending
param page		Specify which page of results to return, default is 0 (first page)
param limit		How many records to return per call, default is 50

Example:

```
client.api.cmd.get_highstate_minions(limit=1)

RPCResponse(riq=4,
  ret={
    'summary': {
      'total_minions': 97,
      'avg_states_per_minion': 9.71,
      'minions_failed': 19,
      'minions_succeeded': 78,
      'minions_changed': 3,
      'minions_unchanged': 94,
      'avg_duration': 64888.62
    },
    'count': 97,
    'results': [
      {
        'master_uuid': '1d19cf58-cb2f-4190-8f1d-32e462bbd91c',
        'master_id': 'master_dev',
        'minion_id': 'dev01',
        'jids': ['20220828231050673988'],
        'saltenvs': ['pre'],
        'total_states': 9,
        'states_failed': 0,
        'states_succeeded': 9,
        'states_changed': 0,
        'states_unchanged': 9,
        'total_duration': 63264.21
      }
    ]
  })
```

(continues on next page)

(continued from previous page)

```

    }
  ]
},
error=None,
warnings=[]
)

```

get_highstate_state_details(...)

Returns Dict.

Get the most recent highstate details for a particular state

Table 25: get_highstate_state_details() parameters

param state		Get highstate details for this state (required, exact match)
param master_id		Filter results by master id (substring match)
param minion_id		Filter results by minion id (substring match)
param result		Report only successful (True) or failed (False) minions
param changes		Report only minions that made changes (True) or no changes (False)
param comment		Report only minions with matching comments (substring match)
param test		Report only results from dry runs (True), default is to exclude
param filter_summary		Pass True to filter summary data by query args (default False)
param sort_by: One of		minion_id, saltenv, jid, result, changes, duration, comment
param reverse		Pass True to sort descending
param page		Specify which page of results to return, default is 0 (first page)
param limit		How many records to return per call, default is 50

Example:

```

client.api.cmd.get_highstate_state_details('ensure_service_running', limit=3)

RPCResponse(rq=13,
  ret={
    'summary': {
      'total_minions': 97,
      'minions_failed': 0,
      'minions_succeeded': 97,
      'minions_changed': 3,
      'minions_unchanged': 94,
      'avg_duration': 10047.06
    },
    'count': 97,
    'results': [
      {
        'master_id': 'master_dev',

```

(continues on next page)

(continued from previous page)

```
        'minion_id': 'dev01',
        'jid': '20220828231050673988',
        'saltenv': 'pre',
        'result': True,
        'duration': 10009.25,
        'num_changes': 0,
        'changes': {},
        'comment': 'No changes made.'
    },
    {
        'master_id': 'master_dev',
        'minion_id': 'dev02',
        'jid': '20220828231050673988',
        'saltenv': 'pre',
        'result': True,
        'duration': 10007.84,
        'num_changes': 0,
        'changes': {},
        'comment': 'No changes made.'
    },
    {
        'master_id': 'master_dev',
        'minion_id': 'dev03',
        'jid': '20220828231050673988',
        'saltenv': 'pre',
        'result': True,
        'duration': 10084.85,
        'num_changes': 0,
        'changes': {},
        'comment': 'No changes made.'
    }
]
},
error=None,
warnings=[])
```

get_highstate_states(...)

Returns Dict.

Get state information from the most recent highstate runs

Table 26: get_highstate_states() parameters

param state		Report states with matching names (substring match)
param sls		Report states from matching sls files (substring match)
param saltenv		Report results run against matching saltenvs (substring match)
param jid		Report results from jid (substring match)
param result		Report only successful (True) or failed (False) states
param changes		Match only states reporting changes (True) or no changes (False)
param test		Report only results from dry runs (True)
param age_limit		Ignore data older than this (minutes), default is no age limit
param filter_summary		Pass True to filter summary data by query args (default False)
param sort_by: One of		state, sls, total_minions, minions_failed, minions_succeeded, minions_changed, minions_unchanged, avg_duration
param reverse		Pass True to sort descending
param page		Specify which page of results to return, default is 0 (first page)
param limit		How many records to return per call, default is 50

Example

```

client.api.cmd.get_highstate_states(limit=3)

RPCResponse(riq=10,
  ret={
    'summary': {
      'total_states': 10,
      'avg_minions_per_state': 97.0,
      'states_failed': 1,
      'states_succeeded': 9,
      'states_changed': 7,
      'states_unchanged': 3,
      'avg_duration': 6547.77
    },
    'count': 10,
    'results': [
      {
        'state': 'clean_up',
        'sls': 'app',
        'total_minions': 70,
        'minions_failed': 0,
        'minions_succeeded': 70,
        'minions_changed': 0,
        'minions_unchanged': 70,
        'avg_duration': 2040.62,
        'jids': ['20220829081050673988'],
        'saltenvs': ['prod']
      },
      {
        'state': 'default_html_page',

```

(continues on next page)

(continued from previous page)

```

        'sls': 'web',
        'total_minions': 100,
        'minions_failed': 0,
        'minions_succeeded': 100,
        'minions_changed': 3,
        'minions_unchanged': 97,
        'avg_duration': 9048.9,
        'jids': ['20220828231050673988', '20220829003551779461'],
    ],
    'saltenvs': ['pre', 'prod']
},
{
    'state': 'ensure_mysql_running',
    'sls': 'db',
    'total_minions': 100,
    'minions_failed': 20,
    'minions_succeeded': 80,
    'minions_changed': 3,
    'minions_unchanged': 97,
    'avg_duration': 5047.29,
    'jids': ['20220828231050673988', '20220829003551779461'],
    'saltenvs': ['pre', 'prod']
}
]
},
error=None,
warnings=[])

```

get_highstate_stats(...)

Returns Dict.

Get daily historical highstate summary statistics.

Table 27: get_highstate_stats() parameters

param begin_date: Get statistics beginning at this date (default		earliest available).
param end_date: Get statistics ending at this date (default		today).
param reverse		Pass True to sort from most recent to least recent

Example:

```

client.api.cmd.get_highstate_stats(begin_date='2022-08-31', reverse=True)

RPCResponse(riq=16,
  ret={
    'count': 3,
    'results': [
      {
        'date': '2022-09-02',

```

(continues on next page)

(continued from previous page)

```

        'total_minions': 107,
        'minions_changed': 0,
        'minions_unchanged': 107,
        'minions_succeeded': 107,
        'minions_failed': 0,
        'minion_avg_duration': 32100.0,
        'total_states': 10,
        'states_changed': 2,
        'states_unchanged': 8,
        'states_succeeded': 8,
        'states_failed': 2
    },
    {
        'date': '2022-09-01',
        'total_minions': 107,
        'minions_changed': 0,
        'minions_unchanged': 107,
        'minions_succeeded': 107,
        'minions_failed': 0,
        'minion_avg_duration': 32097.0,
        'total_states': 10,
        'states_changed': 0,
        'states_unchanged': 10,
        'states_succeeded': 10,
        'states_failed': 0
    },
    {
        'date': '2022-08-31',
        'total_minions': 107,
        'minions_changed': 1,
        'minions_unchanged': 106,
        'minions_succeeded': 107,
        'minions_failed': 0,
        'minion_avg_duration': 32106.0,
        'total_states': 10,
        'states_changed': 0,
        'states_unchanged': 10,
        'states_succeeded': 10,
        'states_failed': 0
    }
]
},
error=None,
warnings=[]
)

```

pause_job(...)

Returns <class 'str'>.

Pass a pause command through to the masters. Note that the pause command is itself a command, so this method returns a jid corresponding to the `state.pause` call.

param job_uuid UUID of the job to attempt to pause

param jid Jid of the job to attempt to pause

return a Jid corresponding to the pause command

resume_job(...)

Returns <class 'str'>.

Pass a resume command through to the masters

param job_uuid UUID of the job to attempt to resume

param jid Jid of the job to attempt to resume

return a Jid corresponding to the resume command

route_cmd(...)

Returns <class 'str'>.

Create a command and route it to Salt. For more information on commands and jobs, see the Aria Automation Config documentation on VMware's Doc Center.

Table 28: route_cmd() parameters

job_uuid		UUID for this job, can be <code>None</code> in which case RaaS will generate a new UUID.
cmd		One of <code>local</code> (for targeting minions), <code>runner</code> (master-level command), or <code>wheel</code> (master wheel calls)
fun		Dotted-notation function to run (e.g. <code>test.ping</code> or <code>network.ipaddrs</code>)
masters		A list of master names that should receive this command. Applicable only for <code>runner</code> and <code>wheel</code> commands. For <code>local</code> cmds all targeting is specified in <code>tgt</code> .
arg		A dictionary containing the keys <code>arg</code> and/or <code>kwarg</code> representing arguments to pass to the <code>fun</code> being called.
tgt		A dictionary containing targeting information. A target can contain a master name of <code>*</code> to indicate all Aria Automation Config-connected masters. If <code>*</code> is present, it must be the only master listed in <code>tgt</code> .
tgt_uuid		The UUID of an existing target group that should receive this command.
jid		Optional jid for the command. If not supplied, a jid will be generated.
disposition		Command disposition. See the discussion below.

Either `tgt` or `tgt_uuid` should be used, but not both.

`tgt` has the following form:

```
{
  name_of_master: {
    "tgt": target_string,
    "tgt_type": target_type
  },
  another_master: {
    "tgt": target_string,
    "tgt_type": target_type
  },
  ...
}
```

The `disposition` parameter determines how runner and wheel commands are routed. The default behavior is to route these commands to each entity represented in the command's `masters` list. If any such entity is a cluster-id instead of a salt-master id, the command will be routed to one master in that cluster. This behavior corresponds to the disposition value `["all-masters", "any-in-cluster"]`.

In other words, the string `"all-masters"` causes the command to be routed to all the entities represented in the command's `masters` list, and the string `"any-in-cluster"` means that when the command is routed to a cluster, one arbitrary salt-master from that cluster will execute the command.

To change the routing of a command, two other strings can be included in the `disposition` list.

- Passing `"any-master"` causes the command to be routed to a single arbitrary entity from the command `masters` list.
- Passing `"all-in-cluster"` causes the command to be routed to all salt-masters within each cluster receiving the command.

Here are some `route_cmd()` examples with descriptions of how the commands are routed:

- `route_cmd(..., masters=["master1", "master2"])` routes the command to both salt-masters
- `route_cmd(..., masters=["m1", "m2"], disposition=["any-master"])` routes the command to salt-master `m1` or salt-master `m2`
- `route_cmd(..., masters=["master_dev", "cluster_prod"])` routes the command to the `master_dev` salt-master and to one arbitrary salt-master within the `cluster_prod` cluster
- `route_cmd(..., masters=["cluster1"], disposition=["all-in-cluster"])` routes the command to all salt-masters within the `cluster1` cluster
- `route_cmd(..., masters=["c1", "c2"], disposition=["any-master", "all-in-cluster"])` routes the command to cluster `c1` or cluster `c2`, and within the chosen cluster, all salt-masters will receive the command.
- `route_cmd(..., masters=["m1", "c1"], disposition=["any-master", "all-in-cluster"])` routes the command either to salt-master `m1` or to cluster `c1`. If the command is routed to cluster `c1`, all salt-masters within that cluster will receive it.

General `route_cmd()` examples follow.

Example:

```
client.api.cmd.route_cmd(
    cmd="local",
    fun="cmd.run",
    tgt={"master3_master": {"tgt": "master3", "tgt_type": "glob"}},
```

(continues on next page)

(continued from previous page)

```
    arg={"arg": ["ls /etc"]},
)

client.api.cmd.route_cmd(
    job_uuid="5c5cc410-4f9f-11e6-88bc-080027a7289c",
    tgt={"master3_master": {"tgt": "*", "tgt_type": "glob"}},
)
```

Example with an argument:

```
client.api.cmd.route_cmd(
    cmd="local",
    fun="cmd.run",
    tgt={"master3_master": {"tgt": "master3", "tgt_type": "glob"}},
    arg={"arg": ["ls /etc"]},
)
```

Example of a state execution with pillar data:

```
client.api.cmd.route_cmd(
    cmd="local",
    fun="state.apply",
    tgt={"master3_master": {"tgt": "master3", "tgt_type": "glob"}},
    arg={"arg": ["my_state"], "kwarg": {"pillar": {"key": "value"}}},
)
```

Example of an orchestration with pillar data:

```
client.api.cmd.route_cmd(
    cmd="runner",
    masters=["*"],
    fun="state.orch",
    arg={"kwarg": {"mods": "orch.my_orch", "pillar": {"key": "value"}}},
)
```

Example with an existing job_uuid

```
client.api.cmd.route_cmd(
    job_uuid="5c5cc410-4f9f-11e6-88bc-080027a7289c",
    tgt={"master3_master": {"tgt": "*", "tgt_type": "glob"}},
)
```

stop_job(...)

Returns <class 'str'>.

Pass a 'soft_kill' command through to the masters.

param job_uuid UUID of the job to attempt to kill

param jid Jid of the job to attempt to kill

return a Jid corresponding to the kill command

track_state(...)

Returns None.

Save tracking state for job

1.8.7 conf interface

The LoadedMod class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

delete_master_config(...)

Returns None.

Delete config for a master.

Table 29: delete_master_config() parameters

master_id		Identifier of master
-----------	--	----------------------

get_master_config(...)

Returns <class 'str'>.

Retrieve a master's configuration from the database

Table 30: get_master_config() parameters

master_id		Identifier of master
-----------	--	----------------------

replace_master_config(...)

Returns None.

Replace (wholesale) master config with the contents of `config_yaml`.

Table 31: replace_master_config() parameters

master_id		Identifier of master
config_yaml		A well-formed YAML string

save_master_config(...)

Returns None.

Add/update a master's configuration in the database

Table 32: save_master_config() parameters

master_id		Identifier of master
config_data		A dictionary of key and value pairs

1.8.8 fs interface

The `LoadedMod` class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

`delete_file(...)`

Returns None.

Delete a file from the database.

Table 33: `delete_file()` parameters

<code>file_uuid</code>		UUID of the file in question
<code>saltenv</code>		Salt environment in which to look for the file
<code>path</code>		Fully-qualified path of the file.

Note, use `file_uuid` or `saltenv` and `path`, but not both.

`file_exists(...)`

Returns <class 'bool'>.

Return True if file exists.

For more information on files and environments, see the Aria Automation Config documentation on VMware's Doc Center.

Table 34: `file_exists()` parameters

<code>file_uuid</code>		UUID of the file in question
<code>saltenv</code>		Salt environment in which to look for the file
<code>path</code>		Fully-qualified path of the file.

Note, use `file_uuid` or `saltenv` and `path`, but not both.

`get_env(...)`

Returns List[Dict].

List all files in the given Salt environment.

For more information on files and environments, see the Aria Automation Config documentation on VMware's Doc Center.

Table 35: `get_env()` parameters

<code>saltenv</code>		Salt environment in which to look for the files
<code>include_fs_metadata</code>		If True, include a dictionary key that has metadata associated with the file.

get_envs(...)

Returns List[str].

Get a list of all available Salt environments

For more information on files and environments, see the Aria Automation Config documentation on VMware's Doc Center.

get_file(...)

Returns Dict.

Get a file from the database. Returns a dictionary with the file's metadata and a data field with the actual file contents.

For more information on files and environments, see the Aria Automation Config documentation on VMware's Doc Center.

Table 36: get_file() parameters

file_uuid		UUID of the file in question
saltenv		Salt environment in which to look for the file
path		Fully-qualified path of the file.

Note, use file_uuid or saltenv and path, but not both.

get_file_access(...)

Returns Dict.

Return access metadata for this file.

Table 37: get_file_access() parameters

file_uuid		UUID referencing desired file.
-----------	--	--------------------------------

save_file(...)

Returns None.

Add or update a file to the database

For more information on files and environments, see the Aria Automation Config documentation on VMware's Doc Center.

Table 38: save_file() parameters

file_uuid		UUID of the file in question
saltenv		Salt environment in which to look for the file
path		Fully-qualified path of the file.
contents		The file's contents. If the file is plain text then contents is the literal content of the file. If it is a binary format then contents should be base64-encoded.
content_type		MIME content-type. Will be guessed if passed in as None.

Note, use `file_uuid` or `saltenv` and `path`, but not both.

`save_file_access(...)`

Returns None.

Save access metadata for this file.

