
Arca

Release 0.1.1

May 07, 2018

Contents

1	Quickstart	3
1.1	Glossary	3
1.2	Example	3
2	Installation	5
2.1	Requirements	5
2.2	Installation	5
3	Settings	7
3.1	Configuring Arca	7
3.2	Basic options	8
4	Tasks	9
4.1	Arguments	9
4.2	Result	10
5	Caching	11
6	Backends	13
6.1	Current Environment	14
6.2	Virtual Environment	14
6.3	Docker	14
6.4	Vagrant	15
6.5	Your own	16
7	Cookbook	17
7.1	I need some files from the repositories	17
7.2	I will be running a lot of tasks from the same repository	17
8	Reference	19
8.1	Arca	19
8.2	Task	22
8.3	Result	22
8.4	Backends	23
8.5	Exceptions	30
8.6	Utils	31
9	Changes	33

9.1	0.1.1 (2018-04-23)	33
9.2	0.1.0 (2018-04-18)	33
9.3	0.1.0a0 (2018-04-13)	33
Python Module Index		35

Arca is a library for running Python functions (callables) from git repositories in various states of isolation. Arca can also cache the results of these callables using [dogpile.cache](#).

1.1 Glossary

- **Arca** - name of the library. When written as *Arca*, the main interface class is being referenced.
- **Task** - definition of the function (callable), consists of a reference to the object and arguments.
- **Backend** - a way of running tasks.

1.2 Example

To run a Hello World example you'll only need the *arca.Arca* and *arca.Task* classes. *Task* is used for defining the task that's supposed to be run in the repositories. *Arca* takes care of all the settings and provides the basic API for running the tasks.

Let's say we have the following file, called `hello_world.py`, in a repository `https://example.com/hello_word.git`, on branch `master`.

```
def say_hello():  
    return "Hello World!"
```

To call the function using Arca, the following example would do so:

```
from arca import Arca, Task  
  
task = Task("hello_world:say_hello")  
arca = Arca()  
  
result = arca.run("https://example.com/hello_word.git", "master", task)  
print(result.output)
```

The code would print `Hello World!`. `result` would be a *Result* instance which currently only has one attribute, `output`, with the output of the function call. If the task fails, *arca.exceptions.BuildError* would be raised.

By default, the *Current Environment Backend* is used to run tasks, which uses the current Python, launching the code in a subprocess. You can learn about backends [here](#).

2.1 Requirements

- Python ≥ 3.6

Requirements for certain backends:

- [Docker](#) (for *Docker Backend* and *Vagrant Backend*)
- [Vagrant](#) (for the *Vagrant Backend*)

2.2 Installation

To install the last stable version:

```
python -m pip install arca
```

Or if you wish to install the upstream version:

```
python -m pip install git+https://github.com/mikicz/arca.git#egg=arca
```


3.1 Configuring Arca

There are multiple ways to configure *Arca* and its backends. (The used options are described *below*.)

1. You can initialize the class and backends directly and set it's options via constructor arguments.

```
from arca import Arca, VenvBackend

arca = Arca(
    base_dir=".custom_arca_dir",
    backend=VenvBackend(cwd="utils")
)
```

This option is the most direct but it has one caveat - options set by this method cannot be overridden by the following methods.

2. You can pass a dict with settings. The keys have to be uppercase and prefixed with `ARCA_`. Keys for backends can be set in two ways. The first is generic `ARCA_BACKEND_<key>`, the second has a bigger priority `ARCA_<backend_name>_BACKEND_<key>`. For example the same setting as above would be written as:

```
arca = Arca(settings={
    "ARCA_BASE_DIR": ".custom_arca_dir",
    "ARCA_BACKEND": "arca.VenvBackend",
    "ARCA_VENV_BACKEND_CWD": "utils",
    "ARCA_BACKEND_CWD": "", # this one is ignored since it has lower priority
})
```

3. You can configure *Arca* with environ variables, with keys being the same as in the second method. Environ variables override settings from the second method.

You can combine these methods as long as you remember that options explicitly specified in constructors cannot be overridden by the settings and environ methods.

3.2 Basic options

This section only describes basic settings, visit the *cookbook* for more.

3.2.1 Arca class

base_dir (*ARCA_BASE_DIR*)

Arca needs to clone repositories and for certain backends also store some other files. This options determines where the files should be stored. The default is `.arca`. If the folder doesn't exist it's created.

backend (*ARCA_BACKEND*)

This options tells how the tasks should be launched. This setting can be provided as a string, a class or a instance. The default is `arca.CurrentEnvironmentBackend`, the *Current Environment Backend*.

3.2.2 Backends

This section describes settings that are common for all the backends.

requirements_location (*ARCA_BACKEND_REQUIREMENTS_LOCATION*)

Tells backends where to look for a requirements file in the repositories, so it must be a relative path. You can set it to `None` to indicate there are no requirements. The default is `requirements.txt`.

cwd (*ARCA_BACKEND_CWD*)

Tells Arca in what working directory the tasks should be launched, so again a relative path. The default is the root of the repository.

CHAPTER 4

Tasks

`arca.Task` instances are used to define what should be run in the repositories. The definition consists of a string representation of a callable and arguments.

EntryPoints (from the `entrypoints` library) are used for defining the callables. Any callable can be used if the result is json-serializable by the standard library `json`.

Let's say we have file `package/test.py` in the repository:

```
def func():
    x = Test()
    return x.run()

class Test:
    def run(self):
        ...
        return "Hello!"

    @staticmethod
    def method():
        x = Test()
        return x.run()
```

In that case, the following two tasks would have the same result:

```
task1 = Task("package.test:func")
task2 = Task("package.test:Test.method")
```

4.1 Arguments

Both positional and keyword arguments can be provided to the task, however they need to be json-serializable (so types `dict`, `list`, `str`, `int`, `float`, `bool` or `None`).

