CVP Uploader Documentation

Release 0.3.3

EMEA

Aug 22, 2019

Overview:

CHAPTER 1

Arista Cloudvision Portal Python scripts

This repository provides a set of python scripts to interact with Arista Cloudvision server. All of them are based on cvprac library to interact using APIs calls between your client and CVP interface.

1.1 Container manager for CoudVision

Generic script to manage containers on Arista Cloudvision server. It is based on cvprac library to interact using APIs calls between your client and CVP interface.

Script filename: `cvp-container-manager`

Supported Features

- **Check** if container exists on CVP.
- **Create** a container on CVP topology
- **Delete** a container from CVP topology.
- Move a devices to an existing container.

1.2 CloudVision Configlet manager

Generic script to manage configlet on an Arista Cloudvision server. It is based on cvprac library to interact using APIs calls between your client and CVP interface.

Script filename: `cvp-configlet-manager`

Supported Features

- **Update** existing remote configlet.
- Execute configlet update.
- Wait for task result.

- **Delete** configlet from server.
- **Creating** a new Configlet.
- **attach** and **detach** devices to/from existing configlet.
- Creating **change-control**.
- **Scheduling** change-control.
- Collect tasks to attach to change-control.

1.3 CloudVision tasks Management

Generic script to interact with tasks on an Arista Cloudvision server. It is based on cvprac library to interact using APIs calls between your client and CVP interface.

Script filename: `cvp-task-manager`

Supported Features

- **Execute** All pending tasks

1.3.1 Known Issues

Due to a change in CVP API, change-control needs to get snapshot referenced per task. Current version of `cvprack` does not support it in version 1.0. (Issue #75)

Fix is available in develop version. To install development version, use pip:

```
$ pip install git+https://github.com/aristanetworks/cvprac.git@develop
```

> Only required if you want to play with change-control

CHAPTER 2

Getting Started

```
$ pip install git+https://github.com/titom73/arista-cvp-scripts.git

# Update your credential information
$ cat <<EOT > env.variables.sh
export CVP_HOST='xxx.xxx.xxx.xxx'
export CVP_PORT=443
export CVP_PROTO='https'
export CVP_USER='username'
export CVP_PASS='password'
export CVP_TZ='Europe/Paris'
export CVP_COUNTRY='France'
EOT

# run script
$ cvp-configlet-manager -j actions.json
```

CHAPTER 3

License

Project is published under BSD License.

CHAPTER 4

Ask question or report issue

Please open an issue on Github this is the fastest way to get an answer.

CHAPTER 5

Contribute

Contributing pull requests are gladly welcomed for this repository. If you are planning a big change, please start a discussion first to make sure we'll be able to merge it.

CHAPTER 6

Installation:

6.1 Installation

Script can be used with 2 different installation method:

- git clone for testing. In this case it is recommended to use a virtual-environment
- Python PIP module to install binary directly to your syste. A virtual-environment is also recommended for testing purpose.

6.1.1 Installation with PIP

```
$ pip install git+https://github.com/titom73/arista-cvp-scripts.git

# Update your credential information
$ cat <<EOT > env.variables.sh
export CVP_HOST='xxx.xxx.xxx.xxx'
export CVP_PORT=443
export CVP_PROTO='https'
export CVP_USER='username'
export CVP_PASS='password'
EOT

$ source env.variables

# run script to create a configlet
$ cvp-configlet-manager -j examples/actions.configlet.create.json

# run script to play with containers
$ cvp-container-manager -j examples/actions.containers.json
```

6.1.2 Git Clone

It is highly recommended to use Python virtual environment for testing

```
$ git clone https://github.com/titom73/arista-cvp-scripts.git

$ python setup.py

# Update your credential information
$ cat <<EOT > env.variables.sh
export CVP_HOST='13.57.194.119'
export CVP_PORT=443
export CVP_PROTO='https'
export CVP_USER='username'
export CVP_PASS='password'
EOT
```

6.1.3 Known Issue

Due to a change in CVP API, change-control needs to get snapshot referenced per task. Current version of `cvprack` does not support it in version 1.0.1

Fix is available in develop version. To install development version, use pip:

```
$ pip install git+https://github.com/aristanetworks/cvprac.git@develop
```

6.1.4 Build development environment

It is highly recommended to use Python virtual environment for testing

```
$ git clone https://github.com/titom73/arista-cvp-scripts.git
$ cd arista-cvp-scripts

# Create virtualenv
$ virtualenv -p /usr/bin/python2.7 .venv

# Load virtualenvironment
$ source .venv/bin/activate

# Install module in develop mode for auto reload changes
$ python setup.py develop

# Install Python linter
$ pip install flake8

# Install pre-commit hook
$ ln -s -f ../../.ci/pre-commit .git/hooks/pre-commit
```

6.2 Script options

Script provides a set of different options and all can be set by using *SHELL* environment variables or *CLI* parameters.

6.2.1 Options within shell environment

By default, script will lookup for a set of variables in your environment:

- `CVP_HOST`: Hostname or IP address of CVP server
- `CVP_PORT`: CVP port to use to communicate with API engine. Default is 443
- `CVP_PROTO`: Transport protocol to discuss with CVP. Default is HTTPS
- `CVP_USER`: Username to use for CVP connection
- `CVP_PASS`: Password to use for CVP connection
- `LOG_LEVEL`: Script verbosity. Default is info
- `CVP_TZ`: Timezone used to configure change-control
- `TZ_COUNTRY`: Country to use in change-control configuration.
- `CERT_VALIDATION`: Whether or not activate SSL Cert validation. Default is False to manage self signed certificates.