Table 39: `save_file_access()` parameters

<code>file_uuid</code>		UUID referencing desired file.
<code>access_payload</code>		Dictionary containing role names as keys and a list of allowed access types as values

`update_file(...)`

Returns None.

Add or update a file to the database

For more information on files and environments, see the Aria Automation Config documentation on VMware's Doc Center.

Table 40: `update_file()` parameters

<code>file_uuid</code>		UUID of the file in question
<code>saltenv</code>		Salt environment in which to look for the file
<code>path</code>		Fully-qualified path of the file.
<code>contents</code>		The file's contents. If the file is plain text then <code>contents</code> is the literal content of the file. If it is a binary format then <code>contents</code> should be base64-encoded.
<code>content_type</code>		MIME content-type. Will be guessed if passed in as None.

Note, use `file_uuid` or `saltenv` and `path`, but not both.

1.8.9 job interface

The `LoadedMod` class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

`delete_job(...)`

Returns None.

Delete a job.

Table 41: `delete_job()` parameters

<code>job_uuid</code>		UUID referencing job to be deleted.
<code>force</code>		Force deletion of target group even with schedule jobs depending on it(which will also get deleted).

raises FailedResourceDependency The job has one or more schedules that depend on it and *force* is false. The message in the exception identifies the dependent schedules.

`get_job_access(...)`

Returns Dict.

Return access metadata for this job.

Table 42: `get_job_access()` parameters

<code>job_uuid</code>		UUID referencing desired job.
-----------------------	--	-------------------------------

`get_jobs(...)`

Returns Dict.

Get a job by uuid or search jobs by other query parameters.

Table 43: `get_jobs()` parameters

<code>param uuid job_uuid</code>		UUID referencing job. If left blank, all jobs are returned.
<code>param str name</code>		Text to search for in the job name
<code>param str desc</code>		Text to search for in the job description
<code>param str cmd</code>		Job command to match ('local', 'runner', 'wheel', or 'ssh')
<code>param UUID tgt_uuid</code>		Target group UUID to match
<code>param UUID role_uuid</code>		Role UUID to match (Admins only)
<code>param bool include_no_tgt_jobs</code>		Include jobs that have no <code>tgt_uuid</code> as well
<code>param str fun</code>		Salt function to match (e.g., 'test.ping')
<code>param list masters</code>		List of salt master ids to match
<code>param str sort_by</code>		Field to sort by ('name', 'desc', 'cmd', or 'fun')
<code>param bool reverse</code>		Whether to sort in descending order
<code>param int limit</code>		Maximum number of jobs to return, default 50
<code>param int page</code>		Page of jobs to return (offset = page * limit)

The flag `include_no_tgt_jobs` is intended to be used when a list of jobs that can be run at the time is desired. For example, if a user clicks on a target in the GUI and is presented with a list of jobs that go with that target, we may also want to show jobs that have NO target attached since those jobs may be run regardless of what target is selected.

This flag has no effect if `tgt_uuid` is None.

The return payload is a dict with the following elements:

```
{
  'count': 100,      # total job count
  'limit': 50,      # results count
  'results': [...]  # jobs
}
```

save_job(...)

Returns <class 'str'>.

Create or update a job

Parameters :name: Name to give to the job. :desc: Descriptive text for job. :cmd: One of *local* (for targeting minions), *ssh* (aka salt-ssh),

runner (master-level command), *wheel* (master wheel calls)

fun Dotted-notation function to run (e.g. *test.ping* or *network.ipaddrs*)

arg A dictionary containing the keys *arg* and/or *kwarg* defining job inputs that can be supplied at execution time (details below)

masters A list of master names that should receive this command.

job_uuid UUID referencing job. If left blank, one will be generated. If not blank, but no job with this UUID exists, a new job is created. If not blank and a UUID exists, that job is updated with the passed information.

tgt_uuid UUID for a target group.

This endpoint returns the UUID for the job.

Job Inputs

A job input definition describes an argument that can be supplied at job execution time. Inputs make a job more flexible than if salt function arguments are all hard-coded. Currently only keyword job inputs are supported. Positional arguments and pillar data are sent verbatim to salt at job execution time.

When creating or updating a job, specify positional, keyword, and pillar job inputs in the *arg* parameter with the following structure:

```
arg = {
  "arg": [
    'hello', # positional argument 0
    'world', # positional argument 1
    # ...
  ],
  "kwarg": {
    "name1": {...}, # keyword job input `name1`
    "name2": {...}, # keyword job input `name2`
    # ...
    "pillar": {
      "pillar1": {...}, # pillar argument `pillar1`
      "pillar2": {...}, # pillar argument `pillar2`
      # ...
    }
  }
}
```

Each job input definition consists of the following fields:

```
{
  "display_name": "Display name goes here",           # description
  ↪presented in the UI
  "input_type": "string" | "select" | "number" | "bool", # "select" ==
  ↪choose one of the options in "select"
  "select": ["Option 1", "Option 2", "Option 3"],      # if input_type
  ↪== "select"
```

(continues on next page)

(continued from previous page)

```

    "default": "Default value",                # should match❌
    ↪input_type                                # help presented❌
    "help": "Helpful text",
    ↪in the UI
    "required": True | False,                  # True == must❌
    ↪be specified at execution time
    "hidden": True | False,                    # True == not❌
    ↪presented in the UI
}

```

Required job inputs must be specified at job execution time, i.e., passed to `cmd.route_cmd()`. If a required input has a default value, it can be omitted from the `route_cmd()` payload. A hard-coded argument can be implemented by marking the input “required” and “hidden” and specifying the desired default value.

Here is an example invocation of `save_job()` and a call to `route_cmd()` to execute the job.

```

job_uuid = client.api.job.save_job(
    name="Deploy VM",
    desc=None,
    cmd="local", # salt
    fun="state.apply",
    arg={
        "arg": ["West", 35],
        "kwarg": {
            "mods": {
                "display_name": "State",
                "input_type": "string",
                "default": "deploy_vm.sls",
                "help": None,
                "required": True,
                "hidden": True
            },
            "saltenv": {
                "display_name": "Environment",
                "input_type": "string",
                "default": "SSE",
                "help": None,
                "required": True,
                "hidden": True
            },
            "reason": {
                "display_name": "Comment / Reason",
                "input_type": "string",
                "default": "VM Deployment",
                "help": None,
                "required": True,
                "hidden": True
            }
        },
    },
    tgt_uuid="7f93b928-388b-11e6-b133-346895ecb8f3" # all minions
)

jid = cmd.route_cmd(
    job_uuid=job_uuid,
    arg={
        "arg": [] # raas will copy positional args verbatim from the job
    }
)

```

(continues on next page)

(continued from previous page)

```
}    "kwarg": {} # raas will apply defaults for mods, saltenv, reason
```

)

save_job_access(...)

Returns <class 'uuid.UUID'>.

Save access metadata for this job.

Table 44: save_job_access() parameters

job_uuid		UUID referencing desired job.
access_payload		Dictionary containing role names as keys and a list of allowed access types as values

1.8.10 kv interface

The LoadedMod class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

delete_key(...)

Returns <class 'int'>.

Delete a key for the current user.

Table 45: delete_key() parameters

key		The key to delete.
-----	--	--------------------

Returns the number of keys deleted (0 or 1).

delete_system_key(...)

Returns <class 'int'>.

Delete a system key. System keys are used internally and cannot be retrieved directly.

Table 46: delete_system_key() parameters

key		The system key to delete.
-----	--	---------------------------

Returns the number of system keys deleted (0 or 1).

`get_key(...)`

Returns Union[NoneType, int, str, Dict, List].

Get the value of a key for the current user.

Table 47: `get_key()` parameters

key		The key to get.
-----	--	-----------------

Returns the value of the key, or None if the key was not found.

`list_keys(...)`

Returns List[str].

Get a list of key names for the current user.

Returns a (possibly empty) list of key names.

`set_key(...)`

Returns None.

Set the value of a key for the current user.

Table 48: `set_key()` parameters

key		The key to set.
value		The key value.

`set_system_key(...)`

Returns None.

Set the value of a system key. System keys are used internally and cannot be retrieved directly.

Table 49: `set_system_key()` parameters

key		The system key to set.
value		The value to set.

1.8.11 license interface

The LoadedMod class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

`apply_vra_license(...)`

Returns None.

Validate a vRA license key and apply it if it checks out.

Table 50: `apply_vra_license()` parameters

<code>vra_license_key</code>		vRA license key
------------------------------	--	-----------------

Example:

```
client.api.license.apply_vra_license('0J4C2-0FK4K-M8UF2-02824-14EH4')

RPCResponse(riq=3,
             ret=None,
             error=None,
             warnings=[])
```

`check_usage(...)`

Returns Dict.

Return a dictionary that contains information about the license status.

Table 51: `check_usage()` parameters

<code>force_check</code>		If False uses information cached over time since getting actual usage can be somewhat time-consuming for large environments. If True forces the usage calculation and then returns results.
--------------------------	--	---

Note: Make sure to track your license status and keep it active. If your license expires, the Aria Automation Config service will stop.

Example:

```
client.api.license.check_usage()

RPCResponse(riq=17,
             ret={u'timestamp': u'2017-11-30T15:58:24.475034',
                  u'messages': [{u'detail': u'license expires 2018-06-01',
                                u'level': u'info'},
                                {u'detail': u'4 masters in use is within hard limit of 10',
                                u'level': u'info'},
                                {u'detail': u'4243 minions in use is within hard limit of 10000
→ ',
                                u'level': u'info'}]}],
             error=None, warnings=[])
```

delete_license(...)

Returns Optional[str].

Delete the license matching this uuid or serial number. Deleting an active license with no alternative license in place can cause RaaS to stop working.

- UUID or serial number for the license to delete.

Example:

```
client.api.license.delete_license(license_uuid_or_serial='0J4C2-0FK4K-M8UF2-02824-
↪14EH4')

RPCResponse(riq=4,
             ret='0J4C2-0FK4K-M8UF2-02824-14EH4',
             error=None,
             warnings=[])

client.api.license.delete_license(license_uuid_or_serial='dd1b95fe-930b-193d-a7f8-
↪98172192d31a')

RPCResponse(riq=4,
             ret='dd1b95fe-930b-193d-a7f8-98172192d31a',
             error=None,
             warnings=[])
```

get_current_license(...)

Returns Dict.

Retrieve the current license for RaaS. The returned license structure reflects the minion counts and features of all current licenses.

Note: Make sure to track your license status and keep it active. If your license expires, the Aria Automation Config service will stop.

Example:

```
c.api.license.get_current_license()

RPCResponse(riq=16,
             ret={u'uuid': u'1dbb6f3c-c622-11e6-874c-002590058d18',
                  u'license': {u'masters':
                               {u'policy': u'fixed',
                                u'hard_limit': 10},
                               u'term': {u'duration': -1,
                                           u'policy': u'fixed'},
                               u'features': [u'*'],
                               u'minions': {u'policy': u'fixed',
                                              u'hard_limit': 10000},
                               u'metadata': {u'customer': u'Aria Config',
                                              u'uuid': u'1dbb6f3c-c622-11e6-874c-002590058d18',
                                              u'created': u'2016-12-19'}}},
```

(continues on next page)

(continued from previous page)

```
u'desc': [u'Customer:    Aria Config',
          u'License UUID: 1dbb6f3c-c622-11e6-874c-002590058d18',
          u'Created:     2016-12-19',
          u'Duration:    UNLIMITED',
          u'Masters:     up to 10',
          u'Minions:    up to 10000',
          u'Features:    *']],
error=None, warnings=[])
```

get_license(...)

Returns Dict.

Return license for the passed uuid. This routine is deprecated in favor of `get_licenses()`.

- UUID for the desired license.

get_license_features(...)

Returns Dict.

Get the state of license features

get_licenses(...)

Returns List.

Query current licenses.

- UUID for the desired license
- - license_type
 -
 - License type to match
- - features
 -
 - A list of feature strings to match
- - include_expired
 -
 - Pass True to include expired licenses in the results

get_usage_snapshots(...)

Returns List.

Return snapshots of license usage for a given date range.

Snapshots are normally taken every hour.

Table 52: get_usage_snapshots() parameters

date_start		Initial snapshot date.
date_end		Final snapshot date.

Example:

```
client.api.license.get_usage_snapshots('2017-11-01','2017-11-17')

RPCResponse(riq=25,
  ret=[{'license_uuid': u'l1dbb6f3c-c622-11e6-874c-002590058d18',
        'masters': 4,
        'minions': 4212,
        'datetime': u'2017-11-16T18:06:49.171497'},
        {'license_uuid': u'l1dbb6f3c-c622-11e6-874c-002590058d18',
        'masters': 4, 'minions': 4243,
        'datetime': u'2017-11-16T19:55:26.095901'},
        {'license_uuid': u'l1dbb6f3c-c622-11e6-874c-002590058d18',
        'masters': 4,
        'minions': 4222,
        'datetime': u'2017-11-16T20:55:26.092793'} .... ],
  error=None, warnings=[])
```

get_vra_license(...)

Returns List.

Get the current vRA license keys and their license editions.

Example:

```
client.api.license.get_vra_license()

RPCResponse(riq=4,
  ret=[{'serial': '0J4C2-0FK4K-M8UF2-02824-14EH4',
        'edition': 'vac.vrealizeFlex2019Enterprise.subscription.core'},
        {'serial': 'MH027-NCH92-M8HL3-0H1R4-349KH',
        'edition': 'vac.saltStackSecOps.serverVm'}],
  error=None,
  warnings=[])
```

`save_license(...)`

Returns `<class 'uuid.UUID'>`.

Validate a VMware license key and save the associated license to the database. A valid license takes effect immediately.

1.8.12 master interface

The `LoadedMod` class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

`authenticate_master(...)`

Returns Dict.

Authenticate a salt-master. If the master's key is not yet in the database, save it in pending state (a pending key remains in that state until it is accepted, rejected, or deleted, or until another new key is submitted for the same salt-master). If the key is accepted, return a new jwt.

Table 53: `authenticate_master()` parameters

<code>master_id</code>		Salt master ID
<code>pubkey</code>		Salt master public key in PEM format
<code>nonce</code>		Encrypted and signed payloads

The `nonce` dict contains two base64-encoded strings: `"enc"`, a random string which has been encrypted with the salt-master's private auth key, and `"sig"`, a signature on the `"enc"` value created with the raas public auth key. The encryption and the signature serve as a way for the salt-master and raas to authenticate themselves to each other, proving that they each have their private key. If either one doesn't check out, raas will report an error. If both look good, raas will re-encrypt the decrypted `"enc"` string with the salt-master's public key and add it to the return payload.

Return payload:

```
{ "key_state": "...", # accepted, pending, or rejected
  "enc": "...", # nonce, decrypted and re-encrypted
  "jwt": "...", # if the key is accepted
}
```

If the `"enc"` string in the return payload doesn't match what the salt-master sent, it should ignore the response.

`delete_master(...)`

Returns None.

Delete master and its grains from RaaS.

Table 54: `delete_master()` parameters

<code>master_id</code>		ID (not UUID) of the master to delete.
<code>master_uuid</code>		UUID of the master to delete.

Note: use `master_id` or `master_uuid`, but not both.

get_master_grains(...)

Returns Dict.

Get grains for a master.

Table 55: get_master_grains() parameters

master_id		ID (not UUID) of the master.
master_uuid		UUID for the master.
split_cluster		If True return the grains for the individual master. If False return the grains for the cluster in which the master resides.

Example:

```
client.api.master.get_master_grains('saltmaster1_master')

RPCResponse(rq=26, ret=
    {u'saltmaster1_master': {u'grains': {u'biosversion': u'4.2.amazon',
                                         u'kernel': u'Linux',
                                         u'domain': u'localdomain',
                                         u'uid': 0,
                                         u'zmqversion': u'4.1.4',
                                         u'kernelrelease': u'3.10.0-693.5.2.el7.x86_64
→ ',
                                         ..... }}
    error=None, warnings=[])
```

get_master_jwt(...)

Returns Dict.

Get a JWT authenticating using a master RSA key pair

Table 56: get_master_jwt() parameters

master_id		(Required) ID (not UUID) of the master.
encrypted_message		(Required) Encrypted JSON message with {created, master_id}.

get_master_keys(...)

Returns List.

Query salt-master authentication keys

Table 57: get_master_keys() parameters

state: Filter by key state		accepted, pending, or rejected
master_id		Filter by master id (substring match)
sort_by: Sort results		master_id, state, created (default master_id)
reverse		Reverse sort order

get_master_presence(...)