Let's say we the following file `test.py` in the repository:

```
def func(x, *, y=5):  
    return x * y
```

The following tasks would use the fuction (with the default y):

```
task1 = Task("test:func", args=[5]) # -> result would be 25  
task2 = Task("test:func", kwargs={"x": 5}) # -> result would be 25 again
```

Since the x parameter is positional, both ways can be used. However, if we wanted to set y, the task would be set up like this:

```
task1 = Task("test:func", args=[5], kwargs={"y": 10}) # -> 50  
task2 = Task("test:func", kwargs={"x": 5, "y": 10}) # -> 50 again
```

4.2 Result

Anything that's json-serializable can be returned from the endpoints. Unfortunately, due to the way tasks are being launched by various backends, the endpoints **must not** print anything, they can only return values.

Caching

Arca can cache results of the tasks using `dogpile.cache`. The default cache backend is `dogpile.cache.null`, so no caching.

You can setup caching with the backend setting `ARCA_CACHE_BACKEND` and all the arguments needed for setup can be set using `ARCA_CACHE_BACKEND_ARGUMENTS` which can either be a dict or a json string.

Example setup:

```
arca = Arca(settings={
    "ARCA_CACHE_BACKEND": "dogpile.cache.redis",
    "ARCA_CACHE_BACKEND_ARGUMENTS": {
        "host": "localhost",
        "port": 6379,
        "db": 0,
    }
})
```

To see all available cache backends and their settings, please visit the `dogpile.cache` [documentation](#). Some of the other backends might have other python dependencies.

When `Arca` is being initialized, a check is made if the cache backend is writable and readable, which raises an `arca.exceptions.ArcaMisconfigured` if it's not. If the cache requires some python dependency `ModuleNotFoundError` will be raised. If you wish to ignore these errors, `ignore_cache_errors` setting can be used.

Backends

There are currently four different backends. They can also be initialized in few different ways, consistent with general settings. You can use the `ARCA_BACKEND` setting or you can pass a backend keyword directly to *Arca*.

The backend setting can be either a string, class or an instance. All the initializations shown bellow are equivalent, but again, as mentioned in *Configuring Arca*, the backend keyword cannot be overridden by settings or environ variables.

```
from arca import Arca, DockerBackend

Arca(settings={"ARCA_BACKEND": "arca.backend.DockerBackend"})
Arca(settings={"ARCA_BACKEND": DockerBackend})
Arca(backend="arca.backend.DockerBackend")
Arca(backend=DockerBackend)
Arca(backend=DockerBackend())
```

Setting up backends is based on the same principle as setting up *Arca*. You can either pass keyword arguments when initializing the backend class or you can use settings (described in more details in *Configuring Arca*). For example these two calls are equivalent:

```
from arca import Arca, DockerBackend

Arca(settings={
    "ARCA_BACKEND": "arca.backend.DockerBackend",
    "ARCA_BACKEND_PYTHON_VERSION": "3.6.4"
})
Arca(backend=DockerBackend(python_version="3.6.4"))
```

As mentioned in *Basic options*, there are two options common for all backends. (See that section for more details.)

- `requirements_location`
- `cwd`

6.1 Current Environment

arca.backend.CurrentEnvironmentBackend

This backend is the default option, it runs the tasks with the same Python that's used to run Arca, in a subprocess. There are two settings for this backend, to determine how the backend should treat requirements in the repositories.

- **current_environment_requirements:** a path to the requirements of the current environment, the default is `requirements.txt`. `None` would indicate there are no requirements for the current environment.
- **requirements_strategy:** Which approach the backend should take. There are three, the default being `raise`.

(possible settings prefixes: `ARCA_CURRENT_ENVIRONMENT_BACKEND_` and `ARCA_BACKEND_`)

6.1.1 Requirements strategies:

The strategies are defined in a enum, `arca.RequirementsStrategy`. Its values or the string representations can be used in settings.

- `raise`, `RequirementsStrategy.RAISE`: Raise an `arca.exceptions.RequirementsMismatch` if there are any extra requirements in the target repository.
- `ignore`, `RequirementsStrategy.IGNORE`: Ignore any extra requirements.
- `install_extra`, `RequirementsStrategy.INSTALL_EXTRA`: Install the requirements that are extra in the target repository as opposed to the current environment.

6.2 Virtual Environment

arca.backend.VenvBackend

This backend uses the Python virtual environments to run the tasks. The environments are created from the Python used to run Arca and they are shared between repositories that have the same exact requirement file. The virtual environments are stored in folder `venv` in folder determined by the `Arca base_dir` setting, usually `.arca`.

(possible settings prefixes: `ARCA_VENV_BACKEND_` and `ARCA_BACKEND_`)

6.3 Docker

arca.backend.DockerBackend

This backend runs tasks in docker containers. To use this backend the user running Arca needs to be able to interact with `docker` (see [documentation](#)).

This backend firstly creates an image with requirements and dependencies installed so the installation only runs one. By default the images are based on [custom images](#), which have Python and several build tools pre-installed. These images are based on `alpine` and use `pyenv` to install Python. You can specify you want to base your images on a different image with the `inherit_image` setting.

Once `arca` has an image with the requirements installed, it launches a container for each task and kills it when the task finishes. This can be modify by setting `keep_container_running` to `True`, then the container is not killed and can be used by different tasks running from the same repository, branch and commit. This can save time on starting up containers before each task. You can then kill the containers by calling `DockerBackend` method `stop_containers`.

If you're using arca on a CI/CD tool or somewhere docker images are not kept long-term, you can setup pushing images with the installed requirements and dependencies to a docker registry and they will be pulled next time instead of building them each time. It's set using `use_registry_name` and you'll have to be logged in to docker using `docker login`. If you can't use `docker login` (for example in PRs on Travis CI), you can set `registry_pull_only` and Arca will only attempt to pull from the registry and not push new images.

Settings:

- **python_version:** What Python version should be used. In theory any of [these versions](#) could be used, but only CPython 3.6 has been tested. The default is the Python version of the current environment. This setting is ignored if `inherit_image` is set.
- **keep_container_running:** When `True`, containers aren't killed once the task finishes. Default is `False`.
- **apk_dependencies:** For some python libraries, system dependencies are required, for example `libxml2-dev` and `libxslt-dev` are needed for `lxml`. With this settings you can specify a list of system dependencies that will be installed via alpine `apk`. This setting is ignored if `inherit_image` is set since arca can't determined how to install requirements on an unknown system.
- **disable_pull:** Disable pulling prebuilt arca images from Docker Hub and build even the base images locally.
- **inherit_image:** If you don't wish to use the arca images you can specify what image should be used instead.
- **use_registry_name:** Uses this registry to store images with installed requirements and dependencies to,

tries to pull image from the registry before building it locally to save time. * **registry_pull_only:** Disables pushing to registry.