In your shell, execute following commands:

```
export CVP_HOST='IP_ADDRESS_OF_CVP_SERVER'
export CVP_PORT=443
export CVP_PROTO='https'
export CVP_USER='YOUR_CVP_USERNAME'
export CVP_PASS='YOUR_CVP_PASSWORD'
export CVP_TZ=France
export CVP_COUNTRY='France'
```

A script example is available in the repository for informational purpose

It can be configured in your `~/.bashrc` or in VARIABLES of a CI/CD pipeline as well.

6.2.2 Options from the CLI

This approach overrides options defined in your shell environment

```
$ cvp-configlet-manager-h

usage: cvp-configlet-uploader.py [-h] [-v] [-c CONFIGLET] [-u USERNAME]
                                 [-p PASSWORD] [-s CVP] [-d DEBUG_LEVEL]
                                 [-j JSON]

Configlet Uploader to CVP

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  -c CONFIGLET, --configlet CONFIGLET
                        Configlet path to use on CVP
  -u USERNAME, --username U SERNAME
                        Username for CVP
  -p PASSWORD, --password PASSWORD
                        Password for CVP
  -s CVP, --cvp CVP     Address of CVP server
  -d DEBUG_LEVEL, --debug_level DEBUG_LEVEL
```

(continues on next page)

(continued from previous page)

```
                        Verbose level (debug / info / war ning / error /
                        critical)
  -j JSON, --json JSON  File with list of actions to execute)
```

CHAPTER 7

Arista Cloudvision Portal Python scripts

This repository provides a set of python scripts to interact with Arista Cloudvision server. All of them are based on cvprac library to interact using APIs calls between your client and CVP interface.

7.1 Container manager for CoudVision

Generic script to manage containers on Arista Cloudvision server. It is based on cvprac library to interact using APIs calls between your client and CVP interface.

Script filename: `cvp-container-manager`

Supported Features

- **Check** if container exists on CVP.
- **Create** a container on CVP topology
- **Delete** a container from CVP topology.
- Move a devices to an existing container.

7.2 CloudVision Configlet manager

Generic script to manage configlet on an Arista Cloudvision server. It is based on cvprac library to interact using APIs calls between your client and CVP interface.

Script filename: `cvp-configlet-manager`

Supported Features

- **Update** existing remote configlet.
- Execute configlet update.
- Wait for task result.

- **Delete** configlet from server.
- **Creating** a new Configlet.
- **attach** and **detach** devices to/from existing configlet.
- Creating **change-control**.
- **Scheduling** change-control.
- Collect tasks to attach to change-control.

7.3 CloudVision tasks Management

Generic script to interact with tasks on an Arista Cloudvision server. It is based on cvprac library to interact using APIs calls between your client and CVP interface.

Script filename: `cvp-task-manager`

Supported Features

- **Execute** All pending tasks

7.3.1 Known Issues

Due to a change in CVP API, change-control needs to get snapshot referenced per task. Current version of `cvprack` does not support it in version 1.0. (Issue #75)

Fix is available in develop version. To install development version, use pip:

```
$ pip install git+https://github.com/aristanetworks/cvprac.git@develop
```

> Only required if you want to play with change-control

CHAPTER 8

Getting Started

```
$ pip install git+https://github.com/titom73/arista-cvp-scripts.git

# Update your credential information
$ cat <<EOT > env.variables.sh
export CVP_HOST='xxx.xxx.xxx.xxx'
export CVP_PORT=443
export CVP_PROTO='https'
export CVP_USER='username'
export CVP_PASS='password'
export CVP_TZ='Europe/Paris'
export CVP_COUNTRY='France'
EOT

# run script
$ cvp-configlet-manager -j actions.json
```

CHAPTER 9

License

Project is published under BSD License.

CHAPTER 10

Ask question or report issue

Please open an issue on Github this is the fastest way to get an answer.

CHAPTER 11

Contribute

Contributing pull requests are gladly welcomed for this repository. If you are planning a big change, please start a discussion first to make sure we'll be able to merge it.

11.1 How-To and use-cases:

11.1.1 How to use configilet manager

Script can be use to manage configlet on a CloudVision (CVP) server

To manage all actions to run on a CVP server is by using a JSON file to list a set of actions. This json file is provided to the script by using `-json` trigger on CLI.

JSON file is an array of entries where every single entry in JSON file describe a task to run:

```
[
  {
      //task 1
  },
  {
      //task 2
  }
]
```

Current version of code support all the actions listed below:

- Create a configlet
- Update content of a configlet
- Delete a configlet from Cloud Vision Portal
- Add a device to an existing configlet
- Remove a device from an existing configlet

Note: For the first 2 options, a local content for any configlet shall be present to push content to Cloud Vision. In other scenario, only the name of the configlet targetting by your action should be defined.

