restfulgrok Documentation

Release 1.2.0

Espen Angell Kristiansen

Sep 27, 2017

Contents

Fabric (http://fabfile.org) tasks for Amazon Web Services with some extra utilities for Ubuntu.

CHAPTER 1

Install

```
$ pip install awsfabrictasks
```

CHAPTER 2

Code and issues

Get the sourcecode, and submit issues at our github repo: https://github.com/espenak/awsfabrictasks

CHAPTER 3

Contribute

1. Create an issue describing your problem, improvement, etc (unless you are fixing an existing issue).
2. Fix the issue and create a pull request. This is a perfect example of what a pull request should include: https://github.com/espenak/awsfabrictasks/pull/9.

CHAPTER 4

Wiki

Please contribute your tips, tricks and guides to the Wiki.

CHAPTER 5

Getting started

Fabric

Learn how to use Fabric.

The awsfab command

Fabric is great for remote execution because it allows you to run a task on any SSH-server with the following syntax:

```
$ fab -H server1,server2,server3 task1 task2
```

The problem with Fabric on AWS EC2 is that we do not have a static dns address to give to `-H`. `awsfab` wraps `fab` and allows us to use:

```
$ awsfab -E <Name-tag of an EC2 instance>,<Name-tag of another....> task1 task2
```

If your instance is not tagged with a name (the tag must be capitalized: `Name`), you can use `--ec2ids` instead.

Required settings

See *awsfab_settings.py — Settings*.

CHAPTER 6

Making a fabfile.py and use awsfabrictasks

Example fabfile.py

Create a `fabfile.py` just as you would with Fabric, and import tasks from `awsfabrictasks`:

```
from fabric.api import task, run
from awsfabrictasks.decorators import ec2instance


############################
# Add some of our own tasks
############################

@task
def uname():
    """
    Run ``uname -a``
    """
    run('uname -a')


@task
@ec2instance(nametag='tst')
def example_nametag_specific_task():
    """
    Example of using ``@ec2instance``.
    Enables us to run::

        awsfab example_nametag_specific_task``

    and have it automatically use the EC2 instance tagged with ``Name="tst"``.
    """
    run('uname -a')
```

```
####################
# Import awsfab tasks
####################
from awsfabrictasks.ec2.tasks import *
from awsfabrictasks.regions import *
from awsfabrictasks.conf import *
```

Using the example

List basic information about your instances with:

```
$ awsfab ec2_list_instances
```

Start one of your existing EC2 instances (the example assumes it is tagged with `Name="mytest"`):

```
$ awsfab -E mytest ec2_start_instance
```

Login (SSH) to the instance we just started:

```
$ awsfab -E mytest ec2_login
```

See:

```
$ awsfab -l
```

or *Tasks* for more tasks.

Launch/create new EC2 instances

See *Example awsfab_settings.py* for and example of how to setup your EC2 launch configurations. After you have added `EC2_LAUNCH_CONFIGS` to your `awsfab_settings.py`, simply run:

```
$ awsfab ec2_launch_instance:<nametag>
```

where `<nametag>` is the name you want to tag your new instance with. You will be asked to choose a config from `EC2_LAUNCH_CONFIGS`, and to confirm all your choices before the instance in created.

CHAPTER 7

More task-examples

The best examples are the provided tasks. Just browse the source, or use the `[source]` links in the *tasks docs*.

CHAPTER 8

Documentation

Tasks

awsfabrictasks.ec2.tasks

General tasks for AWS management.

awsfabrictasks.ec2.tasks.**ec2_add_tag**
Add tag to EC2 instance. Fails if tag already exists.

Parameters

- **tagname** – Name of the tag to set (required).
- **value** – Value to set the tag to. Default to empty string.

awsfabrictasks.ec2.tasks.**ec2_set_tag**
Set tag on EC2 instance. Overwrites value if tag exists.

Parameters

- **tagname** – Name of the tag to set (required).
- **value** – Value to set the tag to. Default to empty string.

awsfabrictasks.ec2.tasks.**ec2_remove_tag**
Remove tag from EC2 instance. Fails if tag does not exist.

Parameters **tagname** – Name of the tag to remove (required).

awsfabrictasks.ec2.tasks.**ec2_launch_instance**
Launch new EC2 instance.