(possible settings prefixes: `ARCA_DOCKER_BACKEND_` and `ARCA_BACKEND_`)

6.4 Vagrant

arca.backend.VagrantBackend

This backend might be reworked completely, consider its API unstable.

If you're extra paranoid you can use Vagrant to completely isolate the tasks. This backend is actually a subclass of `DockerBackend` and uses docker in the VM to run the tasks. Currently the backend works by building the image with requirements and dependencies locally and pushing it to registry (`push_to_registry_name` is required), spinning up a VM for each repository, pulling the image in the VM and running the task. Docker and Vagrant must be runnable by the current user.

The backend inherits all the settings of `DockerBackend` except for `keep_container_running` and has these extra settings:

- **box:** Vagrant box used in the VM. Either has to have docker version `>= 1.8` or not have docker at all, in which case it will be installed when spinning up the VM. The default is [ailispaw/barge](#).
- **provider:** Vagrant provider, default is `virtualbox`. Visit [vagrant docs](#) for more.
- **quiet:** Tells Vagrant and Fabric (which is used to run the task in the VM) to be quiet. Default is `True`. Vagrant and Docker output is logged in separate files for each run in a folder `logs` in the [Arca](#) `base_dir`. The filename is logged in the arca logger (see below)
- **destroy:** Destroy the VM right after the task is finished if `True` (default). If `False` is set, the VM is only halted.

(possible settings prefixes: `ARCA_VAGRANT_BACKEND_` and `ARCA_BACKEND_`)

6.5 Your own

You can also create your own backend and pass it to *Arca*. It has to be a subclass of *arca.BaseBackend* and it has to implement its *run* method.

7.1 I need some files from the repositories

Besides running functions from the repositories, there also might be some files in the repositories that you need, e.g. images for a webpage. With the *Arca* method `static_filename` you can get the absolute path to that file, to where Arca cloned it. The method accepts a relative path (can be a `pathlib.Path` or `str`) to the file in the repository.

Example call (file `images/example.png` from the branch `master` of `https://example.com/hello_word.git`):

```
arca = Arca()

path_to_file = arca.static_filename("https://example.com/hello_word.git",
                                    "master",
                                    "images/example.png")
```

`path_to_file` will be an absolute `Path`.

If the file is not in the repository, `FileNotFoundError` will be raised. If the provided relative path leads out of the repo, `FileOutOfRangeError` will be raised.

7.2 I will be running a lot of tasks from the same repository

Similarly as above, while you're building a webpage you might need to run a lot of tasks from the same repositories, to render all the individual pages. However Arca has some overhead for each launched task, but these two options can speed things up:

7.2.1 Singe pull

This option ensures that each branch is only cloned/pulled once per initialization of *Arca*. You can set it up with the *Arca* `single_pull` option (`ARCA_SINGLE_PULL` setting). This doesn't help to speedup the first task from a repository, however each subsequent will run faster. This setting is quite useful for keeping consistency, since the state of the repository can't change in the middle of running multiple tasks.

You can tell *Arca* to pull again (if a task from that repo/branch is called again) by calling the method *Arca.pull_again*:

```
arca = Arca()

...

# only this specific branch will be pulled again
arca.pull_again(repo="https://example.com/hello_word.git", branch="master")

# all branches from this repo will be pulled again
arca.pull_again(repo="https://example.com/hello_word.git")

# everything will be pulled again
arca.pull_again()
```

7.2.2 Running container

If you're using the *Docker* backend, you can speed up things by keeping the containers for running the tasks running. Since a container for each repository is launched, this can speed up things considerably, because starting up, copying files and shutting down containers takes time.

This can be enabled by setting the `keep_container_running` option to `True`. When you're done with running the tasks you can kill the containers by calling the method *DockerBackend.stop_containers*:

```
arca = Arca(backend=DockerBackend())

...

arca.backend.stop_containers()
```

8.1 Arca

```
class arca.Arca (backend: Union[typing.Callable, arca.backend.base.BaseBackend, str,
                               arca.utils.NotSet] = NOT_SET, settings=None, single_pull=None, base_dir=None,
                               ignore_cache_errors=None) → None
```

Basic interface for communicating with the library, most basic operations should be possible from this class.

Available settings:

- **base_dir**: Directory where cloned repositories and other files are stored (default: `.arca`)
- **single_pull**: Clone/pull each repository only once per initialization (default: `False`)
- **ignore_cache_errors**: Ignore all cache error initialization errors (default: `False`)

```
cache_key (repo: str, branch: str, task: arca.task.Task, git_repo: git.repo.base.Repo) → str
```

Returns the key used for storing results in cache.

```
current_git_hash (repo: str, branch: str, git_repo: git.repo.base.Repo, short: bool = False) → str
```

Parameters

- **repo** – Repo URL
- **branch** – Branch name
- **git_repo** – `Repo` instance.
- **short** – Should the short version be returned?

Returns Commit hash of the currently pulled version for the specified repo/branch

```
get_backend_instance (backend: Union[typing.Callable, arca.backend.base.BaseBackend, str,
                                     arca.utils.NotSet]) → arca.backend.base.BaseBackend
```

Returns a backend instance, either from the argument or from the settings.

Raises `ArcaMisconfigured` – If the instance is not a subclass of `BaseBackend`

get_files (*repo: str, branch: str, *, depth: Union[int, NoneType] = 1, reference: Union[pathlib.Path, NoneType] = None*) → Tuple[git.repo.base.Repo, pathlib.Path]

Either clones the repository if it's not cloned already or pulls from origin. If `single_pull` is enabled, only pulls if the repo/branch combination wasn't pulled again by this instance.

