Cabrita Documentation

Release 2.1.0

Chris Maillefaud

Jan 10, 2019

Contents:

CHAPTER 1

Cabrita Tutorial

maxdepth 2 :caption: Modules:

1.1 In-depth Customization

To customize cabrita you can create a yaml file, let's called `cabrita.yml` to create or config _boxes_. You can select which docker containers will show in each box and what info these boxes will show for each service inside them.

To understand his basic structure, copy and paste this yaml and save the `cabrita.yml` in the same directory where your *docker-compose.yml* is located:

```
version: 2
title: My Docker Project
background_color: grey # options: black, blue, cyan, grey, yellow, white
compose_files:
  - ./docker-compose.yml
boxes:
  main_box:
    main: true
    name: My Services
```

This file will create a dashboard called *My Docker Project*, which will read all services from the `docker-compose.yml` file and add all of them to the box called *My Services*.

This is the main box (`main: true`), which means all docker services which are not included in any other box, will be added here.

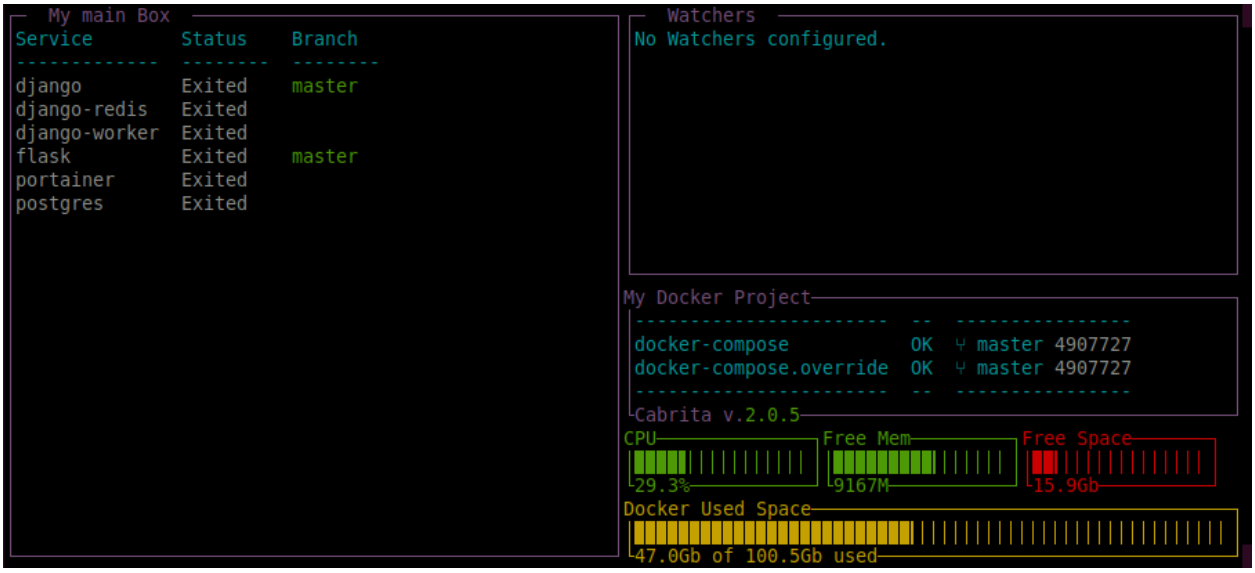
To use this file, use the `--path` option for cabrita command line. You can also define the `CABRITA_PATH` environment variable for this path.

Using the our docker-compose in `/examples` folder:

```
# You can call directly the app passing the yaml path
$ cd path/to/examples
$ TEST_PROJECT_PATH=$(pwd) docker-compose up -d
$ cabrita --path cabrita.yml

# Or you can use the CABRITA_PATH environment variable
$ export CABRITA_PATH=/path/to/yml
$ cabrita
```

The new dashboard will show:

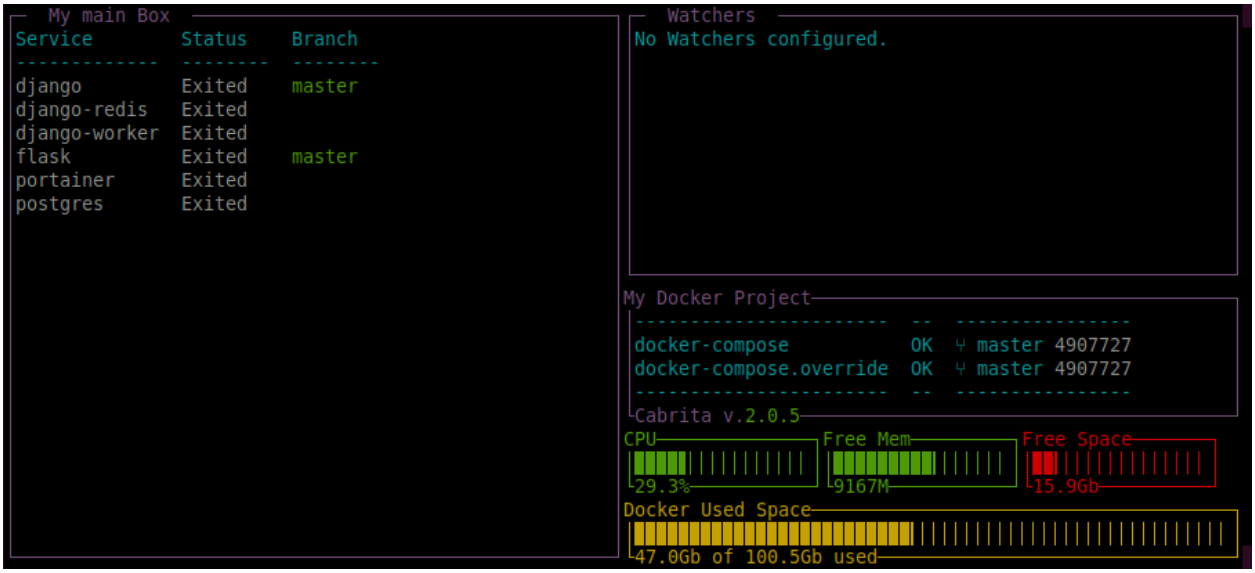


1.1.1 Customize Ports

You can define, for each box, if the docker ports will be show in dashboard. Let's add two new options: `port_view` and `port_detail` on `cabrita.yml`:

```
version: 2
title: My Docker Project
background_color: grey # options: black, blue, cyan, grey, yellow, white
compose_files:
  - ./docker-compose.yml
boxes:
  main_box:
    main: true
    name: My Services
    port_view: status # options: column, name, status
    port_detail: internal # options: internal, external or both
```

The new dashboard will show:



1.1.2 Customize Git info

For more git information, let's add two new options: `show_revision` and `watch_branch`:

```
version: 2
title: My Docker Project
background_color: grey # options: black, blue, cyan, grey, yellow, white
compose_files:
  - ./docker-compose.yml
boxes:
  main_box:
    main: true
    name: My Services
    port_view: status # options: column, name, status
    port_detail: internal # options: internal, external or both
    show_revision: true # will show commit hash and git tag if available
```

The "**Branch**" column are displayed in each box, by default (you can disable it with the `show_git: false` option). This column shows the actual branch name for each service in box. If the branch is dirty (i.e. has non-committed modifications), text color will be yellow.

The `show_revision` option show, for each service in box, the commit hash and git tag, if available.

1.1.3 The watch branch

Frequently, we need to know if the *local code* we are running inside the containers are not up-to-date with the last modifications in the cloud. To resolve this, in the yaml file we can use the `watch_branch` option:

```
version: 2
title: My Docker Project
background_color: grey # options: black, blue, cyan, grey, yellow, white
compose_files:
  - ./docker-compose.yml
boxes:
  main_box:
```

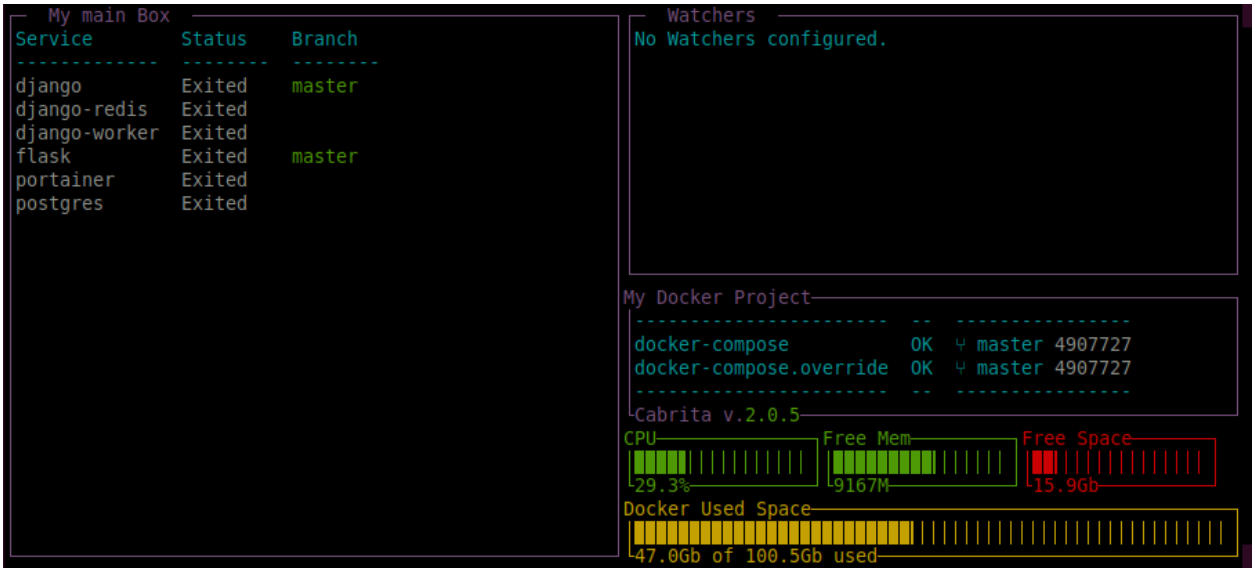
(continues on next page)