Returns Dict.

Retrieve presence status for all salt masters in the infrastructure.

Table 58: get_master_presence() parameters

master_id		(Optional) ID (not UUID) of the master.
cluster_id		(Optional) Cluster with which the master is associated with.
status		(Optional) Presence status of the master.
sort_by		(Optional) Sort by one of master_id, cluster_id, status or last_seen.
reverse		(Optional) Sort by in ascending order of the sort_by column when reverse is False, and vice-versa.
page		(Optional) Return data from this page number.
limit		(Optional) Set page size (maximum number of records to return).

Example:

```
client.api.master.get_master_presence()

RPCResponse(riq=6543, ret={
  'results': [
    {'uuid': 'aa3616d5-bf1e-40ad-a207-45c2a06bb47f', 'master_id': 'master1',
    ↪ 'cluster_id': 'salt',
      'last_seen': '2020-11-14T19:15:24.200667', 'status': 'lost'},
    {'uuid': 'd1a884c6-16bd-4095-b226-568d0e578a00', 'master_id': 'master2',
    ↪ 'cluster_id': 'salt',
      'last_seen': '2020-11-14T19:15:24.200667', 'status': 'present'},
  ],
  'count': 2
},
error=None, warnings=[])
```

get_plugin_status(...)

Returns Dict.

Retrieve the status of the plugin on all connected masters. Depends on get_master_presence.

Table 59: get_plugin_status() parameters

master_id		(Optional) ID (not UUID) of the master.
cluster_id		(Optional) Cluster with which the master is associated with.
plugin_status		(Optional) List of plugin_statuses to filter on
sort_by		(Optional) Sort by one of master_id, cluster_id, status or last_seen.
reverse		(Optional) Sort by in ascending order of the sort_by column when reverse is False, and vice-versa.
page		(Optional) Return data from this page number.
limit		(Optional) Set page size (maximum number of records to return).

get_plugin_versions(...)

Returns Dict.

Retrieve available master plugin versions from RaaS.

Example:

```
client.api.master.get_plugin_versions()

RPCResponse(
  riq=3,
  ret={
    "current": {
      "build_date": "2022-10-04T17:28:57",
      "git_describe": "v8.10.1.0",
      "name": "/app/raas/raas/static/master_plugins/SSEAPE-8.10.1.0-py3-
↪none-any.whl",
      "message": "success",
      "sha256":
↪"9b7279f1bc927e20c6965f194f93880c5b262777daddf1052cc1b08047bccee",
      "filename": "SSEAPE-8.10.1.0-py3-none-any.whl"
    },
    "rollback": {
      "name": "SSEAPE-8.10.0.0-py3-none-any.whl",
      "message": "Unpacking of whl /app/raas/raas/static/master_plugins/
↪SSEAPE-8.10.0.0-py3-none-any.whl failed with return code 1: Missing SSEAPE-
↪8.10.0.0.dist-info/RECORD file"
    }
  }
)
```

```
“
    “filename”: “SSEAPE-8.10.0.0-py3-none-any.whl”
  }, error=None, warnings=[]
)
```

request_master_key(...)

Returns Dict.

Request RSA Public key for master authentication use.

Table 60: request_master_key() parameters

master_id		(Required) ID (not UUID) of the master requesting key.
-----------	--	--

rotate_master_key(...)

Returns Dict.

Rotate RSA Public key for master authentication use.

Table 61: rotate_master_key() parameters

master_id		(Required) ID (not UUID) of the master requesting key.
-----------	--	--

save_master(...)

Returns <class 'str'>.

Update or add a salt-master to the database.

Returns the UUID of the master in the database.

Table 62: save_master() parameters

master_id		ID (not UUID) of new/updated master
cluster_id		ID of cluster to which this master belongs or None if no cluster is in place.
master_uuid		UUID of master. If blank and no master with master_id exists a new one will be assigned.
grains		Master-level grains for this new master.

set_master_key_state(...)

Returns List.

Set RSA key state of masters.

Table 63: set_master_key_state() parameters

masters		(Required) Master IDs
action		(Required) accept, reject, delete.

set_plugin_status(...)

Returns None.

Set status of the Aria Automation Config (AAC) plugin on a salt-master

Table 64: set_plugin_status() parameters

param master_id		ID (not UUID) of the salt-master to update.
param status		Plugin update status
param last_update_jid		(Optional) last known jid used to upgrade/update the plugin
param log_messages		log messages that the runner wants to report in the UI :return:

update_plugin(...)

Returns <class 'str'>.

Triggers update of the Aria Automation Config (AAC) plugin on a salt-master to the latest available version. This call routes a salt-run command to the master to do the upgrade and returns the jid associated with that command.

Table 65: update_plugin() parameters

master_id		ID (not UUID) of the salt-master to update.
-----------	--	---

Returns the jid of the command routed to the master to do the upgrade.

update_target_group_from_master(...)

Returns None.

Allows a master to tell RaaS that it has an updated list of minions for a given target group.

Table 66: update_target_group_from_master() parameters

master_id		ID (not UUID) of the master to which these minions belong
tgt_uuid		UUID of the target group to be changed
minions_added		List of minions ids added to the target group
minions_deleted		List of minions ids deleted from the target group
canonical_list		Request that RaaS remove all minions for this master and tg and replace them with the minions_added list.

1.8.13 masterfs interface

The LoadedMod class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

get_masterfs(...)

Returns Dict.

Get a list of master filesystem files

Table 67: get_masterfs() parameters

master_id		ID (not UUID) of master
-----------	--	-------------------------

1.8.14 minions interface

The **LoadedMod** class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

delete_minion_cache(...)

Returns None.

Flush the minion's cache on a master.

Table 68: delete_minion_cache() parameters

master_id		ID (not UUID) of master
minion_id		ID of minion for which the cache should be flushed.

delete_minion_key_state(...)

Returns None.

Remove all minion key state data from a particular master.

Table 69: delete_minion_key_state() parameters

master_id		ID (not UUID) of master.
-----------	--	--------------------------

deploy_minion(...)

Returns <class 'uuid.UUID'>.

Deploy a Salt minion on a target host

master_id Salt master id to which the Salt minion is to be connected.

host_name_ip Host name or IP address of the target system on which minion is to be deployed

os Operating System of the target host on which salt minion is to be deployed

minion_id Salt minion ID to be assigned to the deployed minion. Optional.

username User name to connect to the target system through SSH. Required during minion deployment. Optional during state file run.

password Password to connect to the target system through SSH. One of password / private_key / private_key_path is required during minion deployment. Optional during state file run.

private_key Private key to connect to the target system through SSH. One of password / private_key / private_key_path is required during minion deployment. Optional during state file run.

private_key_path Private key to connect to the target system through SSH. One of password / private_key / private_key_path is required during minion deployment. Optional during state file run.

saltenv Saltenv to use when running state files. Optional.

pillarenv Pillarenv to use when running state files. Optional.

installer_file_name Salt minion installer file name on the master. Optional.

state_files List of state files to run on the deployed minion. Optional.

variables Parameters required by the state file to run on the deployed minion. Optional.

additional_minion_params Additional configuration parameters for the salt minion, to be passed in as dictionary. Refer: <https://docs.saltproject.io/en/latest/ref/configuration/minion.html>

additional_auth_params Additional auth params that can be passed in for provisioning the salt minion. This API uses saltify driver of salt-cloud under the hood. Custom values for provider and profile can be passed under keys provider and profile respectively. These will be merged/override the default salt-cloud configuration for saltify driver. Example: {'provider': {'driver': 'digitalocean', 'personal_access_token': 'xxx', 'location': 'NY'},

'profile': {'image': 'fedora 17', 'size': '256 server'}}

Refer: <https://docs.saltproject.io/en/master/topics/cloud/profiles.html>

get_minion_cache(...)

Returns Dict.

Retrieve the minion cache for a particular master.

Table 70: get_minion_cache() parameters

master_id		ID for master.
minion_id		ID for minion.

Example:

```
client.api.minions.get_minion_cache(master_id='saltmaster1_master',
                                   minion_id='s01-m01')

RPCResponse(riq=28, ret=
  {u'saltmaster1_master':
    {u's01-m01':
      {u'grains':
        {u'biosversion': u'4.2.amazon',
          u'kernel': u'Linux',
          u'domain': u'us-west-1.compute.internal', ..... }}}},
  error=None, warnings=[])
```

get_minion_deployments(...)

Returns Dict.

Get minion deployments done using Aria Automation Config.

deploy_uuid Filter by deployment uuid. Optional.

master_id Filter by Salt master id. Optional.

minion_id Filter by Salt minion ID. Optional.

page Return data from this page number.

limit Set page size (maximum number of minion deployment records to return).

get_minion_details(...)

Returns Dict.

Get minion details.

Table 71: get_minion_details() parameters

master_id		Filter minions by master id.
minion_id		Filter minions by minion id.
tgt_uuid		Limit the query to minions matching this target group.
grains		Filter the minions by matching grains.
sort_by		Sort results by this minion grain.
reverse		Pass True to sort results in descending order.
page		Return data from this page number.
limit		Set page size (maximum number of minions to return).
response_file		Pass 'filename.csv' or .json to return a file.
exact_match: Match the exact master_id and minion_id. Default		False

get_minion_grain_keys(...)

Returns List[str].

Get a list of unique minion grain keys, optionally filtering by master id, minion id, or target group.

Table 72: get_minion_grain_keys() parameters

master_id		Limit the query to minion grains reported by this master
minion_id		Limit the query to grains from this minion
tgt_uuid		Limit the query to minions matching this target group

Note: when `enable_grains_indexing: False` this will always return an empty list.

get_minion_grain_values(...)

Returns List[str].

Get a list of unique minion grain values for a particular key, optionally filtering by master id, minion id, or target group.

Table 73: get_minion_grain_values() parameters

key		Get possible minion grain values for this minion grain key
master_id		Limit the query to minion grains reported by this master
minion_id		Limit the query to grains from this minion
tgt_uuid		Limit the query to minions matching this target group

Note: when `enable_grains_indexing: False` this will always return an empty list.

get_minion_key_state(...)

Returns Dict.

Get minion key states

Table 74: get_minion_key_state() parameters

master_id		Filter results by master id.
minion_id		Filter results by minion id.
key_state		Limit results to this minion key state (accepted, rejected, pending, or denied).
sort_by		Sort results by this field (master_id, minion_id, or key_state).
reverse		Pass True to sort results in descending order.
page		Return data from this page number.
limit		Set page size (maximum number of results to return).

get_minion_presence(...)

Returns Dict.

Retrieve presence status for all minions in the infrastructure.

Example:

```
client.api.minions.get_minion_presence()

RPCResponse(riq=6543, ret={
    'results': [
        {'minion': 's01-m01', 'master': 'saltmaster1_master', 'status': 'present',
        ↪ 'timestamp': u'2017-11-27T23:06:12.674176'},
        {'minion': 's01-m02', 'master': 'saltmaster1_master', 'status': 'lost',
        ↪ 'timestamp': u'2017-11-27T23:06:12.674176'}
    ],
    'count': 2
},
error=None, warnings=[])
```

save_minion_cache(...)

Returns None.

Update the minion data cache for a given master. Note that data about minions with unaccepted keys will be ignored. This function should be called only by a salt-master.

Table 75: save_minion_cache() parameters

master_id		ID (not UUID) of master
minions_delta		A dictionary specifying changes to minion grains data to be made in RaaS. The top-level key <code>update</code> contains a dictionary keyed by minion id where each value contains the grains to save for that minion. Any existing grains data for a particular minion will be discarded before the new grains are saved. The top-level key <code>delete</code> contains a list of minion ids identifying grains data to remove from RaaS.

save_minion_key_state(...)

Returns None.

Save minion key state for a given master.

This function should only be called by a salt-master.

Table 76: save_minion_key_state() parameters

master_id		ID (not UUID) for the master to update.
keys		Dictionary containing key state for minions. This dictionary needs to be in this format:

set_minion_key_state(...)

Returns Dict.

Set the minion key state on the provided master id/cluster id.

Table 77: set_minion_key_state() parameters

state		accept, delete, or reject
minions		list of [master_id, minion_id] pairs to set
include_accepted		include accepted keys along with pending keys, works with reject state only
include_rejected		include rejected keys along with pending keys, works with accept state only
include_denied		include denied keys along with pending keys, works with accept and reject state only

1.8.15 pillar interface

The LoadedMod class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

delete_pillar(...)

Returns None.

Delete pillar for the given UUID.

Table 78: delete_pillar() parameters

pillar_uuid		UUID of pillar to delete.
-------------	--	---------------------------

get_pillar_access(...)

Returns Dict.

Get the access metadata for a pillar entry.

Table 79: get_pillar_access() parameters

pillar_uuid		Retrieve metadata for pillar matching this UUID.
-------------	--	--

get_pillars(...)

Returns Dict.

Get the data for the given pillar. For more information on pillar data, see the Aria Automation Config documentation on VMware's Doc Center.

Table 80: get_pillars() parameters

param pillar_uuid		UUID of the pillar structure desired.
param name		name for this particular pillar entry

The return payload is a dict with the following elements:

```
{
  'count': 100,      # total job count
  'limit': 50,      # results count
  'results': [...]  # jobs
}
```

save_pillar(...)

Returns <class 'str'>.

Set data in the requested pillar. For more information on pillar data, see the Aria Automation Config documentation on VMware's Doc Center.

Table 81: save_pillar() parameters

pillar		Data structure to save for this pillar.
pillar_type		Type of pillar
pillar_uuid		UUID for pillar—if blank a new one will be generated
name		Name for this particular pillar entry
desc		Text describing this pillar entry.

`save_pillar_access(...)`

Returns None.

Save the access metadata for a target group.

Table 82: `save_pillar_access()` parameters

<code>pillar_uuid</code>		Save metadata for pillar matching this UUID.
<code>access_payload</code>		Dictionary containing role names as keys and a list of allowed access types as values

1.8.16 ret interface

The LoadedMod class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

`get_fun(...)`

Returns Dict.

Return a dictionary of the last function called for all minions. Including `fun` will restrict the result set to minions for which `fun` was their last function called.

Table 83: `get_fun()` parameters

<code>fun</code>		Limit result set to minions that called this function last.
------------------	--	---

`get_jid(...)`

Returns Dict.

Get the results for a specific job by its job id (`jid`).

More information on jobs can be found in the Aria Automation Config documentation on VMware's Doc Center.

Table 84: `get_jid()` parameters

<code>jid</code>		Job ID.
------------------	--	---------

`get_jids(...)`

Returns `<class 'raas.utils.serialize.JSONString'>`.

Get all Job IDs for a given range. Warning, with no boundaries this call can return a huge amount of data.

More information on jobs can be found in the Aria Automation Config documentation on VMware's Doc Center.

Table 85: `get_jids()` parameters

<code>start</code>		Timestamp from which to start.
<code>end</code>		Timestamp at which to end.
<code>limit</code>		Maximum number of job entries to return.

get_load(...)

Returns Dict.

Return the load for the given JID from the given master.

Table 86: get_load() parameters

master_id		ID for the master (not UUID)
jid		Job ID.

get_minions(...)

Returns Dict[str, List[str]].

Returns a dictionary of the current list of known minion IDs keyed by master ID

get_returns() by JID example:

```
client.api.ret.get_returns(jid='20171205214029544461')
```

get_returns() by function example:

```
client.api.ret.get_returns(minion_id='minion1')
```

get_returns () by minion ID example:

```
client.api.ret.get_returns(minion_id='minion2')
```

Example:

```
client.api.ret.get_minions()

RPCResponse(riq=28, ret=
  {u'saltmaster1_master':
    [u's01-m01', u's01-m02', u's01-m03', ...],
    u'saltmaster2_master':
    [u's02-m01', u's02-m02', u's02-m03', ...],
    u'saltmaster3_master':
    [u's03-m01', u's03-m02', u's03-m03', ...]},
  error=None, warnings=[])
```

get_returns(...)

Returns Dict.

Return job results from the job cache based on the provided arguments.

