Vent Documentation

Release 0.6.2

Cyber Reboot

Jun 29, 2018

Vent Guides

Vent is a library that includes a CLI designed to serve as a general platform for analyzing network traffic. Built with some basic functionality, Vent serves as a user-friendly platform to build custom `plugins` on to perform user-defined processing on incoming network data. Vent is filetype-agnostic in that the plugins installed within your specific vent instance determine what type of files your instance supports.

Simply create your `plugins`, point Vent to them, install them, and drop a file in Vent to begin processing!

CHAPTER 1

Dependencies

```
docker >= 1.13.1
make (if building from source)
pip3
python3.6.x
```

CHAPTER 2

Getting Set Up

There's two ways to get Vent up and running on your machine:

1. Pip:

```
$ pip3 install vent
```

2. Clone the repo:

```
$ git clone --recursive https://github.com/CyberReboot/vent.git
$ cd vent
```

3. Build from source (for sudo/root privileged users):

```
$ make
```

   Users with limited permissions or require user-local installation can use the following:

```
$ sudo env "PATH=$PATH" make
```

Note: If you already have `docker-py` installed on your machine, you may need to `pip3 uninstall docker-py` first. `vent` will install `docker-py` as part of the installation process. However, there are known incompatibilities of `docker-py` with older versions.

Once installed, it's simply:

```
$ vent
```

CHAPTER 3

Contributing to Vent

Want to contribute? Awesome! Issue a pull request or see more details here.

See this for a crash course on npyscreen: the GUI used by Vent!

3.1 Vent Quickstart Guide

Getting Vent set up is quick and easy.

1. First, use pip to install the latest *stable* version of Vent:

- Using pip:

```
pip3 install vent && vent
```

Developer versions of Vent are available but may not be entirely stable.

- Using Docker:

```
docker pull cyberreboot/vent
docker run -it vent_image_id
```

In order to avoid having to use sudo or run docker as root, adding your current user to the docker group is the recommended way to work.

- Using Git:

```
git clone https://github.com/CyberReboot/vent
cd vent && make && vent
```

2. Now that Vent has started, let's add, build, and start the core tools.

1. In the main menu, press ^X to open the action menu
2. Select Core Tools or press c
3. Select Add all latest core tools or press i. Vent will now clone the core tools' directories from CyberReboot/vent.
4. Select Build core tools from the core tools sub-menu and use the arrow keys and the Enter key to press OK. It's possible to choose which core tools are built using the Space key. Boxes with an X in them have been selected. Note that building the core tools takes a few minutes. Please be patient while Vent creates the images.
5. Once the tool images are built, go back to the core tools sub-menu from main action menu and select Start core tools and hit OK. Much like Build core tools, it's possible to select which core tools are started.

3. The core tools' containers are up and running. Next, let's add some plugins.

1. From the action menu, Select Plugins or press p.
2. Select Add new plugin or press a.
3. For this quick start guide, we will use one of the example plugins provided from CyberReboot/vent-plugins. So just hit OK on the form.
4. Press the Space key to choose the master branch and hit OK.
5. Uncheck all the boxes except for /tcpdump_hex_parser and hit OK. Depending on the plugin, add times may vary so it is not unusual for long plugin add times.

4. Now we have a plugin that can process files with the extension .pcap.

1. Now, at the Vent main menu, look for the field File Drop. This is the folder that Vent watches for new files.
2. Move or copy a .pcap file into the path. Vent will recognize this new file and start tcpdump_hex_parser. Depending on the size of the .pcap file, it could take anywhere from a few seconds to minutes. You should see the jobs running counter increase by one and, after the plugin is finished running, the completed jobs counter will increase by one.

5. Let's look at the results of the plugin using elasticsearch.

1. From the action menu, select Services Running and select Core Services.
2. Copy the address next to elasticsearch into the web browser of choice.
3. On the main page, there should be a section with pcap with the results of the plugin.

Congrats! Vent is setup and has successfully recognized the pcap file and ran a plugin that specifically deals with pcaps. You can now remove the tcpdump_hex_parser via the Plugins sub-menu and create and install your own *Custom Vent Plugins*

3.2 Custom Vent Plugins

Building custom vent plugins is an easy process.

Each custom vent plugin needs at least a Dockerfile and a vent.template. Read more about Dockerfile here.