(continued from previous page)

```
    main: true
    name: My Services
    port_view: status # options: column, name, status
    port_detail: internal # options: internal, external or both
    show_revision: true # will show commit hash and git tag if available
    watch_branch: origin/staging # check how ahead or behind you are regard this␣
↪branch
```

The `watch_branch` will add on "Branch" column how many commits ahead or behind the current branch are in comparison of the *watched* branch.

The new dashboard will show:



1.1.4 The NEED BUILD status

Other thing we frequently need to know if the *docker image* we are running is up-to-date with the last code modifications. If you're using **'docker volumes'_** docker will automatically run the latest code inside containers. But how about modifications in `Dockerfile`, `requirements.txt`, `package.json` and such files?

To resolve this, lets use two new options, called `watch_for_build_using_files` and `watch_for_build_using_git`:

```
version: 2
title: My Docker Project
background_color: grey # options: black, blue, cyan, grey, yellow, white
compose_files:
  - ./docker-compose.yml
boxes:
  main_box:
    main: true
    name: My Services
    port_view: status # options: column, name, status
    port_detail: internal # options: internal, external or both
    show_revision: true # will show commit hash and git tag if available
    watch_branch: origin/staging # check how ahead or behind you are regard this␣
↪branch
```

(continues on next page)

(continued from previous page)

```
  watch_for_build_using_files: # check if Dockerfile last modification date is␣
↪greater than docker image build date
    - Dockerfile
  watch_for_build_using_git: # check if last commit date in flask project is greater␣
↪than docker image build date
    - flask
```

The first option, `watch_for_build_using_files`, means if any file inside list is located in the code folder, and if his last modification date is more recent than his docker image, that image needs to be rebuild.

The second option, `watch_for_build_using_git`, means if the date for the last commit is more recent than his docker image, that image needs to be reduild.

Use the first, when you want to track for new builds using a bunch of files. Use the last, if any modification in project needs to start another build.