Table 87: get_returns() parameters

jid		Job ID.
master_id		Return results only for jobs run on this master.
minion_id		Return results only for jobs executed on this minion.
fun		Return results for this called function.
user		Restrict results to jobs executed by this user.
job_name		Filter by job_name
job_uuid		Return results for this job's UUID (not Job ID (jid))
tgt_name		Filter by tgt_name
tgt_uuid		Return job results executed via this target group.
match_tgt_uuid		Return job results for jobs matching this target group.
args		filter by an argument string
highstate		Return only results that were generated via highstate.
start		Return results starting from this timestamp.
end		Stop returning results at this timestamp.
limit		Limit results to this number of jobs.
has_errors		Return results for those jobs with errors.
sort_by		Return results sorted by minion_id, function, has_errors, or jid.
reverse		Return results sorted in the reverse order.
page		Return job results from this page. Page starts from 0.

get_returns() example:

```
client.api.ret.get_returns()
```

The return payload is a dictionary with the following elements:

```
{
  'count': 100,      # total job returns count
  'limit': 50,      # results count
  'results': [...]  # job returns
}
```

Example:

```
client.api.ret.get_returns(jid='20150430164924016227')

RPCResponse(
  { 'count': 1,
    'limit': 1,
    'results': [ { 'alter_time': '2018-06-29T21:29:37.720943',
                  'full_ret': { '_stamp': '2015-04-30T16:49:24.379894',
                                'cmd': '_return',
                                'fun': 'pillar.items',
                                'fun_args': [],
                                'id': 'raas_test_ret_returns_minion',
                                'jid': '20150430164924016227',
                                'master_id': 'raas_test_ret_returns_master',
                                'retcode': 0,
                                'return': { 'baz': 'qux',
                                             'xc-cluster': True,
                                             'qux': 'baz' },
                                'success': True },
                  ]
  }
```

(continues on next page)

(continued from previous page)

```
'fun': 'pillar.items',
'fun_args': [],
'has_errors': False,
'jid': '20150430164924016227',
'master': 'raas_test_ret_returns_master',
'master_uuid': '5c56659b-0619-45bc-8ad7-e805c28d85ce',
'minion_id': 'raas_test_ret_returns_minion',
'return': {'baz': 'qux', 'lxc-cluster': True, 'qux': 'baz'}}]})
```

1.8.17 schedule interface

The `LoadedMod` class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

futures(...)

Returns Dict.

Get a list of future times when the scheduled jobs will execute. If `jobs` is passed in then only the futures for those jobs will be calculated. By default the job times will be queried out for 8 weeks

Table 88: futures() parameters

list sched_names		Matching names of the schedule entries to return
list job_names		A list of job names to query
list tgt_names		Filter by target names in the schedule.
UUID uuid		Filter by schedule uuid.
list state		A list of states to match
list state_invert		A list of states to exclude
str fun		The name of the function that was run
str sort_by		Field to sort by. One of 'fun', 'tgt_name', 'job_name', 'sched_name', 'start_time', 'state'.
list daterange		A list of two date strings in ISO 8601 format in UTC. If the provided dates are in a different offset, they will be converted to UTC prior to the date range constraining.
int page		Which page of the records to return (offset = page * limit)
int limit		The number of job instances to return, defaults to 100
bool reverse		Sort ascending (False) or descending (True).

The parameter `daterange`, which is a date string in ISO8601 format has a sibling parameter, `daterange_fmt`, which is a string that can represent an alternate date and time format used to parse the `daterange` values.

`get(...)`

Returns Dict.

Return the full schedule json structure.

For more information on schedules, see the Aria Automation Config documentation on VMware's Doc Center.

param list names Matching names of the schedule entries to return

param bool show_past Include entries that were scheduled to run in the past.

param str query The text to search for in args or kwargs

param str cmd One of local, runner, ssh, wheel

param str fun The name of the function that was run

param list masters A list of masters that the job was assigned to

param str tgt A target string to search for

param bool arg Whether to search inside `*args`

param bool kwarg Whether to search inside `**kwargs`

param list state A list of states to match

param bool enabled is the scheduled item enabled to run or not

param UUID uuid Schedule UUID to match

param list job_names A list of job names to match

param list tgt_names A list of target group names to match

param str sort_by Field to sort by

param bool reverse Pass True to sort in descending order

param int limit How many records to return at a time

param int page Which page of the records to return (offset = page * limit)

NOTE: passing `['*']` as the value of `masters` will match the literal value `*` in the list of masters. `get()` is setup this way so it will be possible to look up schedules that are targeted at all masters. If `*` was treated as the wildcard, it would always match any job targeted

at any master

`get_inflight(...)`

Returns Dict.

Get a list of jobs that are in progress.

Table 89: get_inflight() parameters

cmd		A command name to match against.
filter_find_job		Exclude commands referring to saltutil.find_job.
fun		Command (function) name that was called for this job.
include_adhoc		When this is true, you can search jobs that are running (in-flight), but were not scheduled via the scheduler API (ad-hoc).
job_names		A list of job names to match against.
page		This specifies which page, after the first, of results to return.
limit		Limit search to this number of jobs.
reverse		Sort ascending (False) or descending (True).
sched_names		A list of schedule names to match against.
sort_by		Sort by the specified field, such as 'start_time'.
tgt_names		A list of target names to match against.
paused		Filter on only paused jobs.

remove(...)

Returns <class 'bool'>.

Remove a schedule by uuid.

param uuid The uuid of the schedule to be removed.

save(...)

Returns <class 'uuid.UUID'>.

Add/update a component in the schedule.

For more information on schedules, see the Aria Automation Config documentation on VMware's Doc Center.

param str name The name of the schedule item

param dict schedule A dictionary with schedule details.

param list masters List of masters on which to execute runner and wheel commands.

param str cmd One of local, runner, ssh, wheel

param dict arg Argument dictionary containing kwargs and args for fun below

param dict tgt The minion target dictionary (use this or tgt_uuid, not both)

param str tgt_uuid a UUID for an already saved target (use this or tgt, not both)

param str job_uuid a UUID for an already saved job (use this or fun+arg, not both)

param str fun The function to run (use this or job_uuid, not both)

param bool enabled Set the schedule job enabled or disabled

arg should have this format:

```
arg: { 'arg': [<arg1>, <arg2>, ..., <argn>],
      'kwarg': { '<key1>': <val1>,
                  '<key2>': <val2>,
                  ...
                  '<keyn>': <valn> }
    }
```

schedule should have this format:

```
schedule: { '<descriptor1>': <value1>,
            '<descriptor2>': <value1>,
            ...
            '<descriptorn>': <valuen> }
```

Descriptor above can be a number of things like *seconds*, *once*, *when*, *cron*, etc. See the Aria Automation Config documentation for a complete list.

save_skip(...)

Returns Dict.

Update a list of futures (by UUID) skip flag

Table 90: save_skip() parameters

list uuids		A list of future uuids to set skip flag for.
boolean skip		True or False to skip future

search_history(...)

Returns Dict.

Query the logs of past scheduler runs.

Table 91: search_history() parameters

str query		The text to search for
bool name		Whether to search schedule names
str cmd		One of local, runner, ssh, wheel
str fun		The name of the function that was run
list masters		The master(s) that the job was assigned to
str tgt		A target string to search for
bool arg		Whether to search inside <i>*args</i>
bool kwarg		Whether to search inside <i>**kwargs</i>
list daterange		A list of two date strings in ISO 8601 format
int limit		How many records to return at a time
int page		Which page of the records to return (offset = page * limit)

The parameter *daterange*, which is a date string in ISO8601 format has a sibling parameter, *daterange_fmt*, which is a string that can represent an alternate date and time format used to parse the *daterange* values.

update(...)

Returns <class 'uuid.UUID'>.

Update a schedule.

For more information on schedules, see the Aria Automation Config documentation on VMware's Doc Center.

- param str name** The name of the schedule item
- param dict schedule** A dictionary with schedule details.
- param list masters** List of masters on which to execute runner and wheel commands.
- param str cmd** One of local, runner, ssh, wheel
- param dict arg** Argument dictionary containing kwargs and args for fun below
- param dict tgt** The minion target dictionary (use this or tgt_uuid, not both)
- param str tgt_uuid** a UUID for an already saved target (use this or tgt, not both)
- param str job_uuid** a UUID for an already saved job (use this or fun+arg, not both)
- param str fun** The function to run (use this or job_uuid, not both)
- param bool enabled** Set the schedule job enabled or disabled

arg should have this format:

```
arg: { 'arg': [<arg1>, <arg2>, ..., <argn>],
      'kwarg': { '<key1>': <val1>,
                  '<key2>': <val2>,
                  ...
                  '<keyn>': <valn> }
    }
```

schedule should have this format:

```
schedule: { '<descriptor1>': <value1>,
            '<descriptor2>': <value1>,
            ...
            '<descriptorn>': <valuen> }
```

Descriptor above can be a number of things like *seconds*, *once*, *when*, *cron*, etc. See the Aria Automation Config documentation for a complete list.

1.8.18 sec interface

The **LoadedMod** class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

assess_policy(...)

Returns Dict.

Run assessment of checks in a policy

Table 92: assess_policy() parameters

policy_uuid		Policy UUID to assess.
-------------	--	------------------------

Response

Job ID for assessment run for specified policy.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.assess_policy(
    policy_uuid="1f90f261-1668-486e-a1d3-4dd68ef3020c"
)
```

Response

```
RPCResponse(
    riq=4,
    ret={
        "success": true,
        "errors": [],
        "jid": "20190125001746553804"
    },
    error=None,
    warnings=[])
```

delete_custom_benchmarks(...)

Returns None.

Delete custom benchmarks

Table 93: delete_custom_benchmarks() parameters

benchmark_uuids		List of custom benchmark UUIDs.
-----------------	--	---------------------------------

Returns

Returns nothing if the benchmark was successfully deleted

Examples

Request

```

from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.delete_custom_benchmarks(
    benchmark_uuids=["e912435e-2c38-4cfa-aa1e-985c83aa8a22"]
)

```

Response

```

RPCResponse(
    riq=4,
    ret={},
    error=None,
    warnings=[])

```

delete_custom_checks(...)

Returns None.

Delete custom checks

Table 94: delete_custom_checks() parameters

check_uuids		List of custom check UUIDs.
-------------	--	-----------------------------

Returns

Returns nothing if the check was successfully deleted

Examples

Request

```

from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.delete_custom_checks(
    check_uuids=["e912435e-2c38-4cfa-aa1e-985c83aa8a22"]
)

```

Response

```
RPCResponse(  
    riq=4,  
    ret={},  
    error=None,  
    warnings=[])
```

delete_exemption(...)

Returns Dict.

Delete an exemption

Table 95: delete_exemption() parameters

policy_uuid		UUID of the policy.
exemption_uuid		UUID of the exemption.

Returns

Examples

Request

```
from sseapiclient import APIClient  
client = APIClient('http://localhost', 'root', 'salt')  
client.api.sec.delete_exemption(  
    policy_uuid="e912435e-2c38-4cfa-aa1e-985c83aa8a22",  
    exemption_uuid="3a776df3-9c4d-4d54-8e1f-dc0c189f0f68"  
)
```

Response

```
RPCResponse(  
    riq=4,  
    ret={"uuid": "3a776df3-9c4d-4d54-8e1f-dc0c189f0f68"},  
    error=None,  
    warnings=[])
```

delete_policy(...)

Returns <class 'bool'>.

Delete a existing security policy.

Table 96: delete_policy() parameters

policy_uuid		UUID of the policy to delete.
-------------	--	-------------------------------

Returns

Response code from deleting the policy.

Examples**Request**

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.delete_policy(
    policy_uuid="db1ae70f-768e-4486-adae-b83546a09dea"
)
```

Response

```
RPCResponse(
    riq=4,
    ret=True,
    error=None,
    warnings=[])
```

download_content(...)

Returns None.

Download Locke tarball and expand the files into hub.opts['cachedir'] + /locke Once the download and expansion is complete, call ingest to put them in the raas filesystem

download_policy_report(...)

Returns <class 'str'>.

Download report for a policy assessment

Table 97: download_policy_report() parameters

policy_uuid		UUID of the policy.
jid		Job ID of the report to download. This parameter is optional.

Response

JSON formatted report file.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.download_policy_report(
    policy_uuid="3c625016-9dbf-44d8-b97c-17f4361b78a0"
)
```

Response

JSON formatted report file.

get_benchmark_dependencies(...)

Returns Dict.

Get the dependencies for a benchmark i.e. which policies and checks reference it

Table 98: get_benchmark_dependencies() parameters

benchmark		UUID of the benchmark.
-----------	--	------------------------

Returns

Returns the list of policies and checks that reference this benchmark

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.get_benchmark_dependencies(
    benchmark_uuid="e912435e-2c38-4cfa-aa1e-985c83aa8a22"
)
```

Response

```
RPCResponse(
  riq=4,
  ret={
    "policies": ["d3c68d55-04d7-4451-8f1e-f2b3e6d9ff3f", "b6da0147-2bbd-4497-b11b-
↪b9f5548e7297"],
    "checks": [{"uuid": "563577e3-e84f-4204-a89b-8456df96c34f", "other_benchmark_
↪references": True},
               {"uuid": "475a6070-56d5-49a0-840b-000efe8780e0", "other_benchmark_
↪references": False}],
  },
  error=None,
  warnings=[])
```

get_benchmarks(...)

Returns Dict.

Get information about benchmarks

Get information about one or more benchmarks.

Table 99: get_benchmarks() parameters

benchmark_uuid		UUID of the benchmark.
policy_uuid		List of benchmarks enabled within a given policy.
names		List of benchmarks matching names.
os		List of benchmarks applicable to specified OS.
authority_names		List of benchmarks matching authority names such as CIS, DISA STIGs.
type_name		List of benchmarks matching type such as Unix, Windows.
limit		Maximum number of policies to return.
sort_by		Sort results by supplied field.
page		Return results specified by page number.

Returns

List of benchmarks matching specified criteria.

Examples

Request 1

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.get_benchmarks()
```

Response 1

```
RPCResponse(  
  riq=4,  
  ret={  
    "count": 7,  
    "results": [  
      {  
        "uuid": "6b8ab80b-87a4-4f0c-8b6a-8bacf7c5853a",  
        "name": "CIS_CentOS_Linux_7_Benchmark_v2.2.0-1",  
        "display_name": "CIS CentOS Linux 7 Benchmark v2.2.0",  
        "version": "1",  
        "description": "Benchmark Description goes here",  
        "authority": "CIS",  
        "os": "None",  
        "type": "cis",  
        "dep_date": null,  
        "last_update": "2019-01-24T19:43:40.310324",  
        "user_flag": "None",  
        "policies": [  
          "b0714471-64cb-4446-a302-1e4ab6ec9cde"  
        ]  
      }  
    ]  
  },  
  error=None,  
  warnings=[])
```

Request 2

```
from sseapiclient import APIClient  
client = APIClient('http://localhost', 'root', 'salt')  
client.api.sec.get_benchmarks(  
    benchmark_uuid="6b8ab80b-87a4-4f0c-8b6a-8bacf7c5853a",  
    policy_uuid="b0714471-64cb-4446-a302-1e4ab6ec9cde"  
)
```

Response 2

```
RPCResponse(  
  riq=4,  
  ret={  
    "count": 1,  
    "results": [  
      {  
        "uuid": "6b8ab80b-87a4-4f0c-8b6a-8bacf7c5853a",  
        "name": "CIS_CentOS_Linux_7_Benchmark_v2.2.0-1",  
        "display_name": "CIS CentOS Linux 7 Benchmark v2.2.0",  
        "version": "1",  
        "description": "Benchmark Description goes here",  
        "authority": "CIS",  
        "os": "None",  
        "type": "cis",
```

(continues on next page)

(continued from previous page)

```

        "dep_date": null,
        "last_update": "2019-01-24T19:43:40.310324",
        "user_flag": "None",
        "policies": [
            "b0714471-64cb-4446-a302-1e4ab6ec9cde"
        ]
    }
]
},
error=None,
warnings=[])

```

get_check_dependencies(...)

Returns Dict.

Get the dependencies for a check i.e. which policies and benchmarks reference it

Table 100: get_check_dependencies() parameters

check_uuid		UUID of the custom check.
include_in_response		Select the blocks of results that should appear in the response. By default all results will appear in response. Following list are acceptable results, policies, benchmarks, exemptions.

Returns

Returns the list of policies and benchmarks that reference this check

Examples

Request

```

from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.get_check_dependencies(
    check_uuid="e912435e-2c38-4cfa-aa1e-985c83aa8a22"
)

```

Response