Create a configlet with add task

To create a new configlet on CVP server, `JSON` file shall have the following structure:

```
{
    "name": "new CVP Configlet",
    "type": "configlet",
    "action": "add",
    "configlet": "configlet.examples/VLANsTEMP",
    "apply": false,
    "devices": [
        "leaf1",
        "leaf2",
        "leaf3"
    ]
}
```

Where **keys** have description below:

- `name`: A name for the task. it is only a local name and it is not used on CVP side.
- `type`: shall be **configlet**. It define what kind of entry to manage on CVP. in this case, we are talking about a configlet.
- `action`: Action to run on configlet. As we want to create a new one, action shall be **add**
- `configlet`: Path to the configlet. Remember that file name will be used as configlet name.
- `apply`: define wether or not we should deploy this configlet to devices. if set to **false**, then a change-control or manual action should be done later.
- `devices`: An array of devices hostname configured on CVP where to attache configlet.

Update content of a configlet with update task

To update an existing configlet on CVP server, `JSON` file shall have the following structure:

```
{
    "name": "new CVP Configlet",
    "type": "configlet",
    "action": "update",
    "configlet": "configlet.examples/VLANs",
    "apply": true
}
```

Where **keys** have description below:

- `name`: A name for the task. it is only a local name and it is not used on CVP side.
- `type`: shall be **configlet**. It define what kind of entry to manage on CVP. in this case, we are talking about a configlet.
- `action`: Action to run on configlet. As we want to create a new one, action shall be **update**
- `configlet`: Path to the configlet. Remember that file name will be used as configlet name.

- `apply`: define wether or not we should deploy this configlet to devices. if set to **false**, then a change-control or manual action should be done later.
- `devices`: An array of devices hostname configured on CVP where to attache configlet.

Note: *Note:* If configlet is not already configured on your CloudVision server, then script try to create it. Creation requires a list devices configured in this specific task.

Delete a configlet with delete task

To delete an existing configlet on CVP server, `JSON` file shall have the following structure:

```
{
    "name": "new CVP Configlet",
    "type": "configlet",
    "action": "delete",
    "configlet": "configlet.examples/VLANsTEMP",
    "apply": true
}
```

Where **keys** have description below:

- `name`: A name for the task. it is only a local name and it is not used on CVP side.
- `type`: shall be **configlet**. It define what kind of entry to manage on CVP. in this case, we are talking about a configlet.
- `action`: Action to run on configlet. As we want to create a new one, action shall be **delete**
- `configlet`: Path to the configlet. Remember that file name will be used as configlet name.
- `apply`: define wether or not we should deploy this configlet to devices. if set to **false**, then a change-control or manual action should be done later.
- `devices`: An array of devices hostname configured on CVP where to attache configlet.

Remove a device from configlet with remove-device task

To remove a device from a configlet on CVP server, `JSON` file shall have the following structure:

```
{
    "name": "new CVP Configlet",
    "type": "configlet",
    "action": "remove-devices",
    "configlet": "configlet.examples/VLANsTEMP",
    "apply": false,
    "devices": [
        "leaf3"
    ]
}
```

Where **keys** have description below:

- `name`: A name for the task. it is only a local name and it is not used on CVP side.
- `type`: shall be **configlet**. It define what kind of entry to manage on CVP. in this case, we are talking about a configlet.

- `action`: Action to run on configlet. As we want to create a new one, action shall be **remove-devices**
- `configlet`: Path to the configlet. Remember that file name will be used as configlet name.
- `apply`: define wether or not we should deploy this configlet to devices. if set to **false**, then a change-control or manual action should be done later.
- `devices`: An array of devices hostname to remove from the configlet.

Attach device to a configlet with add-device task

To attach a device or a list of devices to a configlet on CVP server, `JSON` file shall have the following structure:

```
{
    "name": "new CVP Configlet",
    "type": "configlet",
    "action": "add-devices",
    "configlet": "configlet.examples/VLANsTEMP",
    "apply": false,
    "devices": [
        "leaf3",
        "leaf1"
    ]
}
```

Where **keys** have description below:

- `name`: A name for the task. it is only a local name and it is not used on CVP side.
- `type`: shall be **configlet**. It define what kind of entry to manage on CVP. in this case, we are talking about a configlet.
- `action`: Action to run on configlet. As we want to create a new one, action shall be **add-devices**
- `configlet`: Path to the configlet. Remember that file name will be used as configlet name.
- `apply`: define wether or not we should deploy this configlet to devices. if set to **false**, then a change-control or manual action should be done later.
- `devices`: An array of devices hostname to remove from the configlet.

Change-control building

To delete an existing configlet on CVP server, `JSON` file shall have the following structure:

```
{
    "name": "Change Control to deploy last update",
    "type": "change-control",
    "schedule": "2019-03-15-12-30",
    "snapid": "snapshotTemplate_9_4694793526491",
    "apply": true,
},
```

Where **keys** have description below:

- `name`: A name for the task. it is only a local name and it is not used on CVP side.
- `type`: shall be **change-control**. It define what kind of entry to manage on CVP. in this case, we are talking about a change-control.

- `schedule`: *optional* entry to schedule execution of change control. if not set, change-control is executed 3 minutes after entry registration
- `apply`: If set to `true`, then, script will schedule change-control execution using `schedule` field or 3 minutes after change-control creation. If set to `false`, change control must be executed manually.

Some other options are also available for this action:

- `timezone`: Timezone of the server to manage scheduling. By default, it is set to `Europe/Paris` timezone.
- `country`: Country where CVP is for time managemement as well. By default it is set to `France`.

Warning: Timezone should be defined according time-zone configured on the machine you are running the script. In the meantime, your Cloud Vision server shall be NTP synced with correct timezone as well.