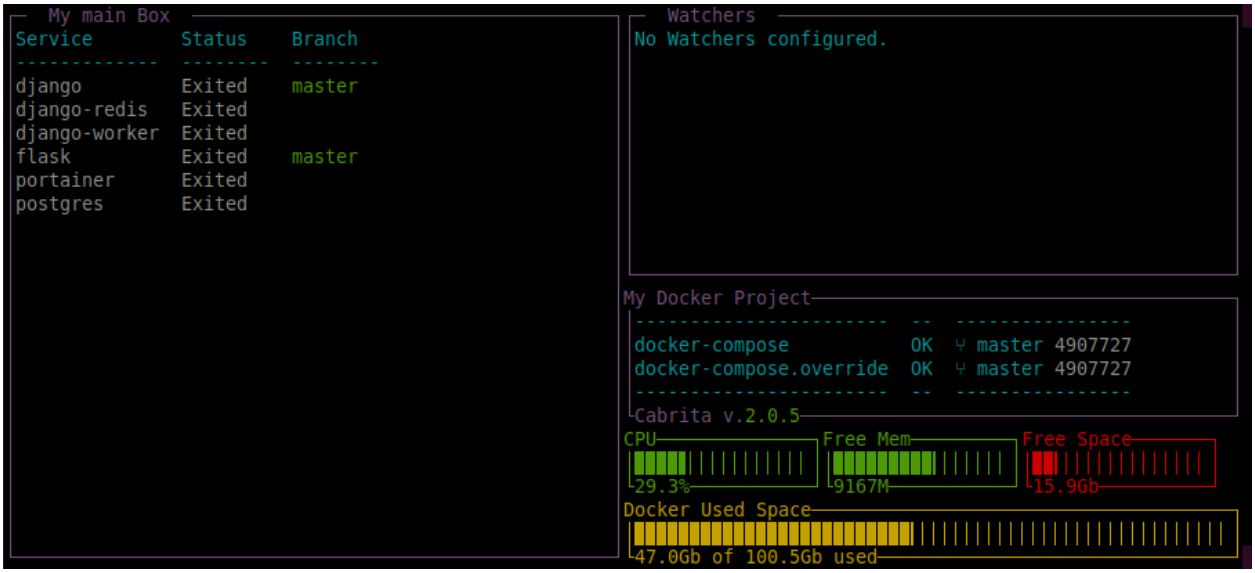
1.1.5 Adding new boxes

Let's add the `docker-compose.override.yml` and create a new box, only for django applications:

```
version: 2
title: My Docker Project
background_color: grey # options: black, blue, cyan, grey, yellow, white
compose_files:
  - ./docker-compose.yml
boxes:
  main_box:
    main: true
    name: My Services
    port_view: status # options: column, name, status
    port_detail: internal # options: internal, external or both
    show_revision: true # will show commit hash and git tag if available
    watch_branch: origin/staging # check how ahead or behind you are regard this␣
↪branch
  watch_for_build_using_files: # check if Dockerfile last modification date is␣
↪greater than docker image build date
    - Dockerfile
  watch_for_build_using_git: # check if last commit date in flask project is greater␣
↪than docker image build date
    - flask
  django:
    name: Django Apps
    show_git: false
    includes:
      - django
```

This file contains a new box, called *django*. The *includes* option is used to create a list of services which will include on this box. This option is mutually exclusive with *main* option, because every service not included in *include* for all boxes will be display in the main box.

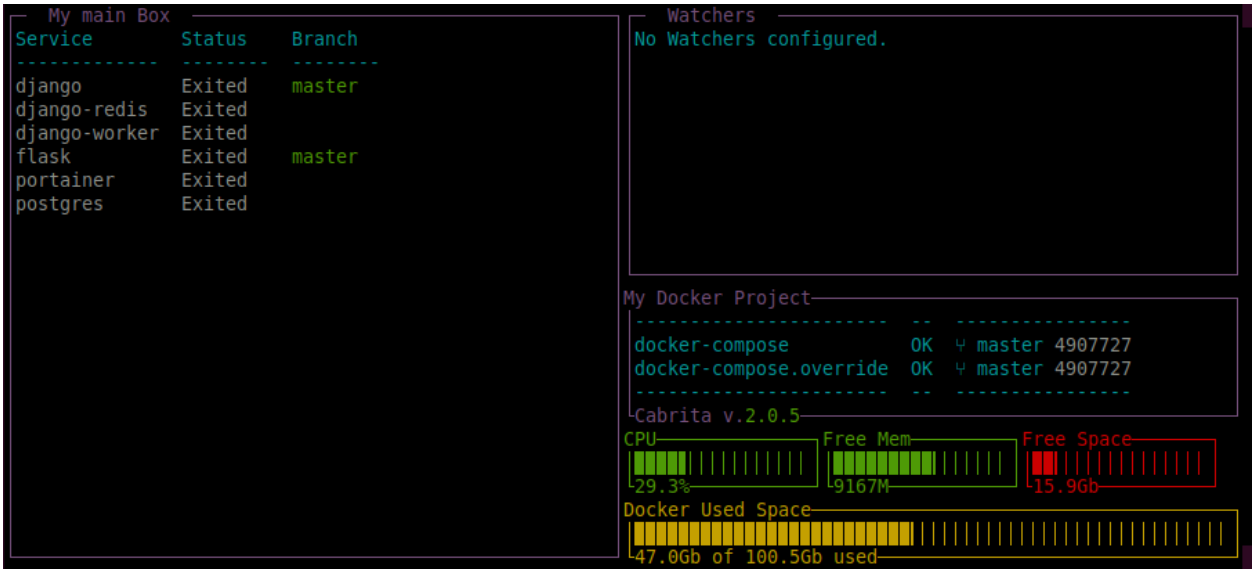
The new dashboard will show:



The "Django Apps" box will show 3 services. But what if we understand these docker containers are just one big application in docker? Because they all got the same name - "django" - we can categorize all services in the same line, using the `categories` option:

```
version: 2
title: My Docker Project
background_color: grey # options: black, blue, cyan, grey, yellow, white
compose_files:
  - ./docker-compose.yml
boxes:
  main_box:
    main: true
    name: My Services
    port_view: status # options: column, name, status
    port_detail: internal # options: internal, external or both
    show_revision: true # will show commit hash and git tag if available
    watch_branch: origin/staging # check how ahead or behind you are regard this␣
↪branch
  watch_for_build_using_files: # check if Dockerfile last modification date is␣
↪greater than docker image build date
    - Dockerfile
  watch_for_build_using_git: # check if last commit date in flask project is greater␣
↪than docker image build date
    - flask
  django:
    name: Django Apps
    show_git: false
    includes:
      - django
    categories:
      - worker
      - redis
```