```
RPCResponse(  
    riq=4,  
    ret={  
        "policies": ["d3c68d55-04d7-4451-8f1e-f2b3e6d9ff3f", "b6da0147-2bbd-4497-b11b-  
↪b9f5548e7297"],  
        "benchmarks": ["563577e3-e84f-4204-a89b-8456df96c34f", "475a6070-56d5-49a0-  
↪840b-000efe8780e0"],  
        "exemptions": [{  
            'uuid': UUID('76523fe6-4feb-4ca3-af98-75a1a58aff0a'),  
            'policy_uuid': UUID('0ac8ba09-399d-4748-880c-9a9088ec5551'),  
            'master_id': None,  
            'minion_id': None,  
            'reason': 'why not'}]  
    },  
    error=None,  
    warnings=[])
```

get_check_details(...)

Returns Dict.

Get detailed information about a check

Table 101: get_check_details() parameters

check_uuid		UUID of the check.
------------	--	--------------------

Response

Details for check matching check UUID.

Examples

Request

```
from sseapiclient import APIClient  
client = APIClient('http://localhost', 'root', 'salt')  
client.api.sec.get_check_details(  
    check_uuid="0c3df74e-d0ca-4c6e-9d2f-39035ff98f7a"  
)
```

Response

```

RPCResponse(
  riq=4,
  ret={
    "uuid": "0c3df74e-d0ca-4c6e-9d2f-39035ff98f7a",
    "name": "locke.system.file.at_cron_authorized_users",
    "version": "1",
    "display_name": "Ensure at/cron is restricted to authorized users",
    "state_fs_uuid": "e78a934f-0355-4fc7-9d4c-3ccb1a5fcb79",
    "mini_meta_fs_uuid": "16666977-7d8b-44bd-9cd3-909546673098",
    .
    .
    .
  },
  error=None,
  warnings=[])

```

get_check_minions(...)

Returns Dict.

Get minion-related information about checks

Table 102: get_check_minions() parameters

policy_uuid		Policy UUID
check_uuid		List information about a specific check UUID included in a policy.
check_names		List of checks with matching check names.
minion_ids		List of minions matching minion-ids.
exempt		Limit results to minions that are exempt (True) or non exempt (False).
jid		Filter results by assessment job id.
state		Filter results by compliance state of the check or minion.
action		Filter results by compliance action, assess or remediate.
include_in_response		Select the blocks of results that should appear in the response. By default all results will appear in response. Following list are acceptable results, check_summary, minion_summary, check_stats, minion_stats, check_report, check_exempt, over-all_stats.
grains		Grains to select, filter and sort by.
comment		Filter results by assessment or remediation comment.
changes		Filter results by assessment or remediation changes.
sort_by		Sort by compliance states Compliant, Not Compliant, Not Applicable, Error, and Unknown.
reverse		Set to True to reverse sort order.
page		Return results specified by page number.
limit		Maximum number of results to return.

Response

Returns results matching criteria sent in the request.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.get_check_minions(
    policy_uuid="3c625016-9dbf-44d8-b97c-17f4361b78a0"
)
```

Response

```
RPCResponse(
  riq=4,
  ret={
    "count": 3,
    "results": [
      {
        "check_name": "locke.system.service.sshd_maxauthtries",
        "check_display_name": "Ensure SSH MaxAuthTries is set to 4 or less",
        "policy_uuid": "3c625016-9dbf-44d8-b97c-17f4361b78a0",
        "policy_name": "SecOps",
        "check_uuid": "80492e63-af0b-41c8-9a5b-b202be658561",
        "check_version": "1",
        .
        .
        .
      }
    ]
  },
  error=None,
  warnings=[])
```

get_check_variables(...)

Returns Dict.

Get variables for one or more checks

Table 103: get_check_variables() parameters

policy_uuid		UUID of the policy.
benchmark_uuids		List of benchmark UUIDs.
check_uuids		List of check UUIDs.
exclude_check_uuids		List of check UUIDs to exclude.
sort_by		Sort results by supplied field.
reverse		Set to True to reverse sort order.
page		Return results specified by page number.
limit		Maximum number of results to return.

Returns

Returns information about variables for one or more check UUIDs.

Examples

Request

```

from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.get_check_variables(
    check_uuids=["2e0e216d-1f8e-4e47-9bd5-e0a33ca7665c"]
)

```

Response

```

RPCResponse(
  riq=4,
  ret={
    "count": 1,
    "results": [{
      "check_uuid": "2e0e216d-1f8e-4e47-9bd5-e0a33ca7665c",
      "check_name": "locke.system.service.sshd_maxauthtries",
      "variables": [{
        "name": "_locke.system.service.sshd_maxauthtries.SSHD_CONFIG_
↪MAXAUTHTRIES",
        "uuid": "a0e2a5fc-4a1f-49e8-a27f-55e983491f99",
        "scope": "local",
        "check_uuid": "2e0e216d-1f8e-4e47-9bd5-e0a33ca7665c",
        "description": "Maximum number of authentication attempts permitted↪
↪per connection",
        "default_value": "4"
      }]
    }]
  },
  error=None,
  warnings=[])

```

get_checks(...)

Returns Dict.

Get information about checks

List information about checks

Table 104: get_checks() parameters

policy_uuid		Filter by this policy_uuid.
benchmark_uuids		List of checks matching one or more benchmark UUIDs.
user_flag		Filter checks by user_flag 'C' - Custom 'S' - Aria Config None - All
names		List of checks matching names.
reverse		Set to True to reverse sort order.
sort_by		Sort results by supplied field.
page		Return results specified by page number.
limit		Maximum number of checks to return.

Returns

List of checks matching selected criteria.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.get_checks(
    benchmark_uuids=["6b8ab80b-87a4-4f0c-8b6a-8bacf7c5853a"],
    policy_uuid = "fc760dae-b3a4-45e6-a910-5f7f76b08b59"
)
```

Response

```
RPCResponse(
  riq=4,
  ret={
    "count": 224,
    "results": [
      {
        "uuid": "0c3df74e-d0ca-4c6e-9d2f-39035ff98f7a",
        "name": "locke.system.file.at_cron_authorized_users",
        "version": "1",
        "display_name": "Ensure at/cron is restricted to authorized users",
        "state_fs_uuid": "e78a934f-0355-4fc7-9d4c-3ccb1a5fcb79",
        "mini_meta_fs_uuid": "16666977-7d8b-44bd-9cd3-909546673098",
        .
```

(continues on next page)

(continued from previous page)

```

        .
        .
    }
    ]
},
error=None,
warnings=[])

```

get_content_stats(...)

Returns <class 'dict'>.

Get the stats for the latest ingested content :param hub: :param custom: True|False :return: dict = {

results: None|dict = { uuid, ingest_date, benchmarks, checks
 } errors: List of error messages

get_policies(...)

Returns Dict.

Get summary information on known security policies.

Get information about configured policies.

Table 105: get_policies() parameters

policy_uuids		List of security policy UUIDs. This parameter is optional when retrieving a list of policies.
names		List of security policy names. This parameter is optional when retrieving a list of policies.
tgt_uuids		List of security policy target group UUIDs. This parameter accepts one or more tgt_uuids.
include_stats		Set to True to include stats in response. Set to False by default.
sort_by		Sort results by supplied field.
reverse		Set to True to reverse sort order.
page		Return results specified by page number.
limit		Maximum number of policies to return.

Returns

List of policies matching the request criteria.

Examples

Request 1

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.get_policies(
    policy_uuids=[
        "f768e864-b5c2-494f-a116-8c335edcb7bd",
        "292372eb-ce12-4c6f-bbfc-bf916ba02649"
    ],
    include_stats=True
)
```

Response 1

```
RPCResponse(
    req=4,
    ret={
        "count": 1,
        "results": [
            {
                "uuid": "51f06de4-60ae-4cf3-83c1-65635941b4ba",
                "name": "Oracle",
                "tgt_uuid": "2b6bbf3d-b728-44de-8c4a-9e019fc50175",
                "tgt_name": "OracleLinux",
                "schedule_uuid": null,
                "last_update": "2018-12-13T05:21:33.170597",
                "last_assess_jid": "20181213052141677720",
                "last_assess_timestamp": "2018-12-13T05:21:41.637187",
                "last_remed_jid": "20181213052240095855",
                "last_remed_timestamp": "2018-12-13T05:22:40.050577",
                "schedule": null,
                "variables": [
                    {
                        "check_uuid": null,
                        "var_name": null,
                        "var_value": null
                    }
                ],
                "stats": {
                    "compliant": 2,
                    "noncompliant": 18
                }
            }
        ],
        "check_count": 20,
        "minion_count": 1
    },
    error=None,
    warnings=[])
```


Request 2

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.get_policies()
```

Response 2

```
RPCResponse(
  riq=4,
  ret={
    "results": [
      {
        "last_remed_jid": null,
        "name": "Security Policy",
        "schedule_uuid": null,
        "uuid": "fae4aa90-8e0c-4fda-b52b-b53ee0593f3e",
        "last_assess_timestamp": null,
        "tgt_uuid": "7f93b928-388b-11e6-b133-346895ecb8f3",
        "stats": {
          "unknown": 88
        },
        "schedule": null,
        "variables": [
          {
            "var_name": null,
            "var_value": null,
            "check_uuid": null
          }
        ],
        "last_remed_timestamp": null,
        "last_assess_jid": null,
        "tgt_name": "All Minions",
        "last_update": "2019-01-13T19:55:54.476425"
      }
    ],
    "check_count": 22,
    "count": 1,
    "minion_count": 0
  },
  error=None,
  warnings=[])
```

get_policy_run_history(...)

Returns Dict.

Get assessment and remediation runs against the policy

Table 106: get_policy_run_history() parameters

policy_uuid		UUID of the policy.
jid		Job ID
fun		Job function name.
state		Filter by one or more statuses of the assessment or remediation runs.
user		Name of the user who ran the policy.
expected		Number of expected minions to return a result.
returned		Number of minions which returned a result.
sort_by		Sort results by supplied field.
reverse		Set to True to reverse sort order.
page		Return results specified by page number.
limit		Maximum number of results to return.

Returns

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.get_policy_run_history(
    policy_uuid="e912435e-2c38-4cfa-aa1e-985c83aa8a22"
)
```

Response

```
RPCResponse(
  riq=4,
  ret={
    "count": 2,
    "results": [{
      "jid": "20190127194046350020",
      "state": "completed_failures",
      "fun": "policy.assessment",
      "user": "root",
      "expected": 8,
      "returned": 8,
      "returned_good": 5,
      "returned_failed": 3,
      "create_time": "2019-01-27T19:42:53.087028",
      "user_uuid": "80c67364-cb31-4f4b-972a-e7ea3f752bb8",
      "uuid": "e912435e-2c38-4cfa-aa1e-985c83aa8a22",
      "name": "demo",
      "tgt_uuid": "7f93b928-388b-11e6-b133-346895ecb8f3",
      "schedule_uuid": null,
      "last_update": "2019-01-27T19:40:40.544772",
      "last_assess_jid": "20190127194046350020",
      "last_assess_timestamp": "2019-01-27T19:40:46.323247",
```

(continues on next page)

(continued from previous page)

```

        "last_remed_jid": null,
        "last_remed_timestamp": null
    }]
},
error=None,
warnings=[])

```

get_stats(...)

Returns Dict.

Get statistics for SecOps

Table 107: get_stats() parameters

include_in_response		Stats to include in response. It accepts following values checks_assessed_all_time checks_remediated_all_time checks_assessed_latest checks_remediated_latest assessment_jobs_run remediation_jobs_run policies_created policies_with_schedule policies_with_expired_schedules policies_never_assessed checks_by_benchmark checks_by_benchmark_in_policy checks_by_benchmark_not_in_policy policies_with_exemptions status_all_time status_since_last_assessment check_status_by_benchmark_all_time check_status_by_benchmark_since_last_assessment content_stats
---------------------	--	---

Returns

Response with stats requested.

Examples**Request**

```

from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.get_stats()

```

Response

```

RPCResponse(
  riq=4,
  ret={
    "policies_created": 2,
    "policies_with_schedule": 0,
    "remediation_jobs_run": 9,
    "checks_by_benchmark": {"PCI-DSS-3.2": 44, "CCE": 81, "800-171": 76, "CIS-
↪ CentOS7": 223, "CIS-RHEL7": 222, "CIS": 182, "CIS-CentOS": 1, "CCE-Win16": 111,
↪ "CIS-RHEL": 1, "CIS_CentOS7": 1, "CIS_RHEL7": 1, "800-53": 68, "CIS-Win16": 34,
↪ "CIS-Win10": 31, "CCE-Win10": 95},
    "status_all_time": {"compliant": 71830, "noncompliant": 85882, "notapplicable
↪ ": 1414},
    "content_stats": {
      "uuid": "16386268-f2e1-4ab3-8e46-7f59131dee48",
      "content_benchmarks": 7,
      "success": true,
      "db_benchmarks": 7,
      "ingest_date": "2019-01-27T14:07:49.898504",
      "content_checks": 437,
      "db_checks": 437,
      "errors": null
    },
    "checks_assessed_all_time": 226,
    "checks_remediated_all_time": 226
  },
  error=None,
  warnings=[])

```

get_stats_history(...)

Returns Dict.

Get statistics History for SecOps

Table 108: get_stats_history() parameters

since_time		Filter results by time.
time_unit		Unit of time to filter the results.
sort_by		Sort results by supplied field.
reverse		Set to True to reverse sort order.
page		Return results specified by page number.
limit		Maximum number of results to return.

Returns

Returns stats history.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.get_stats_history(
    since_time=7300,
    time_unit="days"
)
```

Response

```
RPCResponse(
  riq=4,
  ret={
    "results": [{
      "uuid": "4db84e45-974f-4456-8125-57f7533f3d96",
      "stats": {
        "policies_created": 1,
        "policies_with_schedule": 0,
        "remediation_jobs_run": 1,
        .
        .
        .
      }
    }]
  },
  error=None,
  warnings=[])
```

ingest_custom_content(...)

Returns None.

Download Custom Content tarball :param hub: :param filename: :param content_type: application/x-gzip :param blob: binary blob or base64 encoded :return: dict = {}

remediate_policy(...)

Returns Dict.

Remediate one or more checks in a policy

Table 109: remediate_policy() parameters

policy_uuid		UUID of the policy to remediate.
check_uuids		Check UUIDs included in the policy to remediate.
minions		Minions to remediate.
pre_remediation		State file to run before remediation routine.
post_remediation		State file to run after remediation routine.

Response

Job ID for remediation run.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.remediate_policy(
    policy_uuid="3c625016-9dbf-44d8-b97c-17f4361b78a0",
    minions={"master2_master":["master2"]},
    check_uuids=["80492e63-af0b-41c8-9a5b-b202be658561"]
)
```

Response

```
RPCResponse(
  riq=4,
  ret={
    "success": true,
    "errors": [],
    "jid": "20190125002727876899"
  },
  error=None,
  warnings=[])
```

save_exemption(...)

Returns Dict.

Add or update an exemption on a policy

Table 110: save_exemption() parameters

policy_uuid		UUID of the policy.
reason		Reason for exemption.
check_uuids		UUIDs of checks that are exempt.
minion_ids		Minion IDs that are exempt.

Returns

Success or error code from saving exemption.

