CVP Uploader Documentation

Release 0.9.3

EMEA

Apr 29, 2019

Contents:

CHAPTER 1

Configlet uploader to CVP

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

Supported Features

- **Update** existing remote configlet.
- Execute configlet update.
- Wait for task result.
- **Delete** configlet from server.
- **Creating** a new Configlet.
- Add and remove devices to/from existing configlet.
- Creating **change-control**.
- **Scheduling** change-control.
- Collect tasks to attach to change-control.

Complete documentation available on read the doc

1.1 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
```

1.2 Getting Started

```
$ pip install git+https://github.com/titom73/configlet-cvp-uploader.git

# 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'
export CVP_TZ='France'
export CVP_COUNTRY='France'
EOT

# run script (assuming VLANs configlet is present on CVP)
$ cvp-configlet-uploader -c VLANs
```

1.3 License

Project is published under BSD License.

1.4 Ask question or report issue

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

1.5 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 2

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.

2.1 Installation with PIP

```
$ pip install git+https://github.com/titom73/configlet-cvp-uploader.git

# 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

$ source env.variables.sh

# Create Local configlet
$ cat <<EOT > VLANs
vlan 12
!
vlan 34
!
vlan 73
!
EOT
```

(continues on next page)

(continued from previous page)

```
# run script (assuming VLANs configlet is present on CVP)
$ cvp-configlet-uploader -c VLANs
```

2.2 Git Clone

It is highly recommended to use Python virtual environment for testing

```
$ git clone https://github.com/titom73/configlet-cvp-uploader.git

$ pip install -r requirements.txt

# 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

$ source env.variables.sh

# Create Local configlet
$ cat <<EOT > VLANs
vlan 12
!
vlan 34
!
vlan 73
!
EOT

# run script (assuming VLANs configlet is present on CVP)
$ python bin/cvpConfigletUploader.py -c VLANs
```

2.3 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
```

CHAPTER 3

Script options

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

3.1 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.

3.2 Options from the CLI

This approach overrides options defined in your shell environment

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

CHAPTER 4

How to use configilet uploader

Script can be use in 2 different ways to manage configlet on a CloudVision (CVP) server:

- Use a CLI option to point to configlet to update.
- Use a json file to configure a set of actions to execute against a CVP server.

4.1 Use script parameters to update

For a short demo, it can be useful to just update and deploy content of an existing configlet configured on a CVP server. To do that, use `--configlet` option from your CLI and then point you local version of your configlet.

Warning: This approach should be used only to validate script execution. Features are not all available in this way and you can just update a configlet with with no deployment.

```
$ python cvp-configlet-uploader.py -c configlet.examples/VLANs

--------------------

2019-02-28 13:23:37 INFO     task Short path update is going to update configlet.
↪examples/VLANs
2019-02-28 13:23:37 INFO     Connected to 13.56.115.112
2019-02-28 13:23:37 INFO     *************
2019-02-28 13:23:37 INFO     Starting working with configlet.examples/VLANs
2019-02-28 13:23:37 INFO     Configlet [VLANs] found on 13.56.115.112
2019-02-28 13:23:37 INFO     Get list of applied devices from server
2019-02-28 13:23:38 INFO     Version [u'2018', u'2', u'2']
2019-02-28 13:23:38 INFO     Setting API version to v2
2019-02-28 13:23:39 INFO     Start looking for devices attached to [VLANs]
2019-02-28 13:23:39 INFO       > Configlet [VLANs] is applied to spine1 with␣
↪sysMacAddr 2c:c2:60:56:df:93
```

(continues on next page)

(continued from previous page)

```
2019-02-28 13:23:40 INFO       > Configlet [VLANs] is applied to leaf4 with␣
↪sysMacAddr 2c:c2:60:b5:96:d9
[...]
```

In this scenario, we assume configlet VLANs is already deployed and applied on a group of devices. In this case, we can see configlet is attached to **spine1** and **leaf4**.