11.1.2 How to use container manager

Script uses a JSON file to describe list of actions to run on CloudVision server. This json file is provided to the script by using `` `-json` `` trigger on CLI.

JSON file is an array of entries where every single entry in JSON file describe a task to run:

```
[
  {
      //task 1
  },
  {
      //task 2
  }
]
```

Current version of code support all the actions listed below:

- Create a container in CoudVision topology
- Move a list of devices to an existing container.
- Delete a container from CloudVision topology.

Create a container within CVP Topology:

You can create a container in CloudVision topology using a JSON like below.

JSON example:

```
{
    "name": "Create container",
    "type": "container",
    "action": "create",
    "container": "Test Container",
    "parent": "Tenant"
}
```

Where **keys** have description below:

- `name`: A name for the task. it is only a local name and it is not used on CVP side.

- `type`: shall be **container**. It define what kind of entry to manage on CVP. in this case, we are talking about a container.
- `action`: Action to run on configlet. As we want to attach devices to container, action shall be **creation**
- `container`: Name of existing container where devices will be attached.
- `parent`: Name of parent container. It is value you have in your toipology. By default, container will be created under **Tenant**

Warning: This action execute task directly and there is no way to just provisionned and execute action later or manually.

Example outputs:

```
--------------------

2019-04-30 13:51:51 INFO     creation of container with name Test Container attached␣
↪to Tenant
2019-04-30 13:51:52 INFO     Connected to 54.219.174.143
2019-04-30 13:51:52 INFO     *************
2019-04-30 13:51:52 INFO     Start working with Test Container
2019-04-30 13:51:52 INFO     initializing a container object for Test Container
2019-04-30 13:51:52 INFO     Version [u'2018', u'2', u'2']
2019-04-30 13:51:52 INFO     Setting API version to v2
2019-04-30 13:51:54 WARNING  container Test Container not found
2019-04-30 13:51:54 INFO     create container on CVP server
2019-04-30 13:51:54 INFO     start creation of container attached to Tenant
```

Delete a container from CVP Topology:

You can delete a container in CloudVision topology using a JSON like below.

JSON example:

```
{
    "name": "Create container",
    "type": "container",
    "action": "destroy",
    "container": "Test Container",
    "parent": "Tenant"
}
```

Where **keys** have description below:

- `name`: A name for the task. it is only a local name and it is not used on CVP side.
- `type`: shall be **container**. It define what kind of entry to manage on CVP. in this case, we are talking about a container.
- `action`: Action to run on configlet. As we want to attach devices to container, action shall be **destroy**
- `container`: Name of existing container where devices will be attached.

- `parent`: Name of parent container. It is value you have in your toipology. By default, container will be created under **Tenant**

Note: To execute this action, your container should not contain any attached device. if some are still attached, process will stop.

> **Warning:** This action execute task directly and there is no way to just provisionned and execute action later or manually.

Example outputs:

```
--------------------

2019-04-30 14:17:36 INFO     destruction of container with name Test Container
2019-04-30 14:17:37 INFO     Connected to 54.219.174.143
2019-04-30 14:17:37 INFO     *************
2019-04-30 14:17:37 INFO     Start working with Test Container
2019-04-30 14:17:37 INFO     initializing a container object for Test Container
2019-04-30 14:17:37 INFO     Version [u'2018', u'2', u'2']
2019-04-30 14:17:37 INFO     Setting API version to v2
2019-04-30 14:17:41 INFO     destroy container from CVP server
2019-04-30 14:17:41 INFO     start process to delete container Test Container
```

Move devices to an existing container:

Script provides a mechanism to move devices to an existing container. JSON syntax to support such operation is provided below:

JSON example:

```
{
      "name": "Change CVX to EVPN",
      "type": "container",
      "action": "attach-device",
      "container": "CVX",
      "apply": true,
      "devices": [
          "leaf1",
          "leaf2",
          "cvx01"
      ]
  }
```

Where **keys** have description below:

- `name`: A name for the task. it is only a local name and it is not used on CVP side.
- `type`: shall be **container**. It define what kind of entry to manage on CVP. in this case, we are talking about a container.
- `action`: Action to run on configlet. As we want to attach devices to container, action shall be **attach-device**

- `container`: Name of existing container where devices will be attached.
- `apply`: define wether or not we should deploy this configlet to devices. if set to **false**, then a change-control or manual action should be done later by user.
- `devices`: An array of devices hostname configured on CVP to move to `container`.