The new dashboard will show:



1.1.6 Adding watchers

Watchers are customized file and ping checkers for your project. Let's add a new watcher, to check internet connection:

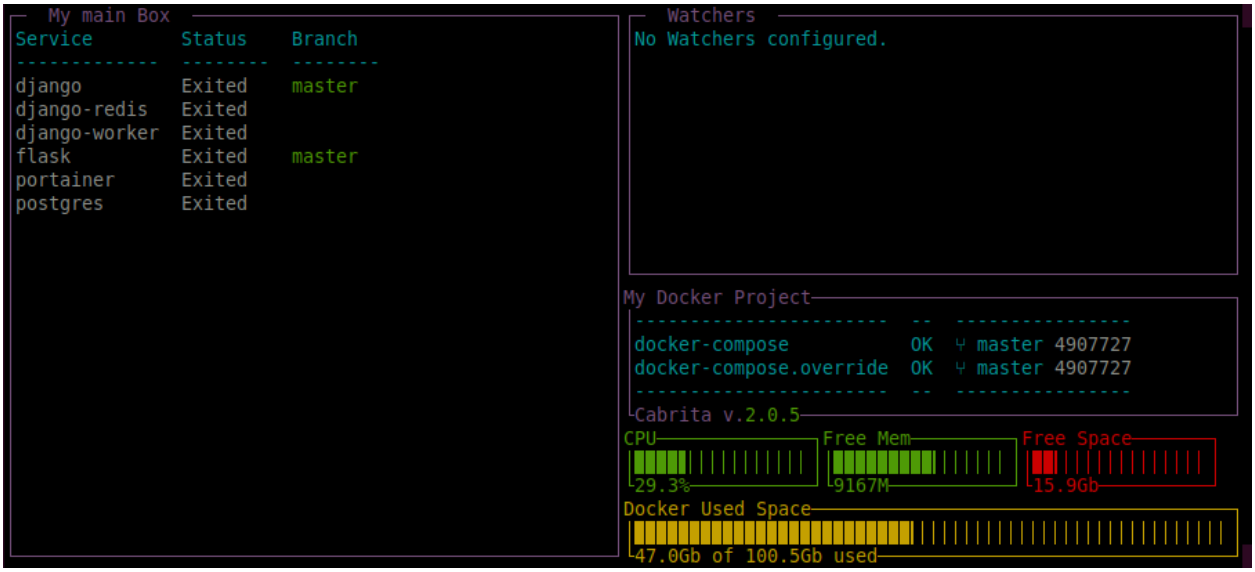
```
version: 2
title: My Docker Project
background_color: grey # options: black, blue, cyan, grey, yellow, white
compose_files:
  - ./docker-compose.yml
boxes:
  main_box:
    main: true
    name: My Services
    port_view: status # options: column, name, status
    port_detail: internal # options: internal, external or both
    show_revision: true # will show commit hash and git tag if available
    watch_branch: origin/staging # check how ahead or behind you are regard this␣
↪branch
  watch_for_build_using_files: # check if Dockerfile last modification date is␣
↪greater than docker image build date
    - Dockerfile
  watch_for_build_using_git: # check if last commit date in flask project is greater␣
↪than docker image build date
    - flask
  django:
    name: Django Apps
    show_git: false
    includes:
      - django
    categories:
      - worker
      - redis
watchers:
  ping:
    google:
      name: Check internet connectivity
```

(continues on next page)

(continued from previous page)

```
      address: https://www.google.com
      message_on_success: UP
      message_on_error: DOWN
```

The new dashboard will show:



CHAPTER 2

Cabrita File Reference

CHAPTER 3

Cabrita API Reference

3.1 cabrita.command

Command module.

This module has the CabritaCommand class, which is responsible for:

1. Load and check for valid cabrita.yml file
2. Load and check for valid docker-compose.yml files
3. Generate and add docker services to boxes
4. Generate and add watchers to dashboard

class cabrita.command.**CabritaCommand**(*cabrita_path: str*, *compose_path: tuple*, *background_color: Union[str, NoneType] = 'black'*, *version: str = 'dev'*) → None

Bases: object

Cabrita Command class.

background_color
Return Background Color enum.

Returns BoxColor instance

execute() → None
Execute dashboard to show data in terminal.

Returns None

has_a_valid_compose
Return if Compose data is valid.

Returns bool

has_a_valid_config
Return if Config data is valid.

Returns bool

prepare_dashboard() → None
Prepare the dashboard.

Returns None

read_compose_files() → None
Read docker compose files data.

Returns None

3.2 cabrita.versions

Version package.

Checks version number for upgrades in PyPI

cabrita.versions.**check_version**() → str
Check if it is the latest version.