Parameters

- **name** – The name to tag the EC2 instance with (required)
- **configname** – Name of the configuration in awsfab_settings.EC2_LAUNCH_CONFIGS. Prompts for input if not provided as an argument.

awsfabrictasks.ec2.tasks.**ec2_start_instance**
Start EC2 instance.

Parameters **nowait** – Set to `True` to let the EC2 instance start in the background instead of waiting for it to start. Defaults to `False`.

awsfabrictasks.ec2.tasks.**ec2_stop_instance**
Stop EC2 instance.

Parameters **nowait** – Set to `True` to let the EC2 instance stop in the background instead of waiting for it to start. Defaults to `False`.

awsfabrictasks.ec2.tasks.**ec2_list_instances**
List EC2 instances in a region (defaults to awsfab_settings.DEFAULT_REGION).

Parameters

- **region** – The region to list instances in. Defaults to ‘‘awsfab_settings.DEFAULT_REGION.
- **full** – Print all attributes, or just the most useful ones? Defaults to `False`.

awsfabrictasks.ec2.tasks.**ec2_print_instance**
Print EC2 instance info.

Parameters **full** – Print all attributes, or just the most useful ones? Defaults to `False`.

awsfabrictasks.ec2.tasks.**ec2_login**
Log into the host specified by –hosts, –ec2names or –ec2ids.

Aborts if more than one host is specified.

awsfabrictasks.ec2.tasks.**ec2_rsync_download_dir**
Sync the contents of `remote_dir` into `local_dir`. E.g.: if `remote_dir` is `/etc`, and `local_dir` is `/tmp`, the `/tmp/etc` will be created on the local host, and filled with all files in `/etc` on the EC2 instance.

Parameters

- **remote_dir** – The remote directory to download into local_dir.
- **local_dir** – The local directory.
- **rsync_args** – Arguments for `rsync`. Defaults to `-av`.
- **noconfirm** – If this is `True`, we will not ask for confirmation before proceeding with the operation. Defaults to `False`.

awsfabrictasks.ec2.tasks.**ec2_rsync_upload_dir**
Sync the contents of `local_dir` into `remote_dir` on the EC2 instance. E.g.: if `local_dir` is `/etc`, and `remote_dir` is `/tmp`, the `/tmp/etc` will be created on the EC2 instance, and filled with all files in `/etc` on the local host.

Parameters

- **local_dir** – The local directory to upload to the EC2 instance.
- **remote_dir** – The remote directory to upload local_dir into.
- **rsync_args** – Arguments for `rsync`. Defaults to `-av`.
- **noconfirm** – If this is `True`, we will not ask for confirmation before proceeding with the operation. Defaults to `False`.

awsfabrictasks.ec2.regions

awsfabrictasks.regions.**list_regions**
List all regions.

awsfabrictasks.regions.**list_zones**
List zones in the given region.

Parameters **region** – Defaults to awsfab_settings.DEFAULT_REGION.

API

awsfabrictasks.conf

class awsfabrictasks.conf.**Settings**
Bases: object

Settings object inspired by django.conf.settings.

as_dict()
Get all settings (uppercase attributes on this object) as a dict.

clear_settings()
Clear all settings (intended for testing). Deletes all uppercase attributes.

pprint()
Prettyprint the settings.

reset_settings(***settings*)
Reset settings (intended for testing). Shortcut for:

```
clear_settings()
set_settings(**settings)
```

set_settings(***settings*)
Set all all the given settings as attributes of this object.

Raises **ValueError** – If any of the keys in settings is not uppercase.

awsfabrictasks.conf.**print_settings**
Pretty-print the settings as they are seen by the system.

awsfabrictasks.utils

exception awsfabrictasks.utils.**InvalidLogLevel**
Bases: exceptions.KeyError

Raised when *getLoglevelFromString()* gets an invalid loglevelstring.

awsfabrictasks.utils.**compute_localfile_md5sum**(*localfile*)
Compute the hex-digested md5 checksum of the given localfile.

Parameters **localfile** – Path to a file on the local filesystem.

awsfabrictasks.utils.**configureStreamLogger**(*loggername*, *level*)
Configure a stdout/stderr logger (logging.StreamHandler) with the given loggername and level. If you are configuring logging for a task, use *configureStreamLoggerForTask()*.

This is suitable for log-configuration for a single task, where the user specifies a loglevel.

Returns The configured logger.

awsfabrictasks.utils.**configureStreamLoggerForTask**(*modulename*, *taskname*, *loglevel*)
Configure logging for a task.