Parameters

- **repo** – Repo URL
- **branch** – Branch name
- **depth** – See `run()`
- **reference** – See `run()`

Returns A `Repo` instance for the repo and a `Path` to the location where the repo is stored.

get_path_to_repo (*repo: str*) → pathlib.Path

Returns a `Path` to the location where all the branches from this repo are stored.

Parameters **repo** – Repo URL

Returns Path to where branches from this repository are cloned.

get_path_to_repo_and_branch (*repo: str, branch: str*) → pathlib.Path

Returns a `Path` to where this specific branch is stored on disk.

Parameters

- **repo** – Repo URL
- **branch** – branch

Returns Path to where the specific branch from this repo is being cloned.

get_repo (*repo: str, branch: str, *, depth: Union[int, NoneType] = 1, reference: Union[pathlib.Path, NoneType] = None*) → git.repo.base.Repo

Returns a `Repo` instance for the branch.

See `run()` for arguments descriptions.

make_region () → dogpile.cache.region.CacheRegion

Returns a `CacheRegion` based on settings.

- Firstly, a backend is selected. The default is `NullBackend` <dogpile.cache.backends.null.NullBackend.
- Secondly, arguments for the backends are generated. The arguments can be passed as a dict to the setting or as a json string. If the arguments aren't a dict or aren't convertible to a dict, `ArcaMisconfigured` is raised.
- Lastly, the cache is tested if it works

All errors can be suppressed by the `ignore_cache_errors` setting.

Raises

- **ModuleNotFoundError** – In case dogpile has trouble importing the library needed for a backend.
- **ArcaMisconfigured** – In case the cache is misconfigured in any way or the cache doesn't work.

pull_again (*repo: Union[str, NoneType] = None, branch: Union[str, NoneType] = None*) → None

When `single_pull` is enables, tells Arca to pull again.

If `repo` and `branch` are not specified, pull again everything.

Parameters

- **repo** – (Optional) Pull again all branches from a specified repository.
- **branch** – (Optional) When `repo` is specified, pull again only this branch from that repository.

Raises `ValueError` – If `branch` is specified and `repo` is not.

repo_id(*repo: str*) → str

Returns an unique identifier from a repo URL for the folder the repo is gonna be pulled in.

run(*repo: str, branch: str, task: arca.task.Task, *, depth: Union[int, NoneType] = 1, reference: Union[pathlib.Path, str, NoneType] = None*) → `arca.result.Result`
Runs the `task` using the configured backend.

Parameters

- **repo** – Target git repository
- **branch** – Target git branch
- **task** – Task which will be run in the target repository
- **depth** – How many commits back should the repo be cloned in case the target repository isn't cloned yet. Defaults to 1, must be bigger than 0. No limit will be used if `None` is set.
- **reference** – A path to a repository from which the target repository is forked, to save bandwidth, `-dissociate` is used if set.

Returns A `Result` instance with the output of the task.

Raises

- `PullError` – If the repository can't be cloned or pulled
- `BuildError` – If the task fails.

save_hash(*repo: str, branch: str, git_repo: git.repo.base.Repo*)

If `single_pull` is enabled, saves the current git hash of the specified repository/branch combination, to indicate that it shouldn't be pull again.

should_cache_fn(*value: arca.result.Result*) → bool

Returns if the result `value` should be cached. By default, always returns `True`, can be overridden.

static_filename(*repo: str, branch: str, relative_path: Union[str, pathlib.Path], *, depth: Union[int, NoneType] = 1, reference: Union[pathlib.Path, str, NoneType] = None*) → `pathlib.Path`

Returns an absolute path to where a file from the repo was cloned to.

Parameters

- **repo** – Repo URL
- **branch** – Branch name
- **relative_path** – Relative path to the requested file
- **depth** – See `run()`
- **reference** – See `run()`

Returns Absolute path to the file in the target repository

Raises

- `FileOutOfRangeError` – If the relative path leads out of the repository path

- **FileNotFoundError** – If the file doesn't exist in the repository.

validate_depth (*depth: Union[int, NoneType]*) → Union[int, NoneType]
Converts the depth to int and validates that the value can be used.

Raises **ValueError** – If the provided depth is not valid

validate_reference (*reference: Union[pathlib.Path, str, NoneType]*) → Union[pathlib.Path, NoneType]
Converts reference to `Path`

Raises **ValueError** – If reference can't be converted to `Path`.

validate_repo_url (*repo: str*)
Validates repo URL - if it's a valid git URL and if Arca can handle that type or repo URL

Raises **ValueError** – If the URL is not valid

8.2 Task

class `arca.Task` (*entry_point: str, *, args: Union[typing.Iterable[typing.Any], NoneType] = None, kwargs: Union[typing.Dict[str, typing.Any], NoneType] = None*) → None

A class for defining tasks the run in the repositories. The task is defined by an entry point, arguments and keyword arguments. The class uses `entrypoints.EntryPoint` to load the callables. As apposed to `EntryPoint`, only objects are allowed, not modules.

Let's presume we have this function in a package `library.module`:

```
def ret_argument (value="Value") :  
    return value
```

This Task would return the default value:

```
>>> Task ("library.module:ret_argument")
```

These two Tasks would returned an overridden value:

```
>>> Task ("library.module:ret_argument", args=["Overridden value"])  
>>> Task ("library.module:ret_argument", kwargs={"value": "Overridden value"})
```

hash

Returns a SHA1 hash of the Task for usage in cache keys.

8.3 Result

class `arca.Result` (*result: Dict[str, Any]*) → None

For storing results of the tasks. So far only has one attribute, `output`.

output = None
The output of the task

8.4 Backends

8.4.1 Abstract classes

class `arca.BaseBackend` (***settings*)

Bases: `object`

Abstract class for all the backends, implements some basic functionality.

Available settings:

- **requirements_location**: Relative path to the requirements file in the target repositories. (default is `requirements.txt`)
- **cwd**: Relative path to the required working directory. (default is `" "`, the root of the repo)

get_requirements_file (*path: pathlib.Path*) → `Union[pathlib.Path, NoneType]`

Gets a `Path` for the requirements file if it exists in the provided `path`, returns `None` otherwise.

get_requirements_hash (*requirements_file: pathlib.Path*) → `str`

Returns an SHA1 hash of the contents of the `requirements_path`.

get_setting (*key, default=NOT_SET*)

Gets a setting for the key.