3.2.1 Vent.template Files

Vent template files are what Vent uses to build images into something it recognizes. Listed here are sections and their various options.

Look below for examples of a `vent.template` file.

-docker

All possible options and their explanations are the same as the parameters for the python Docker container run command.

For example, if we wanted the plugin's container to use the host network stack, the following would be written in the `vent.template` file:

```
[docker]
network_mode = host
```

-gpu

Vent plugins can run on solely on GPU if specified (extra `on`). This section sets the settings for GPU processing. At the moment, **only NVIDIA GPUs are supported**.

dedicated Should the plugin be the only process running on the GPU? If it should be, set the option to `yes`. `no` by default.

device If you know the GPU device's number, you can set it here.

enabled Tell Vent to run the plugin on GPU

mem_mb The amount of RAM(in MB) a plugin requires.

-info

All metadata about the custom plugin goes under this section.

groups the group a plugin belongs to.

name the plugin's name.

-service

Appending info to an exposed port for a running service, multiple exposed ports can be done for the same service (up to 9) by incrementing the value of 1 at the end of each settings to correspond to the order of the ports exposed for that service.

uri_prefix1 Services a tool exposes that need a more specific URI at the beginning

uri_postfix1 Services a tool exposes that need a more specific URI at the end

uri_user1 Services a tool exposes that need a username

uri_pw1 Services a tool exposes that need a password

-settings

Options that specify how the plugin will run.

ext_types Whatever this option is set to is the file extension that Vent will run this plugin on. For example, `ext_types = jpg` means Vent will run this plugin if a `.jpg` file is placed in `File Drop`.

priority Set the order in which tools get started. If a tool belongs to more than one group, it's possible to specify priorities for each group. The priorities are comma separated and in the same order that the groups are specified.

process_base Files put in `File Drop` by a user or process outside of Vent will get processed. This option is set to `yes` by default.

process_from_tool Allows to specify if the tool should process files that are outputs from other tools specifically.

instances Allows you to specify how many instantiations of a tool you want running. For example, you could have two instances of rq_worker running. This would create two rq_worker containers, which would be useful for scaling larger amounts of work.

3.2.2 Example Custom Vent Plugin

Let's create a simple Vent plugin that uses `cat` to output the contents of a `.example` file. Let's create a `vent_plugin_example` directory and enter it.

First, let's create a simple `bash` script that will `cat` the contents of a file.

Listing 1: cat.sh

```
#!bin/bash
cat $1
```

Next, a `Dockerfile` is needed so let's make it.

Listing 2: Dockerfile

```
FROM ubuntu:latest
ADD cat.sh .
ENTRYPOINT ["/cat.sh"]
CMD [""]
```

Lastly, a `vent.template` file is needed so Vent knows how to use the plugin:

Listing 3: vent.template

```
[info]
name = example plugin
groups = example

[settings]
ext_types = example
```

Here's an example of this plugin using GPUs to do work:

Listing 4: vent.template

```
[info]
name = example plugin
groups = example

[settings]
ext_types = example
```

(continues on next page)

(continued from previous page)

```
[gpu]
enabled = yes
mem_mb = 1024
dedicated = yes
```

We need to add this to either a git repo or the docker hub. Let's use git. Push the `vent_plugin_example` into some repo.

Let's now add the custom plugin to Vent. From the plugins sub-menu, select `Add new plugin` and enter the fields with whatever repo `vent_plugin_example` was pushed to. After, select the branch, commit and leave `build` to `True`. Now select `example_plugin` and hit `OK`. Vent will now build the custom plugin.

To test, let's create a test file.

Listing 5: test.example

```
qwerty
```

Finally, with Vent and the plugin up and running and all core tools added, built, and running, let's drop `test.example` into `File Drop`. After a few seconds, the job counter on the main menu of Vent will show that one job is running, and it'll finish soon after and show one completed job.

To check that the plugin worked and outputted `qwerty`, let's check the syslog container using the command `docker logs cyberreboot-vent-syslog-master | grep qwerty`.

If you see this line, congrats! You have successfully built your first Vent plugin.

If the plugin did not function correctly, try rereading the tutorial or check the *Troubleshooting* guide.

Other examples of custom plugins can be found at CyberReboot/vent-plugins.