Shortcut for:

```
configureStreamLogger(modulename + '.' + taskname, loglevel)
```

Example (note that what you put in the loglevel docs for your task depends on how you use the logger):

```
@task
mytask(loglevel='INFO'):
    """
    Does some task.

    :param loglevel:
        Controls the amount of output:

            QUIET --- No output.
            INFO --- Only produce output for changes.
            DEBUG --- One line of output for each file.

    Defaults to "INFO".
    """
    log = configureStreamLoggerForTask(__name__, 's3_syncupload_dir',
                                       getLoglevelFromString(loglevel))
    log.info('Hello world')
```

awsfabrictasks.utils.**force_noslashend**(*path*)
Return `path` with any trailing `/` removed.

awsfabrictasks.utils.**force_slashend**(*path*)
Return `path` suffixed with `/` (path is unchanged if it is already suffixed with `/`).

awsfabrictasks.utils.**getLoglevelFromString**(*loglevelstring*)
Lookup `loglevelstring` in *loglevel_stringmap*.

Raises ***InvalidLogLevel*** – If loglevelstring is not in *loglevel_stringmap*.

Returns The loglevel.

Return type int

awsfabrictasks.utils.**guess_contenttype**(*filename*)
Return the content-type for the given `filename`. Uses `mimetypes.guess_type()`.

awsfabrictasks.utils.**localpath_to_slashpath**(*path*)
Replace `os.sep` in `path` with `/`.

awsfabrictasks.utils.**parse_bool**(*data*)
Return `True` if data is one of:: `'true', 'True', True`. Otherwise, return `False`.

awsfabrictasks.utils.**rsyncformat_path**(*source_dir*, *sync_content=False*)
rsync uses `/` in the source directory to determine if we should sync a directory or the contents of a directory. How rsync works:

Sync contents: Source path ending with `/` means sync the contents (just as if we used `/*` except that `*` does not include hidden files).

Sync the directory: Source path NOT ending with `/` means sync the directory. I.e.: If the source is `/etc/init.d`, and the destination is `/tmp`, the contents of `/etc/init.d` is copied into `/tmp/init.d/`.

This is error-prone, and the consequences can be severe if combined with `--delete`. Therefore, we use a boolean to distinguish between these two methods of specifying source directory, and reformat the path using *force_slashend()* and *force_noslashend()*.

Parameters

- **source_dir** – The source directory. May be a remote directory (i.e.: [USER@]HOSTNAME:PATH), or a local directory.
- **sync_content** – Normally the function automatically makes sure `local_dir` is not suffixed with `/`, which makes rsync copy `local_dir` into `remote_dir`. With `sync_content=True`, the content of `local_dir` is synced into `remote_dir` instead.

awsfabrictasks.utils.**slashpath_to_localpath**(*path*)
Replace `/` in `path` with `os.sep`.

awsfabrictasks.utils.**sudo_chattr**(*remote_path*, *owner=None*, *mode=None*)
Run *sudo_chown()* and *sudo_chmod()* on `remote_path`. If owner or mode is None, their corresponding function is not called.

awsfabrictasks.utils.**sudo_chmod**(*remote_path*, *mode*)
Run `sudo chmod <mode> remote_path`.

awsfabrictasks.utils.**sudo_chown**(*remote_path*, *owner*)
Run `sudo chown <owner> remote_path`.

awsfabrictasks.utils.**sudo_mkdir_p**(*remote_path*, ***chattr_kw*)
`sudo mkdir -p <remote_path>` followed by :func:‘sudo_chattr‘(remote_path, **chattr_kw).

awsfabrictasks.utils.**sudo_upload_dir**(*local_dir*, *remote_dir*, ***chattr_kw*)
Upload all files and directories in `local_dir` to `remote_dir`. Directories are created with *sudo_mkdir_p()* and files are uploaded with *sudo_upload_file()*. `chattr_kw` is forwarded in both cases.

awsfabrictasks.utils.**sudo_upload_file**(*local_path*, *remote_path*, ***chattr_kw*)
Use sudo to upload a file from `local_path` to `remote_path` and run *sudo_chattr()* with the given `chattr_kw` as arguments.

awsfabrictasks.utils.**sudo_upload_string_to_file**(*string_to_upload*, *remote_path*, ***chattr_kw*)
Create a tempfile containing `string_to_upload`, and use *sudo_upload_file()* to upload the tempfile. Removes the tempfile when the upload is complete or if it fails.