Raises `KeyError` – If the key is not set and default isn't provided.

get_settings_keys (*key*)

Parameters can be set through two settings keys, by a specific setting (eg. `ARCA_DOCKER_BACKEND_KEY`) or a general `ARCA_BACKEND_KEY`. This function returns the two keys that can be used for this setting.

inject_arca (*arca*)

After backend is set for a `Arca` instance, the instance is injected to the backend, so settings can be accessed, files accessed etc. Also runs settings validation of the backend.

run (*repo: str, branch: str, task: arca.task.Task, git_repo: git.repo.base.Repo, repo_path: pathlib.Path*)

→ `arca.result.Result`

Executes the script and returns the result.

Must be implemented by subclasses.

Parameters

- **repo** – Repo URL
- **branch** – Branch name
- **task** – The requested `Task`
- **git_repo** – A `Repo` of the repo/branch
- **repo_path** – `Path` to the location where the repo is stored.

Returns The output of the task in a `Result` instance.

serialized_task (*task: arca.task.Task*) → `Tuple[str, str]`

Returns the name of the task definition file and its contents.

snake_case_backend_name

CamelCase -> camel_case

```
class arca.backend.base.BaseRunInSubprocessBackend (**settings)
    Bases: arca.backend.base.BaseBackend

    Abstract class for backends which run scripts in subprocess.

    get_or_create_environment (repo: str, branch: str, git_repo: git.repo.base.Repo, repo_path:
                                pathlib.Path) → str
        Abstract method which must be implemented in subclasses, which must return a str path to a Python
        executable which will be used to run the script.

        See BaseBackend.run to see arguments description.

    run (repo: str, branch: str, task: arca.task.Task, git_repo: git.repo.base.Repo, repo_path: pathlib.Path)
        → arca.result.Result
        Gets a path to a Python executable by calling the abstract method get_image_for_repo and runs the
        task using subprocess.Popen

        See BaseBackend.run to see arguments description.
```

8.4.2 Current environment

```
class arca.CurrentEnvironmentBackend (**settings)
    Bases: arca.backend.base.BaseRunInSubprocessBackend

    Uses the current Python to run the tasks, however they're launched in a subprocess.

    Available settings:

    • current_environment_requirements: Path to the requirements file of the current requirements. Set to
      None if there are none. (default is requirements.txt)

    • requirements_strategy: How should requirements differences be handled. Can be either strings or a
      RequirementsStrategy value. See the RequirementsStrategy Enum for available strategies
      (default is RequirementsStrategy.RAISE)

    get_or_create_environment (repo: str, branch: str, git_repo: git.repo.base.Repo, repo_path:
                                pathlib.Path) → str
        Handles the requirements of the target repository (based on requirements_strategy) and returns
        the path to the current Python executable.

    get_requirements_set (file: pathlib.Path) → Set[str]

        Parameters file – Path to a requirements.txt file.

        Returns Set of the requirements from the file with newlines and extra characters removed.

    handle_requirements (repo: str, branch: str, repo_path: pathlib.Path)
        Checks the differences and handles it using the selected strategy.

    install_requirements (*, path: Union[pathlib.Path, NoneType] = None, requirements:
                            Union[typing.Iterable[str], NoneType] = None, _action: str = 'install')
        Installs requirements, either from a file or from a iterable of strings.

        Parameters

        • path – Path to a requirements.txt file. Has priority over requirements.

        • requirements – A iterable of strings of requirements to install.

        • _action – For testing purposes, can be either install or uninstall

        Raises

        • BuildError – If installing fails.
```

- **ValueError** – If both `file` and `requirements` are undefined.
- **ValueError** – If `_action` not `install` or `uninstall`.

class `arca.RequirementsStrategy`

Enum for defining strategy for *CurrentEnvironmentBackend*

IGNORE = `'ignore'`

Ignores all difference of requirements of the current environment and the target repository.

INSTALL_EXTRA = `'install_extra'`

Installs the extra requirements.

RAISE = `'raise'`

Raises an exception if there are some extra requirements in the target repository.

8.4.3 Python virtual environment

class `arca.VenvBackend` (***settings*)

Bases: *arca.backend.base.BaseRunInSubprocessBackend*

Uses Python virtual environments (see `venv`), the tasks are then launched in a `subprocess`. The virtual environments are shared across repositories when they have the exact same requirements. If the target repository doesn't have requirements, it also uses a virtual environment, but just with no extra packages installed.

There are no extra settings for this backend.

create_or_get_venv (*path: pathlib.Path*) → `pathlib.Path`

Gets the name of the virtualenv from `get_virtualenv_name()`, checks if it exists already, creates it and installs requirements otherwise. The virtualenvs are stored in a folder based on the *Arca* `base_dir` setting.

Parameters `path` – `Path` to the cloned repository.

get_or_create_environment (*repo: str, branch: str, git_repo: git.repo.base.Repo, repo_path: pathlib.Path*) → `str`

Handles the requirements in the target repository, returns a path to a executable of the virtualenv.

get_virtualenv_name (*requirements_file: pathlib.Path*) → `str`

Returns a name of the virtualenv that should be used for this repository.

Either:

- `SHA1` of the requirements file and *Arca* version
- `no_requirements_file` if the requirements file doesn't exist.

Parameters `requirements_file` – `Path` to where the requirements file should be in the cloned repository

8.4.4 Docker

class `arca.DockerBackend` (***kwargs*)

Bases: *arca.backend.base.BaseBackend*

Runs the tasks in Docker containers.

Available settings:

- **python_version** - set a specific version, current env. python version by default

- **keep_container_running** - stop the container right away (default) or keep it running
- **apk_dependencies** - a list of dependencies to install via alpine
- **disable_pull** - build all locally
- **inherit_image** - instead of using the default base Arca image, use this one
- **use_registry_name** - use this registry to store images with requirements and dependencies
- **registry_pull_only** - only use the registry to pull images, don't push updated

build_image (*image_name: str, image_tag: str, build_context: pathlib.Path, requirements_file: Union[pathlib.Path, NoneType], dependencies: Union[typing.List[str], NoneType]*) → `docker.models.images.Image`

Builds an image for specific requirements and dependencies, based on the settings.