Compares actual version vs last known version in PyPI, for upgrades

Returns string

cabrita.versions.**versions**() → Union[List[str], NoneType]
Return the version list data from PyPI.

Returns list

3.3 cabrita.abc

Cabrita base package.

Contains abstract classes and general purpose code.

3.4 cabrita.abc.base

Base Module.

3.4.1 ConfigTemplate

Base class for the config template object. This template will process the YAML files. Subclasses are: Config and Compose classes

3.4.2 InspectTemplate

Base class for the Inspector template object. This template will process docker and git status for compose services Subclasses are: DockerInspect and GitInspect

class cabrita.abc.base.**ConfigTemplate** → None
Bases: abc.ABC

Abstract class for processing yaml files.

add_path (*path: str*, *base_path: str = '/home/docs/checkouts/readthedocs.org/user_builds/cabrita/checkouts/latest/docs'*) → None
Add new path for list for each yaml file.

base_path
Return base path for yaml file.

is_valid
Check if yaml is valid.

load_file_data () → None
Load data from yaml file.

First file will be the main file. Every other file will override data, on reversed order list:

Example:

1. docker-compose.yml (main file)
2. docker-compose-dev.yml (override main)
3. docker-compose-pycharm.yml (override dev)

version
Return version value inside yaml file.

class cabrita.abc.base.**InspectTemplate** (*compose*, *interval: int*) → None
Bases: abc.ABC

Abstract class for compose service inspectors.

can_update
Check if inspector can inspect again.

inspect (*service: str*) → None
Run inspect code.

status (*service*)
Return service status.

If can update, start fetching new data calling self.inspect method.

Parameters **service** – docker service name

Returns dict or dashing obj. If not fetch data yet send default widget.

3.5 cabrita.abc.utils

Base utils module.

cabrita.abc.utils.**format_color** (*text: str*, *style: str*, *theme: str = None*) → str
Format string with color using formatStr method.

cabrita.abc.utils.**get_path** (*path: str*, *base_path: str*) → str
Return real path from string.

Converts environment variables to path Converts relative path to full path

cabrita.abc.utils.**get_sentry_client** () → Union[raven.base.Client, NoneType]
Return synchronous Sentry client if DSN is available.

cabrita.abc.utils.**run_command**(*task*, *get_stdout=False*)

Run subprocess command.

The function will attempt to run the task informed and return a boolean for the exit code or the captured std from console.

Arg task (string): the command to run

Arg get_stdout (bool, optional): capture stdout

Arg run_stdout (bool, optional): capture and run stdout

Returns bool or string

3.6 cabrita.components

Components Sub-Package.

This package has the cabrita components for:

box = the Box class (the building block for the dashboard)
config = the Config class (the data from cabrita.yml file)
dashboard = the Dashboard class (convert boxes to dashing widgets and display it)
docker = the DockerInspect class (the runner for inspect docker containers)
git = the GitInspect class (the runner for inspect git data)
watchers = the Watch class (the collection of internal and user watchers for the dashboard

class cabrita.components.**BoxColor**

Bases: enum.Enum

Enum using Blessing Colors values.

Currently not working correctly.

Color	Normal	Bright
black	0	8
red	1	9
green	2	10
yellow	3	11
blue	4	12
magenta	5	13
cyan	6	14
white	7	15

black = 16

blue = 4

cyan = 14

grey = 0

white = 7

yellow = 11

3.7 cabrita.components.box

Box Module.

This module has the Box Class, which is the building block for dashboards.

Each box updates his data in a separate thread in Python.

class cabrita.components.box.**Box**(*background_color: cabrita.components.BoxColor = <BoxColor.black: 16>*, *compose: cabrita.components.config.Compose = None*, *git: cabrita.components.git.GitInspect = None*, *docker: cabrita.components.docker.DockerInspect = None*) → None

Bases: object

Box Class.

The building block of the dashboard.

This class is called inside the CabritaCommand class, using the data from Config and Compose classes.

add_service(*service: str*) → None

Append new service in box services list.

Parameters **service** – service name

Returns None

background_color

Return the Box Color Enum.

Returns BoxColor object

can_update

Check if box data can be updated.

Returns bool

categories

Return if box will show services as columns (categories).

Default: Show as lines.

Returns list

static format_revision(*table_lines: list*) → list

Format revision info to make all lines have the same width.

Example:

```
Revision 1: master@1234abc
Revision 2: epic/refactoring@5678def

After formatting:

Revision 1:           master@1234abc
Revision 2: epic/refactoring@5678def
```

Parameters **table_lines** – list

Returns list

includes
Return the list of the only services which will be included in this box.

Default is: no filters. This option is mutually exclusive with the 'main' parameter.

Returns list

interval
Return interval in seconds for each box data update.

Minimum: 0.5.