Examples**Request**

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.save_exemption(
    policy_uuid="e912435e-2c38-4cfa-aa1e-985c83aa8a22",
    reason="Approved by CISO",
    minion_ids=[{"master1_master": ["oracle"]}],
    check_uuids=["70ceb294-0ce0-4144-8e8a-26301609cce1"]
)
```

Response

```
RPCResponse(
    riq=4,
    ret={
        "count": 1,
        "results": [{
            "uuid": "3a776df3-9c4d-4d54-8e1f-dc0c189f0f68",
            "policy_uuid": "e912435e-2c38-4cfa-aa1e-985c83aa8a22",
            "check_uuid": "02b952e3-1d8c-4db3-8771-3edf23e0a0dd",
            "master_id": "master1_master",
            "minion_id": "oracle",
            "reason": "Approved by CISO"
        }]
    },
    error=None,
    warnings=[])
```

save_policy(...)

Returns <class 'uuid.UUID'>.

Save a new security policy or update an existing one.

Saves a policy.

Table 111: save_policy() parameters

name		Name of the security policy.
tgt_uuid		UUID of the target group the security policy applies to.
policy_uuid		UUID of the security policy. This parameter is optional when creating a new policy and required when updating an existing policy.
benchmark_uuids		List of UUIDs of existing security policy benchmarks to include in the policy. The policy will include the checks from each benchmark in the list, minus any passed in <code>check_uuids</code> .
check_uuids		Optional list of security policy checks to include in the policy. If <code>benchmark_uuids</code> is also given, the policy will include only checks that appear in at least one of the specified benchmarks.
exclude_check_uuids		Optional list of security policy checks to exclude from the policy. This is useful for omitting some checks that would otherwise be included as part of the benchmarks passed in <code>benchmark_uuids</code> .
variables		Optional list of security policy check variables to apply to the policy. Each entry in the list is a dict containing three entries: <code>check_uuid</code> (UUID of the check the variable applies to), <code>name</code> (variable name), and <code>value</code> (variable value).
schedule		Optional dict defining schedule for running policy assessment. For details on the schedule structure, see the <code>schedule</code> parameter of the <code>schedule.save()</code> RPC method.

Returns

UUID of the new or updated policy.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.save_policy(
    name="Policy 1",
    tgt_uuid="919193da-604e-456d-87c4-90860f5e8b59",
    benchmark_uuids=[
```

(continues on next page)

(continued from previous page)

```

        "22c8539e-f719-4d6a-9e14-c3c638a7be82",
        "026e3afb-d4a7-45be-b405-4f83fa7a3b3a"
    ]
)

```

Return

```

RPCResponse(
    riq=4,
    ret="de301811-80d2-4a89-bd24-93d025347394",
    error=None,
    warnings=[])

```

Request

```

from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.sec.save_policy(
    name="Policy 2",
    tgt_uuid="61edbf8c-8b5c-483c-845b-ff12280aa171",
    benchmark_uuids=["4eec46a0-d029-4c3a-a8dd-c6cbb86b9659"],
    check_uuids=[
        "55c17549-095a-4dba-9817-0db845385eeb",
        "985b5057-42c8-4c86-8c8e-06d8b3f51472",
        "a2f200b6-6040-46bb-9005-9725e9b65490",
        "c0df2e2d-13df-457a-bbca-cb0cbf495525"
    ],
    variables=[{
        "check_uuid": "985b5057-42c8-4c86-8c8e-06d8b3f51472",
        "name": "_locke.system.service.sshd_maxauthtries.SSHD_CONFIG_MAXAUTHTRIES",
        "value": "3"
    }]
)

```

Response

```
RPCResponse(  
    riq=4,  
    ret="de301811-80d2-4a89-bd24-93d025347394",  
    error=None,  
    warnings=[])
```

1.8.19 settings interface

The `LoadedMod` class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

`delete_auth_config(...)`

Returns None.

Delete the authentication configuration by the provided name

Table 112: `delete_auth_config()` parameters

name		The name of the authentication configuration to delete.
------	--	---

`delete_sso_config(...)`

Returns None.

Get SSO configuration settings

Table 113: `delete_sso_config()` parameters

slug		SSO Slug ID.
------	--	--------------

`end_directory_preview(...)`

Returns None.

Clears out the Redis cache entry for this particular preview job. This does not stop the background task, to stop a background Celery task is somewhat ugly, we can revisit this later if it appears we really need to make sure the task is dead.

get_auth_config_access(...)

Returns Dict.

Return access metadata for this authentication configuration.

Table 114: get_auth_config_access() parameters

config_uuid		UUID referencing desired authentication configuration.
-------------	--	--

get_auth_config(...)

Returns ['typing.Dict', 'typing.List[typing.Dict]'].

Return the authentication configuration by the provided name, or all authentication configurations if name is not specified.

Table 115: get_auth_config() parameters

config_name		Name of the configuration to retrieve. Retrieve all if config_name is None.
include_preview		Include any auth_configs that are marked Preview. Defaults to False. Only matters if config_name is None, if a config_name is passed, always return the config whether it is a preview config or not.

get_config_unprotected(...)

Returns Dict.

Get info for the login banner

get_directory_preview_records(...)

Returns None.

Return the records retrieved from the LDAP directory as collected so far. The return payload will contain the LDAP records under the *entries* key as well as the status of the background job that is querying the directory. The status will appear in the *status* key of the return payload.

get_directory_preview_status(...)

Returns None.

Return a status for the background LDAP query job.

get_sso_backends(...)

Returns None.

Get backend field definitions

Table 116: get_sso_backends() parameters

backend		SSO backend ID
---------	--	----------------

get_sso_config(...)

Returns None.

Get SSO configuration settings

Table 117: get_sso_config() parameters

slug		SSO Slug ID.
------	--	--------------

get_telemetry_status(...)

Returns <class 'bool'>.

Return the status of telemetry collection and reporting to VMware. True means telemetry is being sent to VMware. False means no telemetry data is transmitted.

save_auth_config_access(...)

Returns None.

Save access metadata for this authentication configuration.

Table 118: save_auth_config_access() parameters

config_uuid		UUID referencing desired authentication configuration.
access_payload		Dictionary containing role names as keys and a list of allowed access types as values

save_auth_config(...)

Returns None.

Create or update the authentication configuration by the provided config_name.

Table 119: save_auth_config() parameters

config_name		Name of the auth configuration
**		Dictionary with details defining this authentication configuration.
skip_job_scheduling		If False, no scheduler jobs which maintain synchronization with the Active Directory/LDAP backend are created.

save_sso_config(...)

Returns None.

Save SSO configuration settings

Table 120: save_sso_config() parameters

name		Name of the SSO.
backend		SSO backend.
config		Configuration settings.

set_telemetry_status(...)

Returns None.

Set the status of telemetry permission. This API is should be called by LCM with the value of the CEIP opt-in checkbox, or in the SaaS deployments with the value of the SUDP checkbox whenever the customer changes it.

start_directory_preview(...)

Returns None.

Execute a background task that will return a set of example records from the defined LDAP or Active Directory. Returns a *task_id* that can be used to check on the status of the preview. LDAP records are cached in Redis and can be retrieved with *get_directory_preview_records()*.

sync_auth_config(...)

Returns List.

Get updated information from the LDAP backend on users currently in use in SSE.

Table 121: sync_auth_config() parameters

config_name		Name of the configuration to synchronize.
-------------	--	---

test_auth_config(...)

Returns Dict.

Test Active Directory/LDAP connectability and credentials for the provided authentication configuration details

Table 122: test_auth_config() parameters

details		Connection details for the AD/LDAP backend.
---------	--	---

Returns a dictionary with at least one key, *validation_passed*, for which the value is *True* or *False* depending on the outcome of the validation routines. If validation failed, an extra key, *details* is also included with an explanation on what failed..

1.8.20 stats interface

The `LoadedMod` class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

`db_health(...)`

Returns None.

Get database health status

`get_queue_sizes(...)`

Returns None.

Get Celery queue size

Example:

```
client.api.stats.get_queue_sizes()

RPCResponse(riq=5,
             ret={'lr': 0, 'graincache': 0, 'celery': 0},
             error=None,
             warnings=[])
```

`get_system_metric_sources(...)`

Returns List[str].

Get a list of sources for a particular metric. These sources can be passed to `get_system_metrics()` to filter results.

Table 123: `get_system_metric_sources()` parameters

metric		Metric name to report sources for. Metric names are available in the RPC API discovery payload as metrics.
--------	--	--

Example:

```
client.api.stats.get_system_metric_sources('salt_event_size_bytes')

RPCResponse(riq=6,
             ret=['master1', 'master2'],
             error=None,
             warnings=[])
```

get_system_metrics(...)

Returns Dict.

Get system metrics.

Table 124: get_system_metrics() parameters

daterange		A dictionary containing up to 2 of the fields <code>start</code> , <code>end</code> , and <code>timedelta</code> . <code>start</code> and <code>end</code> are date strings in ISO-8601 format, and <code>timedelta</code> is a time span in seconds. If <code>end</code> is not specified, it defaults to now. If <code>timedelta</code> is not specified, it defaults to the maximum allowable time span (24 hours).
metrics		Limit results to these metric names. Default behavior is to return all metrics. Metric names are available in the RPC API discovery payload as <code>metrics</code> .
sources		Limit results to metrics from these sources. Call <code>get_system_metric_sources()</code> to get available sources for a particular metric. Default behavior is to return metrics from all sources.
reverse		Sort by metric timestamp in ascending (False) or descending (True) order. Default is ascending order (most recent metrics last).

In the return payload, metrics are separated by metric name as shown in the truncated example below. Each data point consists of a numeric `value` and a `name` timestamp.

Example:

```
client.api.stats.get_system_metrics()

RPCResponse(rq=5,
    ret={'count': 3018,
        'results': [
            {'name': 'salt_event_count-master1',
                'series': [
                    {'value': 0.0, 'name': '2020-04-03T17:14:53.879032'},
                    {'value': 3.0, 'name': '2020-04-03T17:15:53.873155'},
                    {'value': 7.0, 'name': '2020-04-03T17:16:53.879273'},
                    # ...
                ]},
            {'name': 'salt_event_size_bytes-master1',
                'series': [
                    {'value': 0.0, 'name': '2020-04-03T17:14:53.879032'},
                    {'value': 94.0, 'name': '2020-04-03T17:14:53.873155'},
                    {'value': 135758.857, 'name': '2020-04-03T17:16:53.879273'},
                    # ...
                ]},
            # ...
        ]},
    error=None,
    warnings=[])
```

null(...)

Returns None.
Null RPC method to test the fastest RPC Api response

1.8.21 test interface

The LoadedMod class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

echo(...)

Returns <class 'str'>.
Echo the message.

Table 125: echo() parameters

message		The message to echo.
---------	--	----------------------

error(...)

Returns <class 'bool'>.
Test how an error is returned under the RPC Api. Simply raises a generic exception.

sleep(...)

Returns <class 'str'>.
Sleep for the amount of time provided.

Table 126: sleep() parameters

seconds		Number of seconds to sleep. Float values permitted.
---------	--	---

warnings(...)

Returns <class 'bool'>.
Test how a warning is returned under the RPC Api.

1.8.22 tgt interface

The **LoadedMod** class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

delete_target_group(...)

Returns None.

Delete one or more target groups.

Table 127: delete_target_group() parameters

tgt_uuid		UUID or list of UUIDs referring to the target groups to delete.
force		Force deletion of target group and scheduled jobs depending on it.

raises FailedResourceDependency The target group has one or more resources that depend on it and *force* is false. The message in the exception identifies the dependent resources.

get_target_group_access(...)

Returns Dict.

Get the access metadata for a target group.

Table 128: get_target_group_access() parameters

tgt_uuid		Retrieve metadata for target group matching this UUID.
----------	--	--

get_target_group_minions(...)

Returns Dict.

Get the set of minions that match a target group according to the currently cached minion data. Minions are keyed by master:

For more information on target groups, see the Aria Automation Config documentation on VMware's Doc Center.

Table 129: get_target_group_minions() parameters

tgt_uuid		The UUID of the target group for which to resolve minions.
----------	--	--

get_target_group(...)

Returns Dict.

Retrieve details for a target group. For more information on target groups, see the Aria Automation Config documentation on VMware's Doc Center.

Table 130: get_target_group() parameters

tgt_uuid		Retrieve details for this target group.
master_id		Retrieve details for master with this ID (not UUID).
pillar_uuid		Retrieve target details associated with this pillar (UUID).
role_uuid		Retrieve targets assigned to role (UUID)
name		Search the target group name for this text.
desc		Search the target group description for this text.
include_pillar_data		Include attached pillar data in the return payload (default False).
sort_by		Sort by this field, either 'name' or 'desc'.
reverse		Return results in the reverse order.
limit		Limit maximum target groups to this number (default 50).
page		Return target groups from this page (offset = page * limit).

Use either `tgt_uuid` or `master_id`, but not both. If `tgt_uuid` is used, other filters are not applied.

get_target_group() example:

```
client.api.tgt.get_target_group()
```

The return payload is a dictionary with the following elements:

```
{
  'count': 100,      # total target group count
  'limit': 50,      # results count
  'results': [...]  # target groups
}
```

Example:

```
client.api.tgt.get_target_group(tgt_uuid='7f93b928-388b-11e6-b133-346895ecb8f3')

RPCResponse(
  riq=5,
  ret= {
    u'results': [
      {u'name': u'All Minions',
       u'tgt': {u'*':
                 {u'tgt_type': u'compound',
                  u'tgt': u'*}}},
      u'uuid': u'7f93b928-388b-11e6-b133-346895ecb8f3',
      u'pillars': [],
      u'metadata': {u'auth':
                    {u'access':
                     {u'Superuser':
```

(continues on next page)

(continued from previous page)

```

        {u'write': False,
         u'read': True,
         u'discover': True,
         u'delete': False},
        u'Administrator':
        {u'write': False,
         u'read': True,
         u'discover': True,
         u'delete': False},
        u'User':
        {u'write': False,
         u'read': True,
         u'discover': True,
         u'delete': False}},
        u'owner':
        {u'username': u'--<{internal-access}>--',
         u'config_name': u'internal',
         u'uuid': u'c74f8bee-5ead-453e-8c86-
↪344a4780c053'}}},
        u'desc': u''}],
        u'count': 1,
        u'limit': 1
    },
    u'error'= None,
    u'warnings'= [])

```

save_target_group(...)

Returns <class 'str'>.

Create a new or update an existing target group.

For more information on target groups, see the Aria Automation Config documentation on VMware's Doc Center.

Table 131: save_target_group() parameters

tgt		Target dictionary. See below for example.
name		Name of the target group
tgt_uuid		UUID for the target group. If empty, generate a new UUID. Otherwise <code>tgt_uuid</code> must point to an existing target group.
desc		Text description of this target group.
pillar_uuids		List of UUIDs referring to related pillar structures.
wait_for_match		If <code>True</code> , block this call until the target group is successfully constructed. This can take some time on large RaaS installations. If <code>False</code> then return immediately.

The `tgt` argument takes a dictionary that looks like

```

{<master_id or '*'>:
 {u'tgt_type': <target type>,
  u'tgt': <target string>}}

```

`tgt_type` is the type of target: `glob`, `grain`, `grain_pcre`, `pillar`, `pillar_pcre`, `list`, `ipcidr`, `pcre`, or `node`.

`tgt` is the actual target. For `glob` this would be any string matching minion IDs and utilizing shell-globbing characters (`*`, `?`, `[]`, etc).

For `compound` see Salt's description of compound target indicators for a full explanation, but an overview is below:

Compound matchers allow very granular minion targeting using any of Salt's matchers. The default matcher is a `glob` match, just as with CLI and top file matching. To match using anything other than a `glob`, prefix the match string with the appropriate letter from the table below, followed by an `@` sign.

Letter	Match Type	Example	Alt Delimiter?
G	Grains <code>glob</code>	G@os:Ubuntu	Yes
E	PCRE Minion ID	E@web\d \. (dev qa prod)\.loc	No
P	Grains PCRE	P@os:(RedHat Fedora CentOS)	Yes
L	List of minions	L@minion1.example.com, minion3.domain.com or bl*.domain.com	No
I	Pillar <code>glob</code>	I@pdata:foobar	Yes
J	Pillar PCRE	J@pdata:^(foo bar)\$	Yes
S	Subnet/IP address	S@192.168.1.0/24 or S@192.168.1.100	No
R	Range cluster	R@%foo.bar	No

Matchers can be joined using boolean `and`, `or`, and `not` operators.

Here is an example for creating a target group with a compound target:

```
wintgtresult = c.api.tgt.save_target_group(
    name='Windows',
    desc='Windows',
    tgt={'*':{'tgt_type':'compound', 'tgt':'G@os_family:Windows'}})
```

`save_target_group()` example:

```
client.api.tgt.save_target_group(tgt={'SSE_master': {'tgt_type': 'grain', 'tgt':
↳ 'G@kernel:Linux'}}), name='NewTestLinux')
```

Update existing target example:

```
client.api.tgt.save_target_group(tgt={'SSE_master': {'tgt_type': 'grain', 'tgt':
↳ 'G@kernel:Linux'}}), name='Linux', tgt_uuid='f2ace00c-4fa0-11e6-88bc-080027a7289c',
↳ desc='newdescription')
```

save_target_group_access(...)

Returns Union[bool, uuid.UUID].

Save the access metadata for a target group.