Parameters

- **image_name** – How the image should be named
- **image_tag** – And what tag it should have.
- **build_context** – Path to the cloned repository.
- **requirements_file** – Path to the requirements file in the repository (or `None` if it doesn't exist)
- **dependencies** – List of dependencies (in the formalized format)

Returns The Image instance.

build_image_from_inherited_image (*image_name: str, image_tag: str, build_context: pathlib.Path, requirements_file: Union[pathlib.Path, NoneType]*) → `docker.models.images.Image`

Builds a image with installed requirements from the inherited image. (Or just tags the image if there are no requirements.)

See `build_image()` for parameters descriptions.

check_docker_access ()

Creates a `DockerClient` for the instance and checks the connection.

Raises `BuildError` – If docker isn't accessible by the current user.

container_running (*container_name*) → `Union[docker.models.containers.Container, NoneType]`

Finds out if a container with name `container_name` is running.

Returns `Container` if it's running, `None` otherwise.

get_arca_base (*pull=True*)

Returns the name and tag of image that has the basic build dependencies installed with just pyenv installed, with no python installed. (Builds or pulls the image if it doesn't exist locally.)

get_container_name (*repo: str, branch: str, git_repo: git.repo.base.Repo*)

Returns the name of the container used for the repo.

get_dependencies () → `Union[typing.List[str], NoneType]`

Returns the `apk_dependencies` setting to a standardized format.

Raises `ArcaMisconfigured` – if the dependencies can't be converted into a list of strings

Returns List of dependencies, `None` if there are none.

get_dependencies_hash (*dependencies*)

Returns a SHA1 hash of the dependencies for usage in image names/tags.

get_image (*image_name*, *image_tag*) → `docker.models.images.Image`

Returns a `Image` instance for the provided name and tag.

get_image_for_repo (*repo*: *str*, *branch*: *str*, *git_repo*: `git.repo.base.Repo`, *repo_path*: `pathlib.Path`)
→ `docker.models.images.Image`

Returns an image for the specific repo (based on settings and requirements).

1. Checks if the image already exists locally
2. Tries to pull it from registry (if `use_registry_name` is set)
3. Builds the image
4. Pushes the image to registry so the image is available next time (if `registry_pull_only` is not set)

See `run()` for parameters descriptions.

get_image_name (*requirements_file*: `Union[pathlib.Path, NoneType]`, *dependencies*: `Union[typing.List[str], NoneType]`) → *str*

Returns the name for images with installed requirements and dependencies.

get_image_tag (*requirements_file*: `Union[pathlib.Path, NoneType]`, *dependencies*: `Union[typing.List[str], NoneType]`) → *str*

Returns the tag for images with the dependencies and requirements installed.

64-byte hexadecimal strings cannot be used as docker tags, so the prefixes are necessary. Double hashing the dependencies and requirements hash to make the final tag shorter.

Prefixes:

- Image type:
 - i – Inherit image
 - a – Arca base image
- Requirements:
 - r – Does have requirements
 - s – Doesn't have requirements
- Dependencies:
 - d – Does have dependencies
 - e – Doesn't have dependencies

Possible outputs:

- Inherited images:
 - *ise* – no requirements
 - *ide_<requirements_hash>* – with requirements
- From Arca base image:
 - *ase* – no requirements and no dependencies
 - *asd_<dependencies_hash>* – only dependencies
 - *are_<requirements_hash>* – only requirements
 - *ard_<hash(dependencies_hash + requirements_hash)>* – both requirements and dependencies

get_image_with_installed_dependencies (*image_name: str, dependencies: Union[typing.List[str], NoneType]*) → Tuple[str, str]

Return name and tag of a image, based on the Arca python image, with installed dependencies defined by `apk_dependencies`.

Parameters

- **image_name** – Name of the image which will be ultimately used for the image.
- **dependencies** – List of dependencies in the standardized format.

get_inherit_image () → Tuple[str, str]

Parses the `inherit_image` setting, checks if the image is present locally and pulls it otherwise.

Returns Returns the name and the tag of the image.

Raises **ArcaMisconfiguration** – If the image can't be pulled from registries.

get_or_build_image (*name: str, tag: str, dockerfile: Union[str, typing.Callable[..., str]], *, pull=True, build_context: Union[pathlib.Path, NoneType] = None*)

A proxy for commonly built images, returns them from the local system if they exist, tries to pull them if pull isn't disabled, otherwise builds them by the definition in `dockerfile`.

Parameters

- **name** – Name of the image
- **tag** – Image tag
- **dockerfile** – Dockerfile text or a callable (no arguments) that produces Dockerfile text
- **pull** – If the image is not present locally, allow pulling from registry (default is `True`)
- **build_context** – A path to a folder. If it's provided, docker will build the image in the context of this folder. (eg. if `ADD` is needed)

get_python_base (*python_version, pull=True*)

Returns the name and tag of an image with specified `python_version` installed, if the image doesn't exist locally, it's pulled or built (extending the image from `get_arca_base()`).

get_python_version () → str

Returns either the specified version from settings or a string of the `sys.executable` version.

image_exists (*image_name, image_tag*)

Returns if the image exists locally.

push_to_registry (*image: docker.models.images.Image, image_tag: str*)

Pushes a local image to a registry based on the `use_registry_name` setting.

Raises **PushToRegistryError** – If the push fails.

run (*repo: str, branch: str, task: arca.task.Task, git_repo: git.repo.base.Repo, repo_path: pathlib.Path*) → `arca.result.Result`

Gets or builds an image for the repo, gets or starts a container for the image and runs the script.

Parameters

- **repo** – Repository URL
- **branch** – Branch name
- **task** – *Task* to run.
- **git_repo** – *Repo* of the cloned repository.
- **repo_path** – *Path* to the cloned location.

start_container (*image: docker.models.images.Image, container_name: str, repo_path: pathlib.Path*) → *docker.models.containers.Container*
 Starts a container with the image and name `container_name` and copies the repository into the container.

stop_containers ()
 Stops all containers used by this instance of the backend.

tar_files (*path: pathlib.Path*) → *bytes*
 Returns a tar with the git repository.

tar_runner ()
 Returns a tar with the runner script.

tar_task_definition (*name: str, contents: str*) → *bytes*
 Returns a tar with the task definition.