Example outputs:

```
--------------------

2019-04-30 10:21:54 INFO     device leaf1 is going to be moved to CVX
2019-04-30 10:21:54 INFO     device leaf2 is going to be moved to CVX
2019-04-30 10:21:54 INFO     device cvx01 is going to be moved to CVX
2019-04-30 10:21:55 INFO     Connected to 54.219.174.143
2019-04-30 10:21:55 INFO     *************
2019-04-30 10:21:55 INFO     Start working with CVX
2019-04-30 10:21:55 INFO     initializing a container object for CVX
2019-04-30 10:21:55 INFO     Version [u'2018', u'2', u'2']
2019-04-30 10:21:55 INFO     Setting API version to v2
2019-04-30 10:21:59 INFO     check is devices are already part of container
2019-04-30 10:21:59 INFO     device is not part of that container -- moving forward
2019-04-30 10:21:59 INFO     device is not part of that container -- moving forward
2019-04-30 10:21:59 CRITICAL device is already part of that container -- skipping
2019-04-30 10:21:59 INFO     >---
2019-04-30 10:21:59 INFO     starting process to attach a list of device to CVX
2019-04-30 10:21:59 INFO     >---
2019-04-30 10:21:59 INFO     create change to move leaf1 to CVX
2019-04-30 10:22:03 INFO     task created on CVP: 250
2019-04-30 10:22:03 INFO     >---
2019-04-30 10:22:03 INFO     create change to move leaf2 to CVX
2019-04-30 10:22:06 INFO     task created on CVP: 251
2019-04-30 10:22:06 INFO     >---
2019-04-30 10:22:06 CRITICAL device already attached to CVX
2019-04-30 10:22:06 INFO     >---
2019-04-30 10:22:06 INFO     run pending tasks to related to container CVX
2019-04-30 10:22:06 INFO      -> execute task ID: 250
2019-04-30 10:22:08 INFO       * Wait for task completion (status: ACTIVE) / waiting␣
↪for 0 sec
2019-04-30 10:22:09 INFO       * Wait for task completion (status: ACTIVE) / waiting␣
↪for 1 sec
2019-04-30 10:22:10 INFO       * Wait for task completion (status: ACTIVE) / waiting␣
↪for 2 sec
2019-04-30 10:22:12 INFO       * Wait for task completion (status: COMPLETED) /␣
↪waiting for 3 sec
2019-04-30 10:22:12 INFO      -> task 250 status : COMPLETED
2019-04-30 10:22:12 INFO      -> execute task ID: 251
2019-04-30 10:22:13 INFO       * Wait for task completion (status: ACTIVE) / waiting␣
↪for 0 sec
2019-04-30 10:22:14 INFO       * Wait for task completion (status: ACTIVE) / waiting␣
↪for 1 sec
2019-04-30 10:22:15 INFO       * Wait for task completion (status: ACTIVE) / waiting␣
↪for 2 sec
2019-04-30 10:22:17 INFO       * Wait for task completion (status: COMPLETED) /␣
↪waiting for 3 sec
2019-04-30 10:22:17 INFO      -> task 251 status : COMPLETED
```

11.2 Code documentation

11.2.1 cvprac_abstraction

cvprac_abstraction package

cvprac_abstraction.**config_read**(*config_file='actions.json'*)
Read JSON configuration.

Load information from JSON file defined in `config_file` First, method check if file exists or not and then try to load content using `json.load()` If file is not a JSON or if file does not exist, method return None

Data structure to read:

```
[
    {
        "name": "Change CVX to EVPN",
        "type": "container",
        "action": "attach-device",
        "container": "CVX",
        "apply": true,
        "devices": [
            "leaf1",
            "leaf2",
            "cvx01"
        ]
    },
    ...
]
```

Parameters **config_file** (*str*) – Path to the configuration file

Returns Json structure with all actions to execute

Return type json

cvprac_abstraction.**connect_to_cvp**(*parameters*, *log_level='WARNING'*)
Create a CVP connection.

parameters option should at least contain following elements:

- username
- password
- cvp (server IP or DNS hostname)

Parameters

- **parameters** (*dict*) – Object with all information to create connection
- **log_level** (*str*) – Log level to use for CvpClient logger. Default is WARNING.

Returns cvp client object

Return type cvprac.CvpClient()

cvprac_abstraction.**load_constant**(*key_name*, *default='UNSET'*, *verbose=False*)
Set up constant value from OS Environment.

Help to define CONSTANT from OS Environment. If it is not defined, then, fallback to default value provided within parameters

:Example:

```
>>> USERNAME = load_constant(key_name='USERNAME_1', default='myUser')
>>> print USERNAME
>>> myUsername
```

Parameters

- **key_name** (*string*) – VAR to lookup in os.environment
- **default** (*str, optional*) – Default value to use if key_name is not defined. By default set to UNSET
- **verbose** (*bool, optional*) – Boolean to activate verbos mode

Returns Value to use to configure variable

Return type str

Submodules

cvprac_abstraction.cvpChangeControl module

class cvprac_abstraction.cvpChangeControl.**CvpChangeControl**(*cvp_server*, *name='Automated_Change_Control'*)

Bases: object

Change-control class to provide generic method for CVP CC mechanism.

Change Control structure is based on:

- A name to identify change
- A list of tasks already created on CVP and on pending state
- **An optional scheduling. If no schedule is defined,** then task will be run 3 minutes after creatio of CC

List of public available methods:

add_task()
Append a task to self._list_changes

get_tasks()
Return list of of available tasks for this CC

get_list_changes()
Return list of tasks attached to this CC

create()
Create change-control on CVP server

Example

```
>>> from cvprac_abstraction import CVP
>>> from cvprac_abstraction import connect_to_cvp
>>> from cvprac_abstraction.cvpConfiglet import CvpChangeControl
>>>
>>> parameters['cvp'] = '127.0.0.1'
>>> parameters['username'] = 'arista'
>>> parameters['password'] = 'arista'
>>>
>>> client = connect_to_cvp(parameters)
>>>
>>> change_control = CvpChangeControl(cvp_server=client, name='MyChanegControl')
>>> result = change_control.create(tz=timezone,
                                   country='FR',
                                   schedule=True,
                                   schedule_at='2019-03-01-12h00',
                                   snap_template="snapshotTemplate_9_4694793526491
↪",
                                   change_type='Custom', stop_on_error="true")
>>>
```

Warning:

- Change Control execution is not running snapshot before and after with cvprac 1.0.1

__init__(*cvp_server*, *name='Automated_Change_Control'*)

Class Constructor.