Parameters

- **string_to_upload** – The string to write to the tempfile.
- **remote_path** – See *sudo_upload_file()*.
- **chattr_kw** – See *sudo_upload_file()*.

awsfabrictasks.utils.**loglevel_stringmap = {‘INFO’: 20, ‘QUIET’: 50, ‘WARN’: 30, ‘CRITICAL’: 50, ‘ERROR’:**
Map of strings to loglevels (for the logging module)

awsfabrictasks.ubuntu

Ubuntu utilities.

awsfabrictasks.ubuntu.**set_locale**(*locale='en_US'*)
Set locale to avoid the warnings from perl and others about locale failures.

awsfabrictasks.ec2.api

exception awsfabrictasks.ec2.api.**Ec2RegionConnectionError**(*region*)
Bases: exceptions.Exception

Raised when we fail to connect to a region.

exception awsfabrictasks.ec2.api.**InstanceLookupError**
Bases: exceptions.LookupError

Base class for instance lookup errors.

exception awsfabrictasks.ec2.api.**MultipleInstancesWithSameNameError**
Bases: *awsfabrictasks.ec2.api.InstanceLookupError*

Raised when multiple instances with the same nametag is discovered. (see: *Ec2InstanceWrapper.get_by_nametag()*)

exception awsfabrictasks.ec2.api.**NoInstanceWithNameFound**
Bases: *awsfabrictasks.ec2.api.InstanceLookupError*

Raised when no instace with the requested name is found in *Ec2InstanceWrapper.get_by_nametag()*.

exception awsfabrictasks.ec2.api.**NotExactlyOneInstanceError**
Bases: *awsfabrictasks.ec2.api.InstanceLookupError*

Raised when more than one instance is found when expecting exactly one instance.

exception awsfabrictasks.ec2.api.**WaitForStateError**
Bases: exceptions.Exception

Raises when *wait_for_state()* times out.

class awsfabrictasks.ec2.api.**Ec2InstanceWrapper**(*instance*)
Bases: object

Wraps a boto.ec2.instance.Instance with convenience functions.

Variables ***instance*** – The boto.ec2.instance.Instance.

Parameters **instance** – A boto.ec2.instance.Instance object.

add_instance_to_env()
Add self to fabric.api.env.ec2instances[self.get_ssh_uri()], and register the key-pair for the instance in fabric.api.env.key_filename.

classmethod get_by_instanceid(*instanceid*)
Connect to AWS and get the EC2 instance with the given instance ID.

Parameters **instanceid_with_optional_region** – Parsed with *parse_instanceid()* to find the region and name.

Raises

- ***Ec2RegionConnectionError*** – If connecting to the region fails.
- **LookupError** – If the requested instance was not found in the region.

Returns A *Ec2InstanceWrapper* contaning the requested instance.

classmethod get_by_nametag(*instancename_with_optional_region*)
Connect to AWS and get the EC2 instance with the given Name-tag.

Parameters **instancename_with_optional_region** – Parsed with *parse_instancename()* to find the region and name.

Raises

- ***Ec2RegionConnectionError*** – If connecting to the region fails.
- ***InstanceLookupError*** – Or one of its subclasses if the requested instance was not found in the region.

Returns A *Ec2InstanceWrapper* contaning the requested instance.

classmethod get_by_tagvalue(*tags={}*, *region=None*)
Connect to AWS and get the EC2 instance with the given tag:value pairs.

:param tags A string like 'role=testing,fake=yes' to AND a set of ec2 instance tags

Parameters **region** – optional.

Raises ***Ec2RegionConnectionError*** – If connecting to the region fails.

Returns A list of :class:'Ec2InstanceWrapper's containing the matching instances.

classmethod get_exactly_one_by_tagvalue(*tags*, *region=None*)
Use *get_by_tagvalue()* to find instances by `tags`, but raise `LookupError` if not exactly one instance is found.

classmethod get_from_host_string()
If an instance has been registered in `fabric.api.env` using *add_instance_to_env()*, this method can be used to get the instance identified by `fabric.api.env.host_string`.

get_ssh_key_filename()
Get the SSH indentify filename (.pem-file) for the instance. Searches `awsfab_settings.KEYPAIR_PATH` for `"<instance.key_name>.pem"`.

Raises **LookupError** – If the key is not found.

get_ssh_uri()
Get the SSH URI for the instance.