3.3 Custom Configurations of Plugins

Say your custom plugin, called `example_plugin`, has a `example_plugin.config` file that it needs filled out before the plugin is built into an image.

Listing 6: example_plugin.config

```
[Some_Section]
example_option1 = FILL IN
example_option2 = FILL IN
example_option3 = FILL IN
```

Vent can now fill in those values for you. In your home directory, have a file called `.plugin_config.yml`.

We'll fill in `.plugin_config.yml` with the necessary fields:

Listing 7: .plugin.config.yml

```
example_plugin
    Some_Section:
        example_option1 = 1
        example_option2 = 2
        example_option3 = 3
```

Vent will assume that, in the directory example_plugin, there exists another directory titled config that has the actual config file. The path of our example would be example_plugin/config/example_plugin.config.

Using this .plugin_config.yml, you can fill in multiple tools' config files at the same time.

If you make a change to your config files and want to rebuild the images using these new settings, **clean** and then **build** the plugin tools in the plugins submenu. Vent will fill in the config files with the new values.

3.4 Vent Initialization

This page describes the different actions that vent takes upon initialization.

3.4.1 File Structure

Vent will set up the necessary file structure to store all of the *User Data*

3.4.2 Auto-Install

Vent will detect if there are any vent images already running before an instance of itself is instantiated, and will automatically install the tools that correspond to that image without you having to do anything.

3.4.3 Startup

Before initialization you can specify a certain set of tools that you want vent to install once it starts up for the first time. You can do this by writing a .vent_startup.yml file in your home directory. If you do not know how YAML syntax works, watch this quick video for more info. The syntax that vent uses when parsing the .vent_startup.yml for tools is as follows:

```
repo_name:
  tool_name:
    <additional settings you want>
```

A thing to note, tool_name here means the name of the directory in which the Dockerfile and vent.template for the tool can be found. If you have multiple tools defined in the same directory, then use the syntax @tool_name to differentiate between the different tools in that directory. A lot of the additional settings you can configure are the same as what you would do for *Custom Vent Plugins*. These additional settings will be used to update the default settings for that tool (as defined in its vent.template), they will not overwrite the default settings. Besides those settings, there are some that are specific just to this startup file:

build If you set build to yes, then the tool will be installed and then built immediately rather than just installed.

start If you set start to yes, then the tool will be started immediately. **Note**, you have to set build to yes for a tool in order for you to be able to set start to yes as well.

branch For tools installed from a repo, if you want to install the tools from a specific branch of the repo, specify it with the branch keyword. By default, the master branch will be where the tools are installed from.

version For tools installed from a repo, if you want to install the tools from a specific commit version, specify it with the version keyword. By default, the version HEAD will be used to install tools.

Example Startup

With that said, here is an example .vent-startup.yml file that vent could use:

```
https://github.com/cyberreboot/vent:
  rabbitmq:
    build: yes
    start: yes
  elasticsearch:
  rq_worker:
    build: yes
    start: yes
    settings:
      instances: 2
```

This would install the tools rabbitmq, elasticsearch, and rq_worker. Additionally, this would build and start rabbitmq and rq_worker, and it would set the number of instances in settings to 2 for rq_worker. Notice how with elasticsearch even though there were no additional parameters defined it still had the the colon at the end of its declaration. This is necessary otherwise a parsing error will occur with YAML.

3.5 Troubleshooting

3.5.1 Basic Troubleshooting

Something not working as expected within Vent? Not a problem! Let's first get some basic possible errors out of the way.

1. Is Docker installed and is the daemon running? Vent uses Docker heavily so it is necessary to have the Docker installed and the daemon running.
2. Is this the latest version of Vent? Ways to get the latest Vent version:
   - Using pip:

     ```
     pip3 install vent && vent
     ```

Pip installs the latest *stable* release of Vent. If a problem still persists, try using the latest developer build:

- Using Docker:

  ```
  docker pull cyberreboot/vent
  docker run -it vent_image_id
  ```

- Using Git:

  ```
  git clone https://github.com/CyberReboot/vent
  cd vent && make && vent
  ```

3.5.2 In-Depth Troubleshooting

Still not working? That's fine! Let's get into the nitty gritty and try to figure out what went wrong.

Is it Vent that's causing the problem?