Build class content with followinactivities:

- save cvp_server information
- save name for CC
- instanciate list for tasks
- Collect tasks available from CVP

Parameters

- **cvp_server** (*CvpClient*) – CVP Server information
- **name** (*str*) – Optional - Name of the Change Control. Default is `Automated_Change_Control`

_build_change_dictionnary(*order_mode='linear'*)

Build ordered list to schedule changes.

CVP Change Control expect a list with an order to run tasks. By default, all tasks are executed at the same time. But using order_mode set to incremental every task will be scheduled sequentially in this change-control

Parameters **order_mode** (*str*) – Optional - Method to build task list. Shall be `linear` or `incremental`.

Note: Only linear has been tested.

_retrieve_tasks()
Extract tasks from CVP Server.

Connect to CVP server and collect tasks in pending state These tasks are saved in self._available structure dedicated to pending tasks.

add_task(*task*)
Add a tasks to available list.

This task attach this new tasks to the pending tasks list.

Parameters **task** (*str*) – TaskID from CVP server

create(*mode='linear'*, *country='France'*, *tz='Europe/Paris'*, *schedule=False*, *schedule_at=''*, *snap_template='1708dd89-ff4b-4d1e-b09e-ee490b3e27f0'*, *change_type='Custom'*, *stop_on_error='true'*)
Create a change-control.

Parameters

- **mode** (*str*) – Optional - method to order tasks (default : linear)
- **country** (*str*) – Optional - Country requested by CVP API (default:France)
- **tz** (*str*) – Optional - Timezone required by CVP (default: Europe/Paris)
- **schedule** (*bool*) – Optional - Enable CC scheduling (default: False)
- **schedule_at** (*str*) – Optional - Time to execute CC if scheduled
- **snap_template** (*str*) – Optional - Snapshot template ID to run before / after tasks
- **change_type** (*str*) – Optional - CVP definition for CC Might be Custom or Rollback. (default: Custom)
- **stop_on_error** (*str*) – Optional - boolean string to stop CVP on errors

Returns CVP creation result (None if error occurs)

Return type dict

get_list_changes(*mode='linear'*)
Return list of tasks and their execution order.

Parameters **mode** (*str*) – Information about tasks scheduling. Shall be `linear` or `incremental`.

Note: Only linear has been tested.

Returns List of changes and their order

Return type list

get_tasks(*refresh=False*)
Provide list of all available tasks.

Return list of all tasks getting from CVP and/or attached with add_task method.

Parameters **refresh** (*bool*) – Optional - Make a call to CVP to get latest list of tasks

Returns List of available tasks found in this CC

Return type list

cvprac_abstraction.cvpConfiglet module

class cvprac_abstraction.cvpConfiglet.**CvpConfiglet**(*cvp_server*, *configlet_file=None*, *configlet_name=None*)

Bases: object

Configlet class to provide generic method to manage CVP configlet.

Data Structure

Configlet structure is a name based dictionnary with following keys:

- `name`: Name of configlet. This name is built from filename
- `file`: Complete path of the local configlet file
- `content`: Local Configlet content read from `configlet['file']`
- **`key`: Key ID defined by CVP to identify configlet.** it is found by our instance during update, addition or deletion
- **`devices`: List of devices structure compliant** with `CvpApi.get_device_by_name()` It can be found by using `CvpInventory` object.

List of attributes:

`_cvp_server`
cvprac.CvpClient() object to manage CVP connection

`_devices_configlet`
List of devices attached to configlet

`_configlet`
Dictionary with all configlet information: `name`, `file`, `content`, `key`, `devices`

`_cvp_found`
Boolean to get status of configlet on CVP: True if configlet is on server, False other cases

List of public available methods:

`get_devices`()
Get list of devices for this specific configlet

`update_configlet`()
Start update process for that configlet. Do not deploy content to devices

`deploy_configlet`()
Start configlet creation process. Do not deploy content to devices

`delete_configlet`()
Start configlet deletion process. Do not deploy content to devices

`deploy`()
Deploy (add/update) change to a single device

`deploy_bulk`()
Deploy (add/update) change to all devices

`on_cvp`()
Inform about configlet available on CVP

Example

```
>>> from cvprac_abstraction import CVP
>>> from cvprac_abstraction import connect_to_cvp
>>> from cvprac_abstraction.cvpConfiglet import CvpConfiglet
>>>
>>> parameters['cvp'] = '127.0.0.1'
>>> parameters['username'] = 'arista'
>>> parameters['password'] = 'arista'
>>>
>>> client = connect_to_cvp(parameters)
>>>
>>> my_configlet = CvpConfiglet(cvp_server=client,configlet_file='/path/to/
↪configlet')
>>>
>>> my_configlet.update_configlet()
>>>
>>> my_configlet.deploy_bulk()
```

Note: This class use calls to `cvprac` to get and push data to CVP server.

__init__(*cvp_server*, *configlet_file=None*, *configlet_name=None*)
Class Constructor.

Parameters

- **cvp_server** (*CvpClient*) – CvpClient object from cvprack. Gives methods to manage CVP API
- **configlet_file** (*str*) – Path to configlet file.

_configlet_init()
Create an empty dict for configlet.