Parameters

- **name** – Name of the file
- **contents** – Contents of the definition, utf-8

try_pull_image_from_registry (*image_name, image_tag*) → *Union[docker.models.images.Image, NoneType]*
 Tries to pull a image with the tag `image_tag` from registry set by `use_registry_name`. After the image is pulled, it's tagged with `image_name:image_tag` so lookup can be made locally next time.

Returns A `Image` instance if the image exists, `None` otherwise.

validate_settings ()
 Validates the provided settings.

- Checks `inherit_image` format.
- Checks `use_registry_name` format.
- Checks that `apk_dependencies` is not set when `inherit_image` is set.

Raises `ArcaMisconfigured` – If some of the settings aren't valid.

8.4.5 Vagrant

class `arca.VagrantBackend` (***kwargs*)
 Bases: `arca.backend.docker.DockerBackend`

Uses Docker in Vagrant.

Inherits settings from `DockerBackend`:

- **python_version**
- **apk_dependencies**
- **disable_pull**
- **inherit_image**
- **use_registry_name**

Adds new settings:

- **box** - what Vagrant box to use (must include `docker >= 1.8` or no `docker`)
- **provider** - what provider should Vagrant user

- **quiet** - Keeps the extra vagrant logs quiet.
- **destroy** - Destroy or just halt the VMs (default is True)

check_vagrant_access ()

Raises **BuildError** – If Vagrant is not installed.

create_vagrant_file (*repo: str, branch: str, git_repo: git.repo.base.Repo, repo_path: pathlib.Path*)

Creates a Vagrantfile in the target dir with the required settings and the required docker image. The image is built locally if not already pushed.

get_vagrant_file_location (*repo: str, branch: str, git_repo: git.repo.base.Repo, repo_path: pathlib.Path*) → *pathlib.Path*

Returns a directory where Vagrantfile should be. Based on repo, branch and tag of the used docker image.

run (*repo: str, branch: str, task: arca.task.Task, git_repo: git.repo.base.Repo, repo_path: pathlib.Path*)

Gets or creates Vagrantfile, starts up a VM with it, executes Fabric script over SSH, returns result.

validate_settings ()

Runs *arca.DockerBackend.validate_settings()* and checks extra:

- box format
- provider format
- use_registry_name is set

8.5 Exceptions

exception *arca.exceptions.ArcaException*

Bases: *Exception*

A base exception from which all exceptions raised by Arca are subclassed.

exception *arca.exceptions.ArcaMisconfigured*

Bases: *ValueError, arca.exceptions.ArcaException*

An exception for all cases of misconfiguration.

exception *arca.exceptions.BuildError* (*args, extra_info=None, **kwargs)

Bases: *arca.exceptions.ArcaException*

Raised if the task fails.

extra_info = None

Extra information what failed

exception *arca.exceptions.FileOutOfRangeError*

Bases: *ValueError, arca.exceptions.ArcaException*

Raised if *relative_path* in *Arca.static_filename* leads outside the repository.

exception *arca.exceptions.PullError*

Bases: *arca.exceptions.ArcaException*

Raised if repository can't be cloned or pulled.

exception *arca.exceptions.PushToRegistryError* (*args, full_output=None, **kwargs)

Bases: *arca.exceptions.ArcaException*

Raised if pushing of images to Docker registry in *DockerBackend* fails.

```
full_output = None
```

Full output of the push command

```
exception arca.exceptions.RequirementsMismatch(*args, diff=None, **kwargs)
```

Bases: `ValueError`, `arca.exceptions.ArcaException`

Raised if the target repository has extra requirements compared to the current environment if the `requirements_strategy` of `CurrentEnvironmentBackend` is set to `arca.backends.RequirementsStrategy.raise`.

```
diff = None
```

The extra requirements

```
exception arca.exceptions.TaskMisconfigured
```

Bases: `ValueError`, `arca.exceptions.ArcaException`

Raised if Task is incorrectly defined.

8.6 Utils

```
class arca.utils.LazySettingProperty(*, key, default=NOT_SET, convert: Callable = None)
    → None
```

For defining properties for the `Arca` class and for the backends. The property is evaluated lazily when accessed, getting the value from settings using the instances method `get_setting`. The property can be overridden by the constructor.

```
class arca.utils.NotSet
```

For default values which can't be None.

```
class arca.utils.Settings (data: Union[typing.Dict[str, typing.Any], NoneType] = None) → None
```

A class for handling *Arca* settings.

get (*keys, default: Any = NOT_SET) → Any

Returns values from the settings in the order of keys, the first value encountered is used.

Example:

```
>>> settings = Settings({"ARCA_ONE": 1, "ARCA_TWO": 2})
>>> settings.get("one")
1
>>> settings.get("one", "two")
1
>>> settings.get("two", "one")
2
>>> settings.get("three", "one")
1
>>> settings.get("three", default=3)
3
>>> settings.get("three")
Traceback (most recent call last):
...
KeyError:
```

Parameters

- **keys** – One or more keys to get from settings. If multiple keys are provided, the value of the first key that has a value is returned.
- **default** – If none of the `options` aren't set, return this value.

Returns A value from the settings or the default.

Raises

- **ValueError** – If no keys are provided.
- **KeyError** – If none of the keys are set and no default is provided.

`arca.utils.get_hash_for_file(repo: git.repo.base.Repo, path: Union[str, pathlib.Path]) → str`
Returns the hash for the specified path.

Equivalent to `git rev-parse HEAD:X`

Parameters

- **repo** – The repo to check in
- **path** – The path to a file or folder to get hash for

Returns The hash

`arca.utils.get_last_commit_modifying_files(repo: git.repo.base.Repo, *files) → str`
Returns the hash of the last commit which modified some of the files (or files in those folders).

Parameters

- **repo** – The repo to check in.
- **files** – List of files to check

Returns Commit hash.

`arca.utils.is_dirty(repo: git.repo.base.Repo) → bool`
Returns if the repo has been modified (including untracked files).