Version of the configlet we are pushing is very simple: we have added a new vlan to deploy to existing list of vlan:

```
vlan 12
!
vlan 34
!
vlan 73
!
```

In any case, you have to define connection information. it can be done using CLI options or by loading variables from your environment as described in options section

4.2 Use json file for bulk actions

Another way 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.

4.2.1 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.

4.2.2 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.

4.2.3 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.

4.2.4 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.

4.2.5 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.

4.2.6 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.

CHAPTER 5

Code documentation

cvpConfigletUploader.**action_add**(*configlet_def*, *parameters*)
Manage actions to ADD a configlet.

Create CVP connection and instantiate a CvpConfiglet object Then call appropriate method to start object creation If apply option is set to true, then, generated tasks are applied by CVP. Otherwise, user has to do it manually

Parameters option should at least contain following elements: - username - password - cvp (server IP or DNS hostname)

Parameters

- **configlet_def** (*dict*) – Data from JSON to describe configlet
- **parameters** (*dict*) – Object with all information to create connection

cvpConfigletUploader.**action_update**(*configlet_def*, *parameters*)
Manage actions to UPDATE and existing configlet.

Create CVP connection and instantiate a CvpConfiglet object Then call appropriate method to start object update And finally run tasks

Parameters option should at least contain following elements:

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

Parameters

- **configlet_def** (*dict*) – Data from JSON to describe configlet
- **parameters** (*dict*) – Object with all information to create connection

cvpConfigletUploader.**action_delete**(*configlet_def*, *parameters*)
Manage actions to DELTE a configlet.

Create CVP connection and instantiate a CvpConfiglet object Then call appropriate method to start object deletion

Parameters option should at least contain following elements:

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

Parameters

- **configlet_def** (*dict*) – Data from JSON to describe configlet
- **parameters** (*dict*) – Object with all information to create connection

cvpConfigletUploader.**action_create_change_control**(*parameters*, *data*)

Create a Change-Control.

Create a change-control on CVP based on a JSON definition. Current version supports following entries in JSON: - name: change-control name configured on CVP - type: change-control (Must be set with this vaue to engage CC) - country: Country required by CVP for CC - timezone: Timezone required by CVP to run changes

Expected inputs data JSON file:

```
[
    {
        "name": "Python_CC",
        "type": "change-control",
        "country": "France",
        "timezone": "Europe/Paris"
    }
]
```

Todo: Manage way to retrieve Template ID / As feature is not part of CVPRAC, `snapid` shall be part of the job definition. If not, then we configure it to `None`

Parameters

- **configlet_def** (*dict*) – Data from JSON to describe configlet
- **parameters** (*dict*) – Object with all information to create connection

5.1 Inventory Class

class cvpConfigletUploader.**CvpInventory**(*cvp_server*)

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*)

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 give 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

5.2 Configlet Class

class cvpConfigletUploader.**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 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

Note: This class use call 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

5.3 Change Control Class

class cvpConfigletUploader.**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 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

Todo:

- Implement a way to get snapshot IDs based on name

Warning:

- Change Control execution is not running snapshot before and after

__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

CHAPTER 6

Configlet uploader to CVP

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

Supported Features

- **Update** existing remote configlet.
- Execute configlet update.
- Wait for task result.
- **Delete** configlet from server.
- **Creating** a new Configlet.
- Add and remove devices to/from existing configlet.
- Creating **change-control**.
- **Scheduling** change-control.
- Collect tasks to attach to change-control.

Complete documentation available on read the doc

6.1 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
```

6.2 Getting Started

```
$ pip install git+https://github.com/titom73/configlet-cvp-uploader.git

# 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'
export CVP_TZ='France'
export CVP_COUNTRY='France'
EOT

# run script (assuming VLANs configlet is present on CVP)
$ cvp-configlet-uploader -c VLANs
```

6.3 License

Project is published under BSD License.

6.4 Ask question or report issue

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

6.5 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 7

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

C

D

G

N

O

R

U