Returns float

load_data(*data: dict*) → None
Add data from config class in box.

Parameters **data** – config data parsed from cabrita.yml file.

Returns None

main
Return if this is the main box (the 'main' box parameter).

Default: No. Only one box can be true.

Returns bool

port_detail
Return if box will show external, internal or both docker container exposed ports.

Works in conjunction with 'port_view' parameter. Default: external ports.

Returns PortDetail enum property

port_view
Return if box will show docker container port info (the 'port_view' box parameter).

Default: hidden

Returns PortView enum property

run() → None
Run main code for update box data.

Updates the box widget property.

Returns None

services
Return unique sorted service names inside box.

Returns list

show_git
Return if box will show git information (the 'show_git' box parameter).

Default: True

Returns bool

show_not_found
Return if cabrita will display services when containers not found.

Returns bool

show_revision
Return if box will show git tag/commit hash (the 'show_revision" box parameter).

Default: False

Returns bool

size
Return box size to render in dashboard (the 'size' box parameter).

Default: "large".

Returns string

title
Return box title name (the 'name' box parameter).

Default: "Box".

Returns string

widget
Return dashing widget object.

Returns dashing object

cabrita.components.box.**update_box**(*box*)
Update box data.

This method are called by a thread class to update.

Parameters **box** – the box to update

Returns dashing object

3.8 cabrita.components.config

Config module.

This module has: The **Config class**, which is responsible for handling program options from cabrita.yml file.

The **Compose class**, which is responsible for handling docker-compose data from yamls.

class cabrita.components.config.**Compose** → None
Bases: *cabrita.abc.base.ConfigTemplate*

Main class for Docker-Compose data.

get_build_path(*service_name: str*) → str
Get build full path for service.

Parameters **service_name** – docker service name

Returns str

get_from_service(*service_name: str*, *key: str*) → Any
Get value from key for informed service.

Example: get_from_service("flower", "ports") returns ["5555"]

Parameters

- **service_name** – docker service name
- **key** – search key for service data

Returns List, String, Dict or None

get_ports_from_service (*service: str*) → Union[List, NoneType]
Return ports from service.

The ports can be from the 'ports' or 'expose' parameters in yaml.

Parameters **service** – service name

Returns List or None

is_image (*service_name: str*) → bool
Check if service are built from image or dockerfile.

Parameters **service_name** – docker service name

Returns bool

is_valid
Return if docker-compose are valid.

Returns bool

networks
Return networks configuration in docker-compose yaml files.

Returns dict

services
Return services configuration in docker-compose yaml files.

Returns dict

volumes
Return volumes configuration in docker-compose yaml files.

Returns dict

class cabrita.components.config.**Config** → None
Bases: *cabrita.abc.base.ConfigTemplate*

Cabrita Configuration main class.

background_color
Return background color for box.

Parameter: 'background_color'. Options: Black, Blue, Cyan, Grey, Yellow, White. Default: Black.

Returns BoxColor instance.

background_color_value
Return blessed box color value from enum.

Returns int

boxes
Return box configuration data.

Parameter: 'boxes'. No default available.

Returns dict

compose_files
Return docker-compose paths list.

Parameter: 'compose_files'. Default is empty.

Returns list

generate_boxes (*services: dict*)
Autogenerate boxes in version 0 runs.

The number of boxes are determined by terminal size. The columns visible inside each one are determined by the number of boxes generated.

Parameters **services** – services to be included in dashboard.

Returns dict

static get_compose_path (*compose_path: str*, *base_path: str*) → str
Get docker-compose file full path.

The compose path can be absolute or relative - the 'base_path' will be used to resolve full path.

Parameters

- **compose_path** – the path for docker-compose file.
- **base_path** – the base path for cabrita.yml file.

Returns str

ignore_services
Return ignore services in dashboard.

Parameter: 'ignore_services'. Default: All services viewed.

Returns list

is_valid
Return if configuration is valid.

Calls for the version founded inside yml file: Version 0: No yml available - calls _check_v0(). Version 1: Deprecated - calls _check_v1(). Version 2: Last version - calls _check_v2().

Returns bool

layout
Return dashboard layout.

Parameter: 'layout' Options are: 'horizontal' and 'vertical'. Default: 'horizontal'.

Returns str

title
Return dashboard title.

Parameter: 'title'. Default: 'Docker-Compose'.

Returns str

watchers
Return watchers configuration data.

Parameter: 'watchers'.

No default available. :return:

3.9 cabrita.components.dashboard

Dashboard module.

This module contains the Dashboard class, which is responsible to build all dashing widgets from boxes generate the layout and display it in terminal.

class cabrita.components.dashboard.**Dashboard**(*config: cabrita.components.config.Config*) → None

Bases: object

Dashboard class.

add_box(*box: cabrita.components.box.Box*) → None
Add new box to dashboard.