Firstly, let's see if it's Vent that's causing the problems. Go to the `User Data` folder and open up `vent.log` with your favorite text editor. Let's search the key term `False`. Iterate through the search results and look for `Status of some_function:  False`. This tells us that one of Vent's core functions is not performing as expected. Next to it, there will be an error message explaining what went wrong.

If there's a problem with Vent's implementation, please create an issue here.

Is it a custom plugin that's causing the problem?

If there's no obvious error messages within `vent.log`, let's check any added plugin tools and their containers.

Run the command `docker logs cyberreboot-vent-syslog-master`. This will return all information about all plugin containers and any information regarding the error should be displayed here.

3.6 Contributing to Vent

Want to hack on Vent? Awesome! Here are instructions to get you started. They are probably not perfect, please let us know if anything feels wrong or incomplete.

3.6.1 Contribution guidelines

Pull requests are always welcome

We are always thrilled to receive pull requests, and do our best to process them as fast as possible. Not sure if that typo is worth a pull request? Do it! We will appreciate it.

If your pull request is not accepted on the first try, don't be discouraged! If there's a problem with the implementation, hopefully you received feedback on what to improve.

We're trying very hard to keep Vent lean and focused. We don't want it to do everything for everybody. This means that we might decide against incorporating a new feature. However, there might be a way to implement that feature *on top of* Vent.

Create issues...

Any significant improvement should be documented as a github issue before anybody starts working on it.

...but check for existing issues first!

Please take a moment to check that an issue doesn't already exist documenting your bug report or improvement proposal. If it does, it never hurts to add a quick "+1" or "I have this problem too". This will help prioritize the most common problems and requests.

Conventions

Fork the repo and make changes on your fork in a feature branch.

Make sure you include relevant updates or additions to documentation and tests when creating or modifying features.

Pull requests descriptions should be as clear as possible and include a reference to all the issues that they address.

Code review comments may be added to your pull request. Discuss, then make the suggested modifications and push additional commits to your feature branch. Be sure to post a comment after pushing. The new commits will show up in the pull request automatically, but the reviewers will not be notified unless you comment.

Before the pull request is merged, make sure that you squash your commits into logical units of work using `git rebase -i` and `git push -f`. After every commit the test suite should be passing. Include documentation changes in the same commit so that a revert would remove all traces of the feature or fix.

Commits that fix or close an issue should include a reference like `Closes #XXX` or `Fixes #XXX`, which will automatically close the issue when merged.

Add your name to the `AUTHORS` file, but make sure the list is sorted and your name and email address match your git configuration. The `AUTHORS` file is regenerated occasionally from the git commit history, so a mismatch may result in your changes being overwritten.

3.6.2 Decision process

How are decisions made?

Short answer: with pull requests to the Vent repository.

All decisions affecting Vent, big and small, follow the same 3 steps:

- Step 1: Open a pull request. Anyone can do this.
- Step 2: Discuss the pull request. Anyone can do this.
- Step 3: Accept or refuse a pull request. A maintainer does this.

How can I become a maintainer?

- Step 1: learn the code inside out
- Step 2: make yourself useful by contributing code, bug fixes, support etc.

Don't forget: being a maintainer is a time investment. Make sure you will have time to make yourself available. You don't have to be a maintainer to make a difference on the project!

What are a maintainer's responsibility?

It is every maintainer's responsibility to:

- 1. Deliver prompt feedback and decisions on pull requests.
- 2. Be available to anyone with questions, bug reports, criticism, etc. regarding Vent.

How is this process changed?

Just like everything else: by making a pull request :)

Derivative work from Docker

3.7 New Vent Releases

Get the latest clone of Vent from https://github.com/CyberReboot/vent

1. Change the version number
    - setup.py
    - docs/source/conf.py
2. Edit CHANGELOG.md and include a list of changes that were made. Please follow previous formatting.
3. Run list of authors and put in *AUTHORS* to make sure it is up to date:

    ```
    git log --format='%aN <%aE>' | sort -f | uniq
    ```

4. Commit the changes, open a PR, and merge into master.
5. Now let's upload the release to pypi assuming there's an account with admin access and a corresponding .pypirc:

    ```
    python3 setup.py sdist upload
    ```

6. Create a new github release. Tag and release title are the version number.
7. Finally, change the version number to the next version number with a dev tag. Eg: 0.4.4.dev. Commit the version change, make a PR, and merge to master.