_configlet_lookup()
Check if a configlet is already present on CVP.

Check if CVP has already a configlet configured with the same name. If yes return True and report key under self._configlet['key'] If no, return False

Returns Return `True` or `False` if configlet name is already configured on CVP

Return type bool

_retireve_devices()
Get list of devices attached to the configlet.

If configlet exists, then, retrieve a complete list of devices attached to it.

Returns List of devices from CVP

Return type list

_task_init()
Create an empty dict for task.

_wait_task(*task_id*, *timeout=10*)
Wait for Task execution.

As API call is asynchronous, task will run avec after receiving a status. This function implement a wait_for to get final status of a task As we have to protect against application timeout or task issue, a basic timeout has been implemented

Parameters

- **task_id** (*str*) – ID of the task provided by self._get_task_id()
- **timeout** (*int*) – optional - Timeout to wait for before assuming task failed
- **Returns** –
- **--------** –
- **dict** – Last status message collected from the server

add_device(*device_hostnames*)
Remove device(s) from a configlet.

Remove device from configlet and create a task on CVP to remove configuration generated by configlet from device. For every hostname defined in devices_hostnames, a lookup is done to get a complete data set for that device and a call to remove device is sent.

> **Warning:** This function never send a call to execute task. it is managed by logic out of that object
>
> **Arguments:**
>
> **devices_hostnames {list} – List of devices hostname to remove from** the configlet.

delete_configlet()
Delete a configlet from CVP.

To protect, function first check if configlet exists, if not, we stop and return to next action out of this function. Remove configlet from all devices where it is configured Then if configlet exist, remove configlet from CVP DB

Returns `True` if able to remove configlet / False otherwise

Return type bool

deploy(*device*, *schedule_at=None*, *task_timeout=10*)
Deploy One configlet to One device.

This function manage a deployment this configlet to a given device already attached to the configlet.

Parameters

- **device** (*dict*) – dict representing a device
- **schedule_at** (*str*) – Optional - scheduler to run deployment at a given time
- **task_timeout** (*int*) – Optional - Timeout for task execution default is 10 seconds

> **Warning:** `schedule_at` option is not yet implemented and shall not be used

Returns message from server

Return type dict

deploy_bulk(*device_list=None*, *schedule_at=None*, *task_timeout=10*)
Run configlet deployment against all devices.

Run configlet deployment over all devices attached to this configlet. Every single deployment are managed by function self.deploy()

Parameters

- **device_list** (*list*) – List of devices if it is set to None, then, fallback is to use devices discover initially
- **at** (*str*) – Optional scheduler to run deployment at a given time
- **task_timeout** (*int*) – Optional - Timeout for task execution. Default is 10 seconds

> **Warning:** `schedule_at` option is not yet implemented and shall not be used

Returns A list of tasks executed for the deployment

Return type list

deploy_configlet(*device_hostnames*)
Create configlet on CVP with content from object.

Create a new configlet on CVP server and attached it to all devices you provide in your JSON file. Device attachement is managed with a CvpInventory call to get all information from CVP. It means you just have to provide existing hostname in your JSON

Each time a device is attached to configlet on CVP, it is also added in CvpConfiglet object for futur use

Parameters **devices_hostname** (*list*) – List of hostname to attached to configlet

get_configlet_info()
To share configlet information.

Returns dictionnary with configlet information

Return type dict

get_devices(*refresh=False*)
To share list of devices attached to the configlet.

If list is empty or if refresh trigger is active, function will get a new list of device from self._retireve_devices() Otherwise, just send back list to the caller

Parameters **refresh** (*bool*) – Update device list from CVP (Optional)

Returns List of devices from CVP

Return type list

name()
Expose name of the configlet.

Returns Name of configlet built by `__init__`

Return type str

on_cvp()
Expose flag about configlet configured on CVP.

Return True if configlet is configured on CVP and can be updated. If configlet is not present, then, False

Returns True if configlet already configured on CVP, False otherwise

Return type bool

remove_device(*devices_hostnames*)
Remove device(s) from a configlet.

Remove device from configlet and create a task on CVP to remove configuration generated by configlet from device. For every hostname defined in devices_hostnames, a lookup is done to get a complete data set for that device and a call to remove device is sent.

> **Warning:** This function never send a call to execute task. it is managed by logic out of that object
>
> **Arguments:**
>
> **devices_hostnames {list} – List of devices hostname to remove from** the configlet.

update_configlet()
Update configlet on CVP with content from object.

Check if configlet is configured on CVP server before pushing an update. If configlet is not there, then, stop method execution.

Returns **str**

Return type message from server with result

cvprac_abstraction.cvpContainer module

class cvprac_abstraction.cvpContainer.**CvpContainer**(*name*, *cvp_server*)
Bases: object

Class to manage Container on CVP.

Centralize a abstraction layer of CVPRAC to manage actions related to container.

List of public available methods:

create()
Create a new container on CloudVision Platform

destroy()
Delete an existing with no attached devices container on CloudVision Platform

is_device_attached()
Poll CVP to know if a device is already part of given container

get_info()
Get container's information from CVP server

attach_device()
Create a task to attach a given device to container

attach_device_bulk()
Create a list of tasks to attach many devices to container

run_pending()
Execute all pengind tasks created by cvpContainer object.