Parameters **box** – Box to be added

Returns None

all_boxes
Return all boxes widgets in order.

Returns list

run() → None
Run dashboard code.

This code starts fullscreen mode, hides cursor and display the generated layout. To stop press 'q' or 'ctrl-c'.

Returns None

3.10 cabrita.components.docker

Docker module.

This module contains the DockerInspect class which is responsible to inspect docker data from each service in dashboard.

class cabrita.components.docker.**DockerInspect**(*compose: cabrita.components.config.Compose, interval: int, port_view: cabrita.components.docker.PortView, port_detail: cabrita.components.docker.PortDetail, files_to_watch: List[str], services_to_check_git: List[str]*) → None

Bases: *cabrita.abc.base.InspectTemplate*

DockerInspect class.

inspect(*service: str*) → None
Inspect docker container.

Parameters **service** – service name as defined in docker-compose yml.

Returns None

class cabrita.components.docker.**PortDetail**

Bases: enum.Enum

Port Detail for docker ports.

Ports are determined by the 'ports' and 'expose' parameters inside docker-compose.yml files.

- **external**: show external ports only
- **internal**: show internal ports only

- **both**: show both ports.

both = 'both'

external = 'external'

internal = 'internal'

class cabrita.components.docker.**PortView**
Bases: enum.Enum

Port View for docker ports.

Defines where to show port information on box.

Options:

- **hidden**: Do not show info.
- **column**: Show ports info in a separate column
- **name**: Show ports info after service name
- **status**: Show ports info after service status

column = 'column'

hidden = 'hidden'

status = 'status'

3.11 cabrita.components.git

Git module.

This module has the GitInspect class, which is responsible for inspect git data from docker services git branches in dashboard.

class cabrita.components.git.**GitDirection**
Bases: enum.Enum

Git direction for branch evaluation.

Options:

- **ahead**: check for commits ahead branch
- **behind**: check for commits behind branch

ahead = 1

behind = 2

class cabrita.components.git.**GitInspect**(*compose: cabrita.components.config.Compose*, *interval: int*, *target_branch: str*) → None
Bases: *cabrita.abc.base.InspectTemplate*

GitInspect class.

branch_is_dirty(*path: str = None*) → bool
Check if branch is "dirty".

Ie.: has non-committed modifications.

Parameters **path** – path for branch

Returns bool

get_behind_state(*path*)
Check if service need pull and return status.

Parameters **path** – path to search.

Returns string

get_git_revision(*service*)
Return git revision data from service.

Parameters **service** – service name as defined in docker-compose yml.

Returns string

get_git_revision_from_path(*path*, *show_branch: bool = False*) → str
Get last tag and most recent commit hash from path.

Parameters

- **path** – path to search
- **show_branch** – check if add branch name to data

Returns string

inspect(*service: str*) → None
Inspect git data from service.

If service are not running from a docker image, try to find the current branch in service path. From this, try to find the relative diff between this branch and target branch

Parameters **service** – service name as defined in docker-compose yml.

Returns None

3.12 cabrita.components.watchers

Watchers module.

class cabrita.components.watchers.**DockerComposeWatch**(***kwargs*) → None
Bases: *cabrita.components.watchers.Watch*

Docker Compose Watch class.

Watch for docker-compose file status.

class cabrita.components.watchers.**SystemWatch**(*background_color: cabrita.components.BoxColor = <BoxColor.black: 16>, compose: cabrita.components.config.Compose = None, git: cabrita.components.git.GitInspect = None, docker: cabrita.components.docker.DockerInspect = None*) → None
Bases: *cabrita.components.watchers.Watch*

System Watch class.

Watch for system monitors (using psutil).

class cabrita.components.watchers.**UserWatch**(***kwargs*) → None
Bases: *cabrita.components.watchers.Watch*

User Watch class.

Watch for user defined watchers in cabrita.yml.

external
Return watchers using external tools.

Default: []

Returns list

file
Return watchers for file dict.

Default: {}

Returns dict

ping
Return watchers for ping addresses.

Default: {}

Returns dict

class cabrita.components.watchers.**Watch**(*background_color: cabrita.components.BoxColor = <BoxColor.black: 16>*, *compose: cabrita.components.config.Compose = None*, *git: cabrita.components.git.GitInspect = None*, *docker: cabrita.components.docker.DockerInspect = None*) → None
Bases: *cabrita.components.box.Box*

Watch class.

Base class for watchers based on Box class.

interval
Return interval in seconds for each update.

Default: 30 seconds.

Returns float

run() → None
Check if watch can update his data and execute the update.

Returns None

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

G

H

I

L

M

N

P

R

S

T

U

V

W

Y