3.8 Core Tools and Plugins

3.8.1 Core Tool Explanations

There are currently nine core tools that Vent uses.

elasticsearch

Enables comprehensive text search of syslog.

file_drop

Watches the specified directory for any new files. If a new file is added, it is added to a redis queue.

network_tap

A container that will watch a specific nic using tcpdump to output pcap files based on what was monitored. Has an interface located in System Commands -> Network Tap Interface in the main action menu. The interface has six available actions:

- **Create**: Create a new container with a specified nic, tag, interval (in *seconds*), filter, and iterations. The container is also automatically started on creation.
- **Delete**: Delete a specified network tap container. Containers *must be stopped* before they are able to be deleted.
- **List**: Show all network tap containers. Will return container's ID, if the container is running or not, and the tag provided in create.

- **NICs**: Show all available network interfaces. Will return a list of the names of the available NICs. Note for `Docker for Mac` it will show available network interfaces on the VM running the Docker daemon, not the network interface names on the Mac host.
- **Start**: Start a network tap container if it is exited. Will run with the same options given to the container in `create`.
- **Stop**: Stop a network tap container.
- **Update**: Update the metadata of a network tap container.

rabbitmq

Formats messages received from syslog and sends them to rmq_es_connector.

redis

A key/value store that is used for the queuing system that `file drop` sends to and `rq_worker` pulls out of.

rmq_es_connector

A gateway between the messaging system and `elasticsearch`. This way, the message formatting system is not locked to `rabbitmq`.

rq_worker

The tool that takes files from the `redis` queue and runs plugins that deal with those file extensions.

rq_dashboard

Management console to look at rq_worker's active queue.

syslog

Standard logging system that adheres to the syslog standard. All tool containers send their information to syslog. If there's some unexpected outputs or a container isn't running properly, all information will be in this tool's container.

Access this tool's container with the command: `docker logs cyberreboot-vent-syslog-master`

3.8.2 Core Tool and Plugin Actions

Short explanations of all actions available in the core tools and plugins sub-menu.

Add all latest core/plugin tools

Clone the latest version of the tool. This will **not** update or remove any tools that have already been added.

By default, core tools are cloned from CyberReboot/vent and plugins, if no custom repo is specified, are cloned from CyberReboot/vent-plugins.

Build core/plugin tools

Build docker images from the Dockerfiles obtained from adding.

Clean core/plugin tools

Stop and remove the chosen tools' containers.

Configure core/plugin tools

Edit a tool's vent.template file found in the tool's respective folder. Read more about *Vent.template Files*.

Disable core/plugin tools

Remove chosen tools from menus. For example, let's say there were ten tools but only five were needed. Disabling the five unneeded tools would stop those tools from appearing on the other menus.

Enable core/plugin tools

Opposite of disable tools. Enables the tools so they can be seen again.

Inventory of core/plugin tools

Provides meta data regarding currently added core/plugin tools. It tells if a tool is built, enabled, the name of the image, and the if the tool is currently running.

Remove core/plugin tools

Remove a tool entirely. Any of that tool's containers are also stopped and deleted. The tool must be added again if it is to be used.

Start core/plugin tools

Start the tools' respective containers.

Stop core/plugin tools

Stop the tools' respective containers.

Update core/plugin tools

Pulls the latest commit of the tool from its repo and builds it.

3.9 System Commands

This page describes what the different actions in the `System Commands` sub-menu do.

3.9.1 Backup

Saves the `vent.cfg` and the `plugin_manifest`.

3.9.2 Change vent configuration

Edit the `vent.cfg` in Vent. The `vent.cfg` file can be found in the directory listed next to `User Data` in the main Vent menu.

3.9.3 Detect GPUs

Detect GPU devices on the current machine running Vent. Currently works **with NVIDIA GPUs only** .

3.9.4 Enable Swarm Mode

To Be Implemented.

3.9.5 Factory reset

Remove all Vent related images and containers. Removes the `User Data` directory as well. Please back up any necessary information before activating this option.

3.9.6 Restore

Restores a backup.

3.9.7 Upgrade

To Be Implemented

3.9.8 Network Tap Interface

The interface in which to interact with the `Network Tap` core tool. Read more about *network_tap*. Each form will also have more information regarding the specific action.

3.10 User Data

The User Data directory is listed on the Vent main menu. Here you can find five files and two directories here.