Table 132: save_target_group_access() parameters

tgt_uuid		Save metadata for target group matching this UUID.
access_payload		Dictionary containing role names as keys and a list of allowed access types as values

1.8.23 vman interface

The LoadedMod class allows for the module loaded onto the sub to return custom sequencing, for instance it can be iterated over to return all functions

assess_policy(...)

Returns <class 'str'>.

Run assessment of advisories in a policy

Table 133: assess_policy() parameters

policy_uuid		Policy UUID to assess.
-------------	--	------------------------

Response

Job ID for assessment run for specified policy.

Examples**Request**

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.assess_policy(
    policy_uuid="1f90f261-1668-486e-a1d3-4dd68ef3020c"
)
```

Response

```
RPCResponse(  
    riq=4,  
    ret={  
        "success": true,  
        "errors": [],  
        "jid": "20190125001746553804"  
    },  
    error=None,  
    warnings=[])
```

commit_third_party_staging_data(...)

Returns None.

Commit Third Party Import Staging data to vman_results

Examples

Request

```
from sseapiclient import APIClient  
client = APIClient('http://localhost', 'root', 'salt')  
client.api.vman.commit_third_party_staging_data(import_uuid='some uuid')
```

Response

```
RPCResponse(  
    riq=4,  
    error=None,  
    warnings=[])
```

delete_advisory_state_link(...)

Returns None.

Delete unsupported advisory to state file link.

Table 134: delete_advisory_state_link() parameters

advisory_state_xref_uuid		UUID of the advisory to state file linkage.
--------------------------	--	---

Returns

None

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.delete_advisory_state_link(advisory_state_xref_uuid='<advisory_state_
↪file_xref_uuid>')
```

Response

```
RPCResponse(
    riq=4,
    ret="e5b4970e-250f-4b4e-b7a5-53df6e1d0448",
    error=None,
    warnings=[])
```

delete_connector(...)

Returns <class 'int'>.

Delete connector for a vendor

Table 135: delete_connector() parameters

vendor		Vendor for which the connector is to be deleted.
--------	--	--

Returns

Number of connectors deleted.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.delete_connector(vendor='tenable')
```

Response

```
RPCResponse(  
    riq=4,  
    ret=1,  
    error=None,  
    warnings=[])
```

delete_exemption_group(...)

Returns <class 'int'>.

Delete a specific exemption group.

Table 136: delete_exemption_group() parameters

exemption_group_uuid		UUID of a specific exemption group to delete.
----------------------	--	---

Returns

Number of deleted exemption groups.

Examples

Request

```
from sseapiclient import APIClient  
client = APIClient('http://localhost', 'root', 'salt')  
client.api.vman.delete_exemption_group(  
    exemption_group_uuid="e912435e-2c38-4cfa-aa1e-985c83aa8a22",  
)
```

Response

```
RPCResponse(  
    riq=4,  
    ret=1  
},  
error=None,  
warnings=[])
```


delete_policy(...)

Returns <class 'bool'>.

Delete an existing vulnerability management policy.

Table 137: delete_policy() parameters

policy_uuid		UUID of the policy to delete.
-------------	--	-------------------------------

Returns

Response code from deleting the policy.

Examples**Request**

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.delete_policy(
    policy_uuid="db1ae70f-768e-4486-adae-b83546a09dea"
)
```

Response

```
RPCResponse(
    riq=4,
    ret=True,
    error=None,
    warnings=[])
```

delete_third_party_import(...)

Returns None.

Delete Third Party Import Staging data

Table 138: delete_third_party_import() parameters

import_uuid		UUID of the imported scan.
-------------	--	----------------------------

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.delete_third_party_staging_data(import_uuid=<some uuid>)
```

Response

```
RPCResponse(
    riq=4,
    error=None,
    warnings=[])
```

download_content(...)

Returns Dict.

Download and ingest content

Download Vulnerability management tarball and expand the files into `hub.opts['cachedir'] + /vman` Once the download and expansion is complete, calls ingest to put them in the raas filesystem

get_advisories(...)

Returns Dict.

Get advisories.

Table 139: get_advisories() parameters

advisory_state_xref_uuid		UUID of the advisory to state file linkage.
--------------------------	--	---

Returns

None

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.delete_advisory_state_link(advisory_state_xref_uuid='<advisory_state_
↪file_xref_uuid>')
```

Response

```
RPCResponse(
  riq=4,
  ret="e5b4970e-250f-4b4e-b7a5-53df6e1d0448",
  error=None,
  warnings=[])
```

get_advisory_minions(...)

Returns Dict.

Get assessment and remediation results for a policy

Table 140: get_advisory_minions() parameters

policy_uuid		Get assessment and remediation results about this policy.
advisory_id		Limit results to advisories matching this advisory ID.
display_name		Limit results to display_names matching this display_name.
install_behavior		Limit results to install_behaviors matching this install_behavior.
minion_id		Limit results to minion matching one of this ID.
master_id		Limit results to master matching one of this ID.
pending_reboot		Limit results to pending_reboot matching 'null'/'true'/'false'
osfullname		Limit results to OS matching name.
ipv4		Limit results to matching IPv4.
ipv6		Limit results to matching IPv6.
action		Filter results by compliance action, assess or remediate.
include_in_response		Choose the blocks that should appear in the response.
severity		Filter results by severity.
advisory_title		Filter results by partial advisory title match.
advisory_type: Filter by advisory type. Valid values		supported, unsupported
cve_id		Filter results by partial CVE ID match.
pkg_name		Filter results by partial package name match.
sort_by		Sort results by supplied field.
reverse		Set to True to reverse sort order.
page		Return results specified by page number.
limit		Maximum number of results to return.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_advisory_minions(
    policy_uuid='26854b29-b122-4cc5-86ca-5d7ae3bdb107'
)
```

get_connector(...)

Returns Dict.

Get connector for a vendor

Table 141: get_connector() parameters

vendor		Vendor for which the connector parameters are required.
--------	--	---

Returns

Parameter values required to authenticate with the vendor API.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_connector(vendor='tenable')
```

Response

```
RPCResponse(
    riq=4,
    ret={'access_key': 'xxx', 'secret_key': 'xxx', 'url': 'https://cloud.tenable.com',
        '__secrets': ['access_key', 'secret_key']}
    error=None,
    warnings=[])
```

get_connector_params(...)

Returns Dict.

Get the parameters required for the connector to authenticate with vendor API.

Table 142: get_connector_params() parameters

vendor		Vendor for which the connector parameters are required.
--------	--	---

Returns

Dictionary containing the parameters required to authenticate with the vendor API.

Examples**Request**

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_connector_params(vendor='tenable')
```

Response

```
RPCResponse(
  riq=4,
  ret={
    'accessKey': None,
    'secretKey': None,
    'days_since': 90,
    'url': 'https://cloud.tenable.com',
    '__secrets': ['accessKey', 'secretKey']
  },
  error=None,
  warnings=[])
```

get_content_stats(...)

Returns <class 'dict'>.

Get the stats for the latest ingested content

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_content_stats()
```

Response

```
RPCResponse(
  riq=4,
  ret={
    "creation_date": "2019-08-26T20:52:18.479319",
    "file_name": "vman_2019-08-26T20:52:18.479319_e5b4970e-250f-4b4e-b7a5-
↪53df6e1d0448.tar.gz.e",
    "ingest_date": "2019-09-24T17:04:07.865149",
    "uuid": "e5b4970e-250f-4b4e-b7a5-53df6e1d0448"
  },
  error=None,
  warnings=[])
```

get_detail_report(...)

Returns None.

Get vulnerability detailed report

Table 143: get_detail_report() parameters

report_uuid		UUID of report to retrieve.
-------------	--	-----------------------------

Return: JSON formatted report.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_detail_report(
  report_uuid="1471bdbc-c495-4211-b6f0-6e442ce4dde6"
)
```

Response

```

RPCResponse(
  riq=4,
  ret={
    "minions": {
      "master2": [
        {
          "usage": {
            "cmd": [],
            "enabled": [],
            "network": [],
            "running": []
          },
          "comment": null,
          "pkg_name": "glibc",
          "severity": "medium",
          "master_id": "master2_master",
          "advisory_id": "CESA-2019:2118",
          "master_uuid": "60522eca-8ad0-43fb-9461-171027e03deb",
          "advisory_uuid": "082e07e7-6abd-4617-b4d9-0800091baa9b",
          "pkg_version_target": "2.17-292.el7",
          "pkg_version_current": "2.17-260.el7_6.6",
          "pkg_version_advisory": "2.17-292.el7"
        },
      ],
    },
    "advisories": {
      "CESA-2016:1064": {
        "info": {
          "cve": {
            "uuid": "027bc255-638b-409b-9e66-613538cc93c8",
            "cve_id": "CVE-2016-2149",
            "cvssv2": "AV:N/AC:L/Au:S/C:P/I:N/A:N",
            "cvssv3": "CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N",
            "references": {
              "REDHAT": {
                "url": "https://access.redhat.com/errata/RHSA-
→2016:1064",
                "name": "RHSA-2016:1064"
              }
            },
            "cvssv2_base": 4.0,
            "cvssv3_base": 6.5,
            "description": "Red Hat OpenShift Enterprise 3.2 allows
→remote authenticated users",
            "published_date": "2016-06-08T17:59:00",
            "cvssv2_severity": "MEDIUM",
            "cvssv3_severity": "MEDIUM",
            "cvssv2_impact_score": 2.9,
            "cvssv3_impact_score": 2.9
          },
          "uuid": "18b865af-47ad-4786-8124-d0748a0bb1f2",
          "title": "CentOS OpenShift Enterprise 3.2 security, bug fix, and
→enhancement update",
          "packages": {
            "el7": {

```

(continues on next page)

(continued from previous page)

```

        "kibana": "4.1.2-2.el7aos",
        "lucene": "4.10.4.redhat_1-5.el7",
        "rubygem-fluent-plugin-kubernetes_metadata_filter-doc":
↪ "0.12.0-1.el7aos"
    },
    "severity": "high",
    "advisory_id": "CESA-2016:1064",
    "description": "OpenShift Enterprise by Red Hat is the company's
↪ cloud computing Platform",
    },
    "minions": [
        "master2"
    ]
},
{
    "meta": {
        "policy_uuid": "a5aa646e-f7fe-4d99-945b-7e40b929df7b",
        "policy_name": "awesome policy",
        "target_group_name": "All Minions",
        "creation_date": "2019-09-24T17:11:57.449781",
        "function": "policy.assessment",
        "critical": 42,
        "high": 104,
        "medium": 136,
        "low": 0,
        "none": 0
    }
},
error=None,
warnings=[])

```

get_exemption_groups(...)

Returns Dict.

Get exemption groups.

An exemption group is a combination of minions and advisories that will not be remediated, with an associated reason - perhaps your infrastructure requires an unsecure package, etc.

Table 144: get_exemption_groups() parameters

policy_uuid		UUID of the associated policy.
exemption_group_uuid		UUID of a specific exemption group to return.

Returns

List of exemptions groups for the policy.

Examples

Request

```

from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_exemption_groups(
    policy_uuid="e912435e-2c38-4cfa-aa1e-985c83aa8a22",
)

```

Response

```

RPCResponse(
  riq=4,
  ret={
    "count": 1,
    "results": [{
      "uuid": "3a776df3-9c4d-4d54-8e1f-dc0c189f0f68",
      "policy_uuid": "e912435e-2c38-4cfa-aa1e-985c83aa8a22",
      "advisory_uuid": "02b952e3-1d8c-4db3-8771-3edf23e0a0dd",
      "master_id": "master1_master",
      "minion_id": "oracle",
      "reason": "Approved by CIS0"
    }]
  },
  error=None,
  warnings=[])

```

get_exemptions(...)

Returns Dict.

Get exemptions

Table 145: get_exemptions() parameters

policy_uuid		UUID of the policy.
exemption_uuid		UUID of the exemption.

Returns

List of matching advisories (either by policy_uuid or exemption_id or both)

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_exemptions(
    policy_uuid="e912435e-2c38-4cfa-aa1e-985c83aa8a22",
    exemption_uuid="3a776df3-9c4d-4d54-8e1f-dc0c189f0f68"
)
```

Response

```
RPCResponse(
  riq=4,
  ret={
    "count": 1,
    "results": [{
      "uuid": "3a776df3-9c4d-4d54-8e1f-dc0c189f0f68",
      "policy_uuid": "e912435e-2c38-4cfa-aa1e-985c83aa8a22",
      "advisory_uuid": "02b952e3-1d8c-4db3-8771-3edf23e0a0dd",
      "master_id": "master1_master",
      "minion_id": "oracle",
      "reason": "Approved by CIS0"
    }]
  },
  error=None,
  warnings=[])
```

get_policies(...)

Returns Dict.

Get summary information on vulnerability management policies

Table 146: get_policies() parameters

policy_uuids		List of vulnerability management policy UUIDs. This parameter is optional when retrieving a list of policies.
names		List of vulnerability management policy names. This parameter is optional when retrieving a list of policies.
tgt_uuids		List of policy target group UUIDs. This parameter accepts one or more tgt_uuids.
sort_by		Sort results by supplied field.
reverse		Set to True to reverse sort order.
page		Return results specified by page number.
limit		Maximum number of policies to return.

Returns

List of policies matching the request criteria.

Examples

Request 1

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_policies(
    policy_uuids=[
        "a5aa646e-f7fe-4d99-945b-7e40b929df7b",
    ],
)
```

Response 1

```
RPCResponse(
  riq=4,
  ret={
    "count": 1,
    "results": [
      {
        "uuid": "a5aa646e-f7fe-4d99-945b-7e40b929df7b",
        "name": "awesome policy",
        "tgt_uuid": "7f93b928-388b-11e6-b133-346895ecb8f3",
        "tgt_name": "All Minions",
        "schedule_uuid": null,
        "last_update": "2019-07-18T19:55:05.617862",
```

(continues on next page)

(continued from previous page)

```

        "last_assess_jid": "20190719154009438906",
        "last_assess_timestamp": "2019-07-19T15:40:09.402146",
        "last_remed_jid": null,
        "last_remed_timestamp": null,
        "stats": {
            "CRITICAL": 1
        },
        "schedule": null,
        "minions_assessed": 4
    }
]
},
error=None,
warnings=[])

```

get_policy_run_history(...)

Returns Dict.

Get assessment and remediation runs against the policy

Table 147: get_policy_run_history() parameters

policy_uuid		UUID of the policy.
jid		Job ID
fun		Job function name.
state		Filter by one or more statuses of the assessment or remediation runs.
user		Name of the user who ran the policy.
expected		Number of expected minions to return a result.
returned		Number of minions which returned a result.
sort_by		Sort results by supplied field.
reverse		Set to True to reverse sort order.
page		Return results specified by page number.
limit		Maximum number of results to return.

Returns

Examples

Request

```

from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_policy_run_history(
    policy_uuid="e912435e-2c38-4cfa-aa1e-985c83aa8a22"
)

```

Response

```

RPCResponse(
  riq=4,
  ret={
    "count": 2,
    "results": [{
      "jid": "20190127194046350020",
      "state": "completed_failures",
      "fun": "policy.assessment",
      "user": "root",
      "expected": 8,
      "returned": 8,
      "returned_good": 5,
      "returned_failed": 3,
      "create_time": "2019-01-27T19:42:53.087028",
      "user_uuid": "80c67364-cb31-4f4b-972a-e7ea3f752bb8",
      "uuid": "e912435e-2c38-4cfa-aa1e-985c83aa8a22",
      "name": "demo",
      "tgt_uuid": "7f93b928-388b-11e6-b133-346895ecb8f3",
      "schedule_uuid": null,
      "last_update": "2019-01-27T19:40:40.544772",
      "last_assess_jid": "20190127194046350020",
      "last_assess_timestamp": "2019-01-27T19:40:46.323247",
      "last_remed_jid": null,
      "last_remed_timestamp": null
    }]
  },
  error=None,
  warnings=[]
)

```

get_reports(...)

Returns Dict.

Get vulnerability summary reports

Table 148: get_reports() parameters

policy_uuid		UUID of policy. Optional, when None, daily aggregated reports are returned.
fun		Filter by function, should be <code>policy.assessment</code> or <code>policy.remediate</code> .
start_date		Get reports on or after date.
end_date		Get reports on or before date.
sort_by		Sort on field, should be <code>date</code> only.
reverse		Reverse list order. <code>true</code> or <code>false</code> .
page		Page number, default is 0.
limit: Number of results returned. Default		40.

Returns: List of reports

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_reports()
```

Response

```
RPCResponse(
  riq=4,
  ret={
    "count": 1,
    "results": [
      {
        "report_uuid": "53b79e83-1269-43fb-8e0f-f4d4c201df30",
        "date": "2019-09-24",
        "creation_date": "2019-09-24T17:11:57.449781",
        "policy_uuid": "a5aa646e-f7fe-4d99-945b-7e40b929df7b",
        "fun": "policy.assessment",
        "critical": 42,
        "high": 104,
        "medium": 136,
        "low": 0,
        "none": 0,
        "policy_name": "awesome policy",
        "target_group_name": "All Minions",
        "top_advisories": [
          {
            "name": "systemd security, bug fix, and enhancement update",
            "uuid": "b4c3b40a-9f11-42e4-8777-df541f20f816",
            "score": 9.8,
            "minions": 6
          },
          {
            "name": "systemd security, bug fix, and enhancement update",
            "uuid": "91a0b654-cb2b-48eb-b27c-b085fd035ffa",
            "score": 9.8,
            "minions": 6
          },
          {
            "name": "perl security update",
            "uuid": "4c4549b4-564a-446f-af0e-fd642df58dc8",
            "score": 9.8,
            "minions": 2
          },
          {
            "name": "libssh2 security, bug fix, and enhancement update",
            "uuid": "9acffe23-1344-4d0a-895b-e4cc63d4bcd3",
            "score": 9.1,
            "minions": 2
          },
          {
```

(continues on next page)

(continued from previous page)

```

        "name": "curl security and bug fix update",
        "uuid": "7e3c8b03-38da-40a5-b0bf-e689216e3dda",
        "score": 9.1,
        "minions": 4
    }
}
],
error=None,
warnings=[])

```

get_selected_for_import_cnt(...)