`arca.utils.load_class(location: str) → type`
Loads a class from a string and returns it.

```
>>> from arca.utils import load_class
>>> load_class("arca.backend.BaseBackend")
<class 'arca.backend.base.BaseBackend'>
```

Raises **ArcaMisconfigured** – If the class can't be loaded.

9.1 0.1.1 (2018-04-23)

Updated gitpython to 2.1.9

9.2 0.1.0 (2018-04-18)

Initial release

Changes:

- Updated PyPI description and metadata

9.3 0.1.0a0 (2018-04-13)

Initial alfa release

a

`arca.exceptions`, [30](#)

`arca.utils`, [31](#)

A

Arca (class in arca), 19
arca.exceptions (module), 30
arca.utils (module), 31
ArcaException, 30
ArcaMisconfigured, 30

B

BaseBackend (class in arca), 23
BaseRunInSubprocessBackend (class in arca.backend.base), 23
build_image() (arca.DockerBackend method), 26
build_image_from_inherited_image() (arca.DockerBackend method), 26
BuildError, 30

C

cache_key() (arca.Arca method), 19
check_docker_access() (arca.DockerBackend method), 26
check_vagrant_access() (arca.VagrantBackend method), 30
container_running() (arca.DockerBackend method), 26
create_or_get_venv() (arca.VenvBackend method), 25
create_vagrant_file() (arca.VagrantBackend method), 30
current_git_hash() (arca.Arca method), 19
CurrentEnvironmentBackend (class in arca), 24

D

diff (arca.exceptions.RequirementsMismatch attribute), 31
DockerBackend (class in arca), 25

E

extra_info (arca.exceptions.BuildError attribute), 30

F

FileOutOfRangeError, 30

full_output (arca.exceptions.PushToRegistryError attribute), 30

G

get() (arca.utils.Settings method), 31
get_arca_base() (arca.DockerBackend method), 26
get_backend_instance() (arca.Arca method), 19
get_container_name() (arca.DockerBackend method), 26
get_dependencies() (arca.DockerBackend method), 26
get_dependencies_hash() (arca.DockerBackend method), 26
get_files() (arca.Arca method), 19
get_hash_for_file() (in module arca.utils), 32
get_image() (arca.DockerBackend method), 26
get_image_for_repo() (arca.DockerBackend method), 27
get_image_name() (arca.DockerBackend method), 27
get_image_tag() (arca.DockerBackend method), 27
get_image_with_installed_dependencies() (arca.DockerBackend method), 27
get_inherit_image() (arca.DockerBackend method), 28
get_last_commit_modifying_files() (in module arca.utils), 32
get_or_build_image() (arca.DockerBackend method), 28
get_or_create_environment() (arca.backend.base.BaseRunInSubprocessBackend method), 24
get_or_create_environment() (arca.CurrentEnvironmentBackend method), 24
get_or_create_environment() (arca.VenvBackend method), 25
get_path_to_repo() (arca.Arca method), 20
get_path_to_repo_and_branch() (arca.Arca method), 20
get_python_base() (arca.DockerBackend method), 28
get_python_version() (arca.DockerBackend method), 28
get_repo() (arca.Arca method), 20
get_requirements_file() (arca.BaseBackend method), 23
get_requirements_hash() (arca.BaseBackend method), 23
get_requirements_set() (arca.CurrentEnvironmentBackend method), 24

`get_setting()` (arca.BaseBackend method), 23
`get_settings_keys()` (arca.BaseBackend method), 23
`get_vagrant_file_location()` (arca.VagrantBackend method), 30
`get_virtualenv_name()` (arca.VenvBackend method), 25

H

`handle_requirements()` (arca.CurrentEnvironmentBackend method), 24
`hash` (arca.Task attribute), 22

I

`IGNORE` (arca.RequirementsStrategy attribute), 25
`image_exists()` (arca.DockerBackend method), 28
`inject_arca()` (arca.BaseBackend method), 23
`INSTALL_EXTRA` (arca.RequirementsStrategy attribute), 25
`install_requirements()` (arca.CurrentEnvironmentBackend method), 24
`is_dirty()` (in module arca.utils), 32

L

`LazySettingProperty` (class in arca.utils), 31
`load_class()` (in module arca.utils), 32

M

`make_region()` (arca.Arca method), 20

N

`NotSet` (class in arca.utils), 31

O

`output` (arca.Result attribute), 22

P

`pull_again()` (arca.Arca method), 20
`PullError`, 30
`push_to_registry()` (arca.DockerBackend method), 28
`PushToRegistryError`, 30

R

`RAISE` (arca.RequirementsStrategy attribute), 25
`repo_id()` (arca.Arca method), 21
`RequirementsMismatch`, 31
`RequirementsStrategy` (class in arca), 25
`Result` (class in arca), 22
`run()` (arca.Arca method), 21
`run()` (arca.backend.base.BaseRunInSubprocessBackend method), 24
`run()` (arca.BaseBackend method), 23
`run()` (arca.DockerBackend method), 28
`run()` (arca.VagrantBackend method), 30

S

`save_hash()` (arca.Arca method), 21
`serialized_task()` (arca.BaseBackend method), 23
`Settings` (class in arca.utils), 31
`should_cache_fn()` (arca.Arca method), 21
`snake_case_backend_name` (arca.BaseBackend attribute), 23
`start_container()` (arca.DockerBackend method), 28
`static_filename()` (arca.Arca method), 21
`stop_containers()` (arca.DockerBackend method), 29

T

`tar_files()` (arca.DockerBackend method), 29
`tar_runner()` (arca.DockerBackend method), 29
`tar_task_definition()` (arca.DockerBackend method), 29
`Task` (class in arca), 22
`TaskMisconfigured`, 31
`try_pull_image_from_registry()` (arca.DockerBackend method), 29

V

`VagrantBackend` (class in arca), 29
`validate_depth()` (arca.Arca method), 22
`validate_reference()` (arca.Arca method), 22
`validate_repo_url()` (arca.Arca method), 22
`validate_settings()` (arca.DockerBackend method), 29
`validate_settings()` (arca.VagrantBackend method), 30
`VenvBackend` (class in arca), 25