3.10.1 Files

plugin_manifest.cfg

Meta data about all built core and plugin tools.

status.json

All data regarding finished jobs is written to this JSON file. This file actually doesn't exist until a job is processed and **completed**. Each tool will get it's own JSON entry. If there were two tools that ran and finished, there will be one entry for each tool.

vent.cfg

A configuration file used to customize Vent and its processes.

-main

files The directory of `File Drop`. For example, if `files = /opt/vent_files`, Vent will monitor for any new files in that directory.

services_uri Override the default POST name for all services. So, let's say Vent has `Elasticsearch` at URL `0.0.0.0:3772`.

By adding:

```
services_uri = 5.5.5.5
```

under the main section, the URL for `Elasticsearch` is now `5.5.5.5:3772`.

-network-mapping

nic_name The option here can be whatever is desired. The value is the nic that the plugin tool `replay_pcap` will replay to.

For example:

```
my_nic = enp0s25
```

Now the plugin tool `replay_pcap` will mirror whatever given pcap files to `enp0s25`.

Currently, only one nic at a time is supported.

-nvidia-docker-plugin

port Override port for the Nvidia docker plugin

host Override the host for the Nvidia docker plugin

-external-services

Name of tool we want Vent to use externally rather than internally. For example, if there is already an `Elasticsearch` service that a user wants Vent to use rather than Vent's internal `Elasticsearch`, we could write:

```
[external-services]
Elasticsearch = {"ip_address": "172.10.5.3", "port": "9200",
"locally_active": "yes"}
```

If a port is not specified, then it defaults to core tool's default port. Also, you can toggle whether you want to use the local docker container that vent utilizes or the external service by switching locally_active between yes and no.

-build-options

use_existing_images If set to `yes` it will not try to rebuild or pull images that already exist.

-groups

start_order The order in which the tool groups are started in csv format. For example:

```
core,pcap,replay
```

means that core tools are started first, then any tool in the `pcap` group, and lastly any tool in the `replay` group. Within those groups, plugin priority will determine the start order of the tools.

vent.init

This file only tells Vent if this is the first time Vent is running so the tutorial can be the first menu listed rather than Vent's main menu.

vent.log

Log file that keeps track of the statuses of Vent's functionality. It lists what action is taking place and shows a list of steps taken to perform that action. So if something goes wrong, this file is a good place to start to find out what the problem is.

3.10.2 Directories

.internals

This is a folder that Vent clones to when dealing with core tools. So when Vent adds or builds core tools, this is the working directory.

plugins

This folder deals with any plugin tools with the same philosophy as the `.internals` directory.

3.11 vent.api package

3.11.1 Submodules

3.11.2 vent.api.actions module

3.11.3 vent.api.menu_helpers module

3.11.4 vent.api.plugin_helpers module

3.11.5 vent.api.plugins module

3.11.6 vent.api.templates module

3.11.7 Module contents

3.12 vent.core package

3.12.1 Subpackages

vent.core.file_drop package

Submodules

vent.core.file_drop.file_drop module

class `vent.core.file_drop.file_drop.`**`GZHandler`**(*patterns=None*, *ignore_patterns=None*, *ignore_directories=False*, *case_sensitive=False*)

Bases: `watchdog.events.PatternMatchingEventHandler`

Handles when an event on the directory being watched happens that matches the values in patterns

`created_files = set([])`

`ignore_patters = ['*-miscellaneous*']`

`on_created`(*event*)

Called when a file or directory is created.

Parameters **`event`** (`DirCreatedEvent` or `FileCreatedEvent`) – Event representing file/directory creation.

`on_modified`(*event*)

Called when a file or directory is modified.

Parameters **`event`** (`DirModifiedEvent` or `FileModifiedEvent`) – Event representing file/directory modification.

patterns = ['*']

process (*event*)

event.event_type 'modified' | 'created' | 'moved' | 'deleted'

event.is_directory True | False

event.src_path path/to/observed/file

q = Queue(u'default')

r = StrictRedis<ConnectionPool<Connection<host=redis,port=6379,db=0>>>

Module contents

vent.core.network_tap package

Subpackages

vent.core.network_tap.ncontrol package

Submodules

vent.core.network_tap.ncontrol.ncontrol module

vent.core.network_tap.ncontrol.paths module

class vent.core.network_tap.ncontrol.paths.**CreateR**

Bases: object

This endpoint is for creating a new filter

on_post (*req*, *resp*)

Send a POST request with id/nic/interval/filter/iters and it will start a container for collection with those specifications

class vent.core.network_tap.ncontrol.paths.**DeleteR**

Bases: object

This endpoint is for deleting a network tap filter container

on_post (*req*, *resp*)

Send a POST request with a docker container ID and it will be deleted.