Example

```
>>> from cvprac_abstraction import CVP
>>> from cvprac_abstraction import connect_to_cvp
>>> from cvprac_abstraction.cvpConfiglet import CvpContainer
>>>
```

(continues on next page)

(continued from previous page)

```
>>> parameters['cvp'] = '127.0.0.1'
>>> parameters['username'] = 'arista'
>>> parameters['password'] = 'arista'
>>>
>>> client = connect_to_cvp(parameters)
>>>
>>> container = CvpContainer(name='My New Container', cvp_server=client)
>>>
>>> container.create(parent_name='My Root Container')
```

Note: This class use calls to `cvprac` to get and push data to CVP server.

__init__(*name*, *cvp_server*)
Class Constructor.

Parameters

- **name** (*str*) – container's name to look for on CloudVision server
- **cvp_server** (*cvprack.CvpClient()*) – Object in charge of sending API calls to CVP server.

Returns

Return type None

_container_id(*name=None*)
Get Container ID based on its name.

Parameters **name** (*str, optional*) – Container name to get ID, by default None

Returns container ID configured on CVP

Return type str

_container_info(*name=None*)
Pull CVP to get container information.

Execute a call against CVP to get a dict of information.

Structure is:

```
{
    u'dateTimeInLongFormat': 1513002053415,
    u'key': u'container_8_2864853689536',
    u'mode': u'expand',
    u'name': u'CVX',
    u'root': False,
    u'undefined': False,
    u'userId': u'arista'
}
```

Parameters **name** (*[type], optional*) – Name of container to pull. If not set, name of container used for this instance is configured, by default None

Returns Structure sent back by CVP

Return type dict

_get_devices()
Get list of devices attached to container.

Extract information from CVP to get complete list of devices attached to this container on CVP. Result is saved part of this object.

_wait_task(*task_id*, *timeout=10*)
Wait for Task execution.

As API call is asynchronous, task will run avec after receiving a status. This function implement a wait_for to get final status of a task As we have to protect against application timeout or task issue, a basic timeout has been implemented

Parameters

- **task_id** (*str*) – ID of the task provided by self._get_task_id()
- **timeout** (*int*) – optional - Timeout to wait for before assuming task failed

Returns Last status message collected from the server

Return type dict

attach_device(*hostname*, *deploy=False*)
Move device to container

Move device within container represented in this object. This method create a task to be executed later by user or by the script itself.

Parameters

- **hostname** (*str*) –

  Hostname to move to this container. Complete data set is pulling from CVP if device exists and not attached to this container already.

- **deploy** (*bool, optional*) – Boolean to manage deployment. Not used in this function, by default False

Returns Task ID created by the change on CVP.

Return type str

attach_device_bulk(*hostname_list*, *deploy=False*)
Attach a list of device to existing container.

Get a list of hostname to move to current container. For every hostname, a call to CVP is sent to get device's information and build structure to move it to appropriate container.

Parameters

- **hostname_list** (*list*) – List of device hostname to attach to container.
- **deploy** (*bool, optional*) – Trigger to execute tasks generated during the attach phase, by default False

create(*parent_name='Tenant'*)
Create a container on CVP.

Implement workflow to create a container on CVP. Manage following actions: - Collect Parent container information - Create container.

Parameters **parent_name** (*str, optional*) – Name of parent container to use to attach container, by default 'Tenant'

destroy (*parent_name='Tenant'*)
Remove a container from CVP topology.

Parameters **parent_name** (*str, optional*) – Name of the parent container, by default "Tenant"

Returns Return Nothing

Return type None

get_info ()
Return container's information.

Return container's information pulling from CloudVision server.

Structure is:

```
{
    u'dateTimeInLongFormat': 1513002053415,
    u'key': u'container_8_2864853689536',
    u'mode': u'expand',
    u'name': u'CVX',
    u'root': False,
    u'undefined': False,
    u'userId': u'arista'
}
```

Returns Container information from CVP

Return type dict

is_device_attached (*hostname*)
Test wether or not a device is part of container.

Test if device hostname is already attached to this container. it is based on list provided by self._get_devices()

Parameters **hostname** (*str*) – Hostname to search in container.

Returns True if device is part of container, False if not found.

Return type boolean

run_pending (*task_timeout=10*)
Execute pending tasks related to container

Run tasks created when you change container. It does not manage tasks from other objects.

```
>>> status = my_container.run_pending()
>>> print status
[{id:200, status: completed}, {id:201, status: completed}]
```

Parameters **task_timeout** (*int, optional*) – timer to wait for task completion, by default 10

Returns A list of dictionary where every entry is result of a task:

Return type list()

cvprac_abstraction.cvpInventory module

class cvprac_abstraction.cvpInventory.**CvpInventory**(*cvp_server=<cvprac.cvp_client.CvpClient object>*)

Bases: object

CVP Inventory Class.

Get complete inventory from CVP and expose some functions to get data. It is RO only and nothing is pushed to CVP with this object.

__init__(*cvp_server=<cvprac.cvp_client.CvpClient object>*)
Class Constructor.

Instantiate an Inventory with a REST call to get device list

Parameters **cvp_server** (*cvprack.CvpClient()*) – Your CVP Rack server

get_device_dict(*name*)
Get information for a given device.

Parameters **name** (*str*) – Hostname to lookup

Returns Complete dictionnary sent by CVP

Return type dict

get_devices()
Give a dict of all devices.

Returns **dict**

Return type All devices attached to CVP inventory

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

A

C