Returns None.

Get the number of minion/packages selected for import

Examples**Request**

```

from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_selected_for_import_cnt(import_uuid='some uuid', advisory_
↳ type=VmanAdvisoryType.SUPPORTED)

```

Response

```

RPCResponse(
    riq=4,
    ret={'count': 1},
    error=None,
    warnings=[])

```

get_stats(...)

Returns Dict.

Get statistics for Vulnerability management policies

get_stats_history(...)

Returns Dict.

Get statistics History for vulnerability management

Table 149: get_stats_history() parameters

since_time		Filter results by time.
time_unit		Unit of time to filter the results.
sort_by		Sort results by supplied field.
reverse		Set to True to reverse sort order.
page		Return results specified by page number.
limit		Maximum number of results to return.

Returns

Returns stats history.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_stats_history(
    since_time=7300,
    time_unit="days"
)
```

get_supported_advisories(...)

Returns Dict.

Get supported advisories from third Party Import Staging data

Table 150: get_supported_advisories() parameters

import_uuid		UUID of the imported scan.
advisory_id		Filter by this advisory id. Optional.
minion_id		Filter by this minion_id. Optional.
sort_by		Sort by <i>advisory_id</i> or <i>minion_id</i> . Optional.
reverse		Sort in descending order when True. Optional.
page		Show the results of this page. Starts at 0. Optional.
limit		Limit the results to these many records. Default 50. Optional.

Returns

Dictionary of records with the supported advisories.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_supported_advisories(import_uuid="e5b4970e-250f-4b4e-b7a5-
↳ 53df6e1d0448")
```

Response

```
RPCResponse(
  riq=4,
  ret={'count': 1,
    'results': [{'advisory_uuid': '408675a7-336f-4b15-9bba-7737c263974c',
      'advisory_id': 'CESA-2019:2136',
      'advisory_name': 'Dummy advisory - 9876',
      'comment': '',
      'duration': 0.0,
      'fun': 'policy.assessment',
      'import_uuid': 'b1a050a3-51a5-4885-974a-91f4264e2510',
      'master_id': 'master1',
      'master_uuid': 'f635ffa4-9ebc-407a-a90c-b9f3b7e96864',
      'minion_id': 'minion1',
      'pkg_name': 'libcheetoh4',
      'pkg_version_advisory': '7.58.0-2ubuntu3',
      'selected_for_import': False,
      'severity': 'high',
      'start_time': None,
      'tpa_uuid': 'c00f526f-73f8-408b-903e-2600b03a16a9',
      'uuid': '060df507-12a9-4909-b1be-9fc39aba0130'}]},
  error=None,
  warnings=[])
```

get_third_party_import(...)

Returns Dict.

Get the details of a third party assessment import.

Table 151: get_third_party_import() parameters

import_uuid		UUID of the third party import.
-------------	--	---------------------------------

Returns

Metadata of the imported scan.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_third_party_import(import_uuid='e5b4970e-250f-4b4e-b7a5-
↳ 53df6e1d0448')
```

Response

```
RPCResponse(
  riq=4,
  ret={
    "import_uuid": "e5b4970e-250f-4b4e-b7a5-53df6e1d0448",
    "policy_uuid": "fd678e0e-859a-4b4e-b7a5-43ab6d1f0635",
    "tp_source": "tenable",
    "start_time": "2019-09-24T17:04:07.865149",
    "end_time": "2019-09-24T17:04:07.865149",
    "creation_date": "2019-09-24T20:04:07.865149",
    "analysis_stats": {"supported": 0, "missing_advisory_id": 0, "missing_minion
↳ ": 43},
    "state": "complete",
  },
  error=None,
  warnings=[])
```

get_unsupported_advisories(...)

Returns Dict.

Get unsupported advisories from third Party Import Staging data

Table 152: get_unsupported_advisories() parameters

import_uuid		UUID of the imported scan.
advisory_id		Filter by this advisory id. Optional.
ip		Filter by this ip. Optional.
advisory_type: Type of advisory requested. Optional. Valid values		unmatched, unsupported.
sort_by		Sort by <i>advisory_id</i> or <i>ip</i> . Optional.
reverse		Sort in descending order when True. Optional.
page		Show the results of this page. Starts at 0. Optional.
limit		Limit the results to these many records. Default 50. Optional.

Returns

Dictionary of records with the unsupported advisories with comments.

Examples

Request

```

from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_unsupported_advisories(import_uuid="e5b4970e-250f-4b4e-b7a5-
↳53df6e1d0448")

```

Response

```

RPCResponse(
  riq=4,
  ret={'count': 59,
    'results': [{ 'analysis_comment': 'cannot find minion with ipv4 10.0.2.7',
      'import_uuid': '246e7c39-a6a3-418c-ac83-c3d99a7daaa6',
      'importable': False,
      'tp_advisory_id': 'CESA-2019:2118',
      'tp_extra_data': {'@pluginFamily': 'Settings',
        '@pluginID': '19506',
        '@pluginName': 'Nessus Scan Information',
        '@port': '0',
        '@protocol': 'tcp',
        '@severity': '0',
        '@svc_name': 'general',
        'canvas_package': 'CANVAS',
        'cve': ['CVE-2018-12641',
          'CVE-2018-12697',
          'CVE-2018-1000876'],
        'cvss3_base_score': '7.8',
        'cvss3_temporal_score': '4.9',
        'cvss3_temporal_vector': 'E:F/RL:O/RC:C',
        'cvss3_vector': 'AV:L/AC:L/PR:L/UI:N/S:U/C:H/
↳I:H/A:H',
        'cvss_base_score': '5.0',
        'cvss_temporal_score': '4.1',
        'cvss_temporal_vector': 'E:F/RL:OF/RC:C',
        'cvss_vector': 'AV:N/AC:L/Au:N/C:N/I:N/A:P',
        'description': 'This plugin displays, for
↳each tested host...',
        'exploit_available': 'true',
        'exploit_framework_canvas': 'true',
        'patch_publication_date': '2019/08/29',
        'plugin_modification_date': '2019/03/06',
        'plugin_output': 'Information about this scan
↳...',
        'plugin_publication_date': '2005/08/26',
        'plugin_type': 'summary',
        'risk_factor': 'None',

```

(continues on next page)

(continued from previous page)

```

        'script_version': '1.92',
        'see_also': 'http://cpe.mitre.org/',
        'solution': 'n/a',
        'synopsis': 'This plugin displays information
    ↪ ...',
        'tag': {'Credentialed_Scan': 'true',
        ↪ 'HOST_END': 'Mon Oct 07 21:01:47 2019
        ↪ 'HOST_START': 'Mon Oct 07 20:10:02
        ↪ 'LastAuthenticatedResults':
        ↪ 'TAG':
        ↪ 'host-fqdn': 'master2',
        ↪ 'host-ip': '10.0.2.7',
        ↪ 'host-uuid': '8837e032-e1be-4fcb-acb9-
        ↪ 'hostname': 'master2',
        ↪ 'local-checks-PROTO': 'local',
        ↪ 'mac-address': '02:42:0A:64:00:0B',
        ↪ 'netbios-name': 'master2',
        ↪ 'operating-system': 'Linux Kernel '
        ↪ '4.9.184-linuxkit
        ↪ 'on CentOS Linux '
        ↪ 'release 7.6.1810
        ↪ '(Core)',
        ↪ 'os': 'linux',
        ↪ 'policy-used': 'Basic Agent Scan',
        ↪ 'system-type': 'general-purpose'},
        'vuln_publication_date': '2018/06/22',
        'xref': 'CESA:2019:2075'},
        'tp_ip': '10.0.2.7',
        'uuid': '040f1995-90bb-4776-979b-dab5bfd96ccb'
    }
  ],
  error=None,
  warnings=[])

```

get_vendors_with_connector_support(...)

Returns List.

Get the list of vendors having connector support in raas.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.get_vendors_with_connector_support()
```

Response

```
RPCResponse(
  riq=4,
  ret=['tenable'],
  error=None,
  warnings=[])
```

import_scan(...)

Returns Dict.

Import third party assessment into RaaS.

Table 153: import_scan() parameters

thirdparty_source		Third party sources like tenable where the scan data (xml) is imported from.
data		Contents of the scan data (xml).
policy_uuid		UUID of the policy to import the scan data.
policy_name		Name of the policy to import the scan data.

Returns

A dictionary containing import_uuid and stats about the scan data imported.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.import_scan(policy_name='third_party',
                           thirdparty_source='tenable',
                           data='<xmldata>goes here</xmldata>')
```

Response

```
RPCResponse(  
    riq=4,  
    ret={  
        "import_uuid": "e5b4970e-250f-4b4e-b7a5-53df6e1d0448",  
        "supported": 0,  
        "missing_advisory_id": 0,  
        "missing_minion": 43  
    },  
    error=None,  
    warnings=[])
```

import_scan_via_api(...)

Returns Dict.

Import third party assessment into RaaS via API integration.

Table 154: import_scan_via_api() parameters

thirdparty_source		Third party sources like tenable where the scan data is imported from.
policy_uuid		UUID of the policy to import the scan data.

Returns

A dictionary containing import_uuid

Examples

Request

```
from sseapiclient import APIClient  
client = APIClient('http://localhost', 'root', 'salt')  
client.api.vman.import_scan(policy_uuid='<policy_uuid>',  
                           thirdparty_source='tenable')
```

Response

```
RPCResponse(  
    riq=4,  
    ret={  
        "import_uuid": "e5b4970e-250f-4b4e-b7a5-53df6e1d0448",  
    },  
    error=None,  
    warnings=[])
```

ping(...)

Returns <class 'bool'>.

Check if Vulnerability Management feature is available.

remediate_policy(...)

Returns <class 'str'>.

Remediate one or more advisories in a policy

Table 155: remediate_policy() parameters

policy_uuid		UUID of the policy to remediate.
check_uuids		Check UUIDs included in the policy to remediate.
minions		Minions to remediate.
pre_remediation		State file to run before remediation routine.
post_remediation		State file to run after remediation routine.

Response

Job ID for remediation run.

Examples**Request**

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.remediate_policy(
    policy_uuid="3c625016-9dbf-44d8-b97c-17f4361b78a0",
    minions={"master2_master": ["master2"]},
    check_uuids=["80492e63-af0b-41c8-9a5b-b202be658561"],
    pre_remediation={"saltenv": "base", "location": "/states/pre_run.sls"},
    post_remediation={"saltenv": "other", "location": "/states/post_run_thing.sls"}
)
```

Response

```
RPCResponse(
  riq=4,
  ret={
    "success": true,
    "errors": [],
    "jid": "20190125002727876899"
  },
  error=None,
  warnings=[])
```

save_advisory_state_link(...)

Returns <class 'uuid.UUID'>.

Save unsupported advisory to state link.

Table 156: save_advisory_state_link() parameters

policy_uuid		UUID of the vulnerability management policy.
advisory_id		ID of the unsupported advisory.
state_file_uuid		UUID of the state file to be attached to the advisory.
link_to_policy		State file will be linked only to this policy when True. Otherwise, the state file will be available across all policies.

Returns

UUID of the state file attached to the unsupported advisory.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.save_advisory_state_link(policy_uuid='<policy_uuid>',
                                         advisory_id='<advisory_id>',
                                         state_file_uuid='<state_file_uuid>',
                                         link_to_policy=False)
```

Response

```
RPCResponse(
  riq=4,
  ret="e5b4970e-250f-4b4e-b7a5-53df6e1d0448",
  error=None,
  warnings=[])
```

save_connector(...)

Returns None.

Save connector for a vendor

Table 157: save_connector() parameters

vendor		Vendor for which the connector parameters are required.
connector_params		Dictionary of connector parameters for the vendor.

tenable :accessKey: Access Key from <https://cloud.tenable.com> :secretKey: Secret Key from <https://cloud.tenable.com>
 :days_since: Import vulnerabilities reported for the past so many days.

Returns

Dictionary containing the parameters required to authenticate with the vendor API.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.save_connector(vendor='tenable',
    connector_params={'accessKey': 'xxx', 'secretKey': 'xxx', 'url': 'https://cloud.
    ↪tenable.com'})
```

Response

```
RPCResponse(
    riq=4,
    ret=None
    error=None,
    warnings=[])
```

save_exemption_group(...)

Returns <class 'uuid.UUID'>.

Add or update an exemption on a policy

Table 158: save_exemption_group() parameters

policy_uuid		UUID of the policy.
reason		Reason for exemption.
advisory_uuids		UUIDs of checks that are exempt.
minion_ids		Minion IDs that are exempt.

Returns

UUID of exemption group or error code from saving exemption.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.save_exemption_group(
    policy_uuid="e912435e-2c38-4cfa-aa1e-985c83aa8a22",
    reason="Approved by CISO",
    minion_ids=[{"master1_master":["oracle"]}],
    advisory_uuids=["70ceb294-0ce0-4144-8e8a-26301609cce1"]
)
```

Response

```
RPCResponse(
    riq=4,
    ret="3a776df3-9c4d-4d54-8e1f-dc0c189f0f68",
    error=None,
    warnings=[])
```

save_policy(...)

Returns <class 'uuid.UUID'>.

Save a new vulnerability management policy or update an existing one.

Table 159: save_policy() parameters

name		Name of the vulnerability management policy.
tgt_uuid		UUID of the target group the policy applies to.
policy_uuid		UUID of the policy. This parameter is optional when creating a new policy and required when updating an existing policy.
schedule		Optional dict defining schedule for running policy assessment. For details on the schedule structure, see the <code>schedule</code> parameter of the <code>schedule.save()</code> RPC method.

Returns

UUID of the new or updated policy.

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.save_policy(
    name="Policy 1",
    tgt_uuid="919193da-604e-456d-87c4-90860f5e8b59",
)
```

Return

```
RPCResponse(
    riq=4,
    ret="de301811-80d2-4a89-bd24-93d025347394",
    error=None,
    warnings=[])
```

update_all_third_party_staging_data(...)

Returns None.

Update All Third Party Import Staging data by import_uuid

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.update_all_third_party_staging_data(import_uuid='some uuid',
                                                    selected_for_import=True)
```

Response

```
RPCResponse(
    riq=4,
    error=None,
    warnings=[])
```

update_third_party_staging_data(...)

Returns None.

Update Third Party Import Staging data

Examples

Request

```
from sseapiclient import APIClient
client = APIClient('http://localhost', 'root', 'salt')
client.api.vman.update_third_party_staging_data(tpmp_uuids=['some uuid', 'another uuid
↪'],
                                                selected_for_import=True)
```

Response

```
RPCResponse(
  riq=4,
  error=None,
  warnings=[])
```