Example input: {'id': "12345"}, {'id': ["123", "456"]}

class vent.core.network_tap.ncontrol.paths.**InfoR**

Bases: object

This endpoint is for returning info about this service

on_get (*req*, *resp*)

class vent.core.network_tap.ncontrol.paths.**ListR**

Bases: object

This endpoint is for listing all filter containers

on_get (*req*, *resp*)
Send a GET request to get the list of all of the filter containers

class vent.core.network_tap.ncontrol.paths.**NICsR**
Bases: object

This endpoint is for listing all available network interfaces

on_get (*req*, *resp*)
Send a GET request to get the list of all available network interfaces

class vent.core.network_tap.ncontrol.paths.**StartR**
Bases: object

This endpoint is for starting a network tap filter container

on_post (*req*, *resp*)
Send a POST request with a docker container ID and it will be started.

Example input: {‘id’: “12345”}, {‘id’: [“123”, “456”]}

class vent.core.network_tap.ncontrol.paths.**StopR**
Bases: object

This endpoint is for stopping a network tap filter container

on_post (*req*, *resp*)
Send a POST request with a docker container ID and it will be stopped.

Example input: {‘id’: “12345”}, {‘id’: [“123”, “456”]

class vent.core.network_tap.ncontrol.paths.**UpdateR**
Bases: object

This endpoint is for updating a filter

on_post (*req*, *resp*)
Send a POST request with id and metadata and it will update the existing filter metadata with those specifications

vent.core.network_tap.ncontrol.prestart module

vent.core.network_tap.ncontrol.routes module

vent.core.network_tap.ncontrol.routes.**paths**()

vent.core.network_tap.ncontrol.routes.**routes**()

Module contents

Module contents

vent.core.rmq_es_connector package

Submodules

vent.core.rmq_es_connector.rmq_es_connector module

class vent.core.rmq_es_connector.rmq_es_connector.**RmqEs**(*es_host='elasticsearch'*, *rmq_host='rabbitmq'*)

opens a connection to rabbitmq and receives messages based on the provided binding keys and then takes those messages and sends them to an elasticsearch index

callback(*ch*, *method*, *properties*, *body*)

callback triggered on rabiitmq message received and sends it to an elasticsearch index

channel = None

connections(*wait*)

wait for connections to both rabbitmq and elasticsearch to be made before binding a routing key to a channel and sending messages to elasticsearch

consume()

start consuming rabbitmq messages

es_conn = None

es_host = None

es_port = 9200

queue_name = None

rmq_host = None

rmq_port = 5672

start()

start the channel listener and start consuming messages

vent.core.rmq_es_connector.test_rmq_es_connector module

class vent.core.rmq_es_connector.test_rmq_es_connector.**Method**(*routing_key='foo.bar'*)

create mock method object

routing_key = None

vent.core.rmq_es_connector.test_rmq_es_connector.**test_rmq_es_connector_callback**()

tests the callback function

vent.core.rmq_es_connector.test_rmq_es_connector.**test_rmq_es_connector_connections**()

tests the connections function

vent.core.rmq_es_connector.test_rmq_es_connector.**test_rmq_es_connector_start**()

tests the start function

Module contents

vent.core.rq_dashboard package

Module contents

vent.core.rq_worker package

Submodules

vent.core.rq_worker.test_watch module

vent.core.rq_worker.watch module

`vent.core.rq_worker.watch.`**`file_queue`**`(`*path*, *template_path='/vent/'*, *r_host='redis'*`)`

Processes files that have been added from the rq-worker, starts plugins that match the mime type for the new file.

`vent.core.rq_worker.watch.`**`gpu_queue`**`(`*options*`)`

Queued up containers waiting for GPU resources

Module contents

3.12.2 Module contents

3.13 vent.helpers package

3.13.1 Submodules

3.13.2 vent.helpers.errors module

`vent.helpers.errors.`**`ErrorHandler`**`(`*function*`)`

3.13.3 vent.helpers.logs module

3.13.4 vent.helpers.meta module

3.13.5 vent.helpers.paths module

3.13.6 Module contents

3.14 vent.menus package

3.14.1 Submodules

3.14.2 vent.menus.add module

3.14.3 vent.menus.add_options module

3.14.4 vent.menus.backup module

class `vent.menus.backup.`**`BackupForm`**(**args*, ***keywords*)

Bases: `npyscreen.fmActionForm.ActionForm`

Form that can be used to select a backup file to restore

`create`()

Add backup files to select from

`on_cancel`()

When user clicks cancel, will return to MAIN

`on_ok`()

Perform restoration on the backup file selected

`quit`(**args*, ***kwargs*)

3.14.5 vent.menus.choose_tools module

3.14.6 vent.menus.del_instances module

3.14.7 vent.menus.editor module

3.14.8 vent.menus.help module

class `vent.menus.help.`**`HelpForm`**(**args*, ***keywords*)

Bases: `npyscreen.fmFormWithMenus.ActionFormWithMenus`

Help form for the Vent CLI

`change_forms`(**args*, ***keywords*)

Checks which form is currently displayed and toggles to the other one

`create`()

Override method for creating FormBaseNew form

`exit`(**args*, ***keywords*)

on_cancel()

on_ok()

static switch(*page*)

3.14.9 vent.menus.inventory module

3.14.10 vent.menus.inventory_forms module

3.14.11 vent.menus.logs module

3.14.12 vent.menus.main module

3.14.13 vent.menus.ntap module

3.14.14 vent.menus.services module

3.14.15 vent.menus.tools module

3.14.16 vent.menus.tutorial_forms module

class vent.menus.tutorial_forms.**TutorialAddingFilesForm**(**args*, ***keywords*)
Bases: *vent.menus.tutorials.TutorialForm*

Tutorial Adding Files form for the Vent CLI

class vent.menus.tutorial_forms.**TutorialAddingPluginsForm**(**args*, ***keywords*)
Bases: *vent.menus.tutorials.TutorialForm*

Tutorial Adding Plugins form for the Vent CLI

class vent.menus.tutorial_forms.**TutorialBackgroundForm**(**args*, ***keywords*)
Bases: *vent.menus.tutorials.TutorialForm*

Tutorial Background form for the Vent CLI

class vent.menus.tutorial_forms.**TutorialBuildingCoresForm**(**args*, ***keywords*)
Bases: *vent.menus.tutorials.TutorialForm*

Tutorial Building Cores form for the Vent CLI

class vent.menus.tutorial_forms.**TutorialGettingSetupForm**(**args*, ***keywords*)
Bases: *vent.menus.tutorials.TutorialForm*

Tutorial Getting Setup form for the Vent CLI

class vent.menus.tutorial_forms.**TutorialIntroForm**(**args*, ***keywords*)
Bases: *vent.menus.tutorials.TutorialForm*

Tutorial introduction landing form for the Vent CLI

class vent.menus.tutorial_forms.**TutorialStartingCoresForm**(**args*, ***keywords*)
Bases: *vent.menus.tutorials.TutorialForm*

Tutorial Starting Cores form for the Vent CLI

class vent.menus.tutorial_forms.**TutorialTerminologyForm**(**args*, ***keywords*)
Bases: *vent.menus.tutorials.TutorialForm*

Tutorial terminology form for the Vent CLI

class vent.menus.tutorial_forms.**TutorialTroubleshootingForm**(**args*, ***keywords*)
Bases: *vent.menus.tutorials.TutorialForm*

Tutorial troubleshooting form for the Vent CLI

3.14.17 vent.menus.tutorials module

class vent.menus.tutorials.**TutorialForm**(*title=”*, *text=”*, *next_tutorial=”*, **args*, ***keywords*)
Bases: npyscreen.fmFormWithMenus.ActionFormWithMenus

Tutorial form for the Vent CLI

create()
Overridden to add handlers and content

on_cancel()
When user clicks cancel, will return to MAIN

on_ok()
When user clicks ok, will proceed to next tutorial

quit(**args*, ***kwargs*)
Overridden to switch back to MAIN form

switch(*name*)
Wrapper that switches to provided form

3.14.18 Module contents

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index