Returns "<instance.tags['awsfab-ssh-user']>@<instance.public_dns_name>"

is_running()
Return `True` if state=='running'.

is_stopped()
Return `True` if state=='stopped'.

prettyname()
Return a pretty-formatted name for this instance, using the Name-tag if the instance is tagged with it.

class `awsfabrictasks.ec2.api.`**Ec2LaunchInstance**(*extra_tags={}*, *configname=None*, *configname_help='Please select one of the following configurations:'*, *duplicate_name_protection=True*)

Bases: `object`

Launch instances configured in `awsfab_settings.EC2_LAUNCH_CONFIGS`.

Example:

```
launcher = Ec2LaunchInstance(extra_tags={'Name': 'mytest'})
launcher.confirm()
instance = launcher.run_instance()
```

Note that this class is optimized for the following use case:

- Create one or more instances (initialize one or more Ec2LaunchInstance).
- Confirm using *confirm()* or *confirm_many()*.
- Launch each instance using meth:*Ec2LaunchInstance.run_instance* or *Ec2LaunchInstance.run_many_instances()*.
- Use *Ec2LaunchInstance.wait_for_running_state_many()* to wait for all instances to launch.
- Do something with the running instances.

Example of launching many instances:

a = Ec2LaunchInstance(extra_tags={'Name': 'a'}) b = Ec2LaunchInstance(extra_tags={'Name': 'b'}) Ec2LaunchInstance.confirm_many([a, b]) Ec2LaunchInstance.run_many_instances([a, b]) # Note: that we can start doing stuff with `a` and `b` that does not # require the instances to be running, such as setting tags. Ec2LaunchInstance.wait_for_running_state_many([a, b])

Initialize the launcher. Runs *create_config_ask_if_none()*.

Parameters

- **configname** – Name of a configuration in `awsfab_settings.EC2_LAUNCH_CONFIGS`. If it is `None`, we ask the user for the configfile.
- **configname_help** – The help to show above the prompt for configname input (only used if `configname` is `None`.

confirm()
Use `prettyprint()` to show the user their choices, and ask for confirmation. Runs `fabric.api.abort()` if the user does not confirm the choices.

classmethod confirm_many(*launchers*)
Loop through Use `prettyprint()` to show the user their choices, and ask for confirmation. Runs `fabric.api.abort()` if the user does not confirm the choices.

create_config_ask_if_none()
Set *kw* and *conf* using *configname*. Prompt the user for a configname if bool(*configname*) is `False`.

get_all_tags()
Merge tags from the awsfab_settings.EC2_LAUNCH_CONFIGS config, and the `extra_tags` parameter for __init__, and return the resulting dict.

prettyformat()
Prettyformat the configuration.

run_instance()
Run/launch the configured instance, and add the tags to the instance (*get_all_tags()*).

Returns The launched instance.

classmethod run_many_instances(*launchers*)
Loop through `launchers` and run *run_instance()*.

Parameters

- **launchers** – List of Ec2LaunchInstance objects.
- **kwargs** – Forwarded to *wait_for_running_state()*.

classmethod wait_for_running_state_many(*launchers*, ***kwargs*)
Loop through `launchers` and run *wait_for_running_state()*.

Parameters

- **launchers** – List of Ec2LaunchInstance objects that have been lauched with *Ec2LaunchInstance.run_instance()*.
- **kwargs** – Forwarded to *wait_for_running_state()*.

conf = None
A config dict from awsfab_settings.EC2_LAUNCH_CONFIGS.

configname = None
See the docs for the __init__ parameter.

configname_help = None
See the docs for the __init__ parameter.

instance = None
The instance launced by *run_instance()*. None when run_instance() has not been invoked.

kw = None
Keyword arguments for `run_instances()`.

tag_retry_count = 4
Number of times to retry when adding tags gets EC2ResponseError.

tag_retry_sleep = 2
Number of seconds to sleep before retrying when adding tags gets EC2ResponseError.

`awsfabrictasks.ec2.api.`**ec2_rsync**(**args*, ***kwargs*)
Deprecated since version 1.0.13: Use *ec2_rsync_upload()* instead.

`awsfabrictasks.ec2.api.`**ec2_rsync_download**(*remote_dir*, *local_dir*, *rsync_args='-av'*, *sync_content=False*)
rsync `remote_dir` on the current EC2 instance (the one returned by *Ec2InstanceWrapper.get_from_host_string()*) into `local_dir`.

Parameters **sync_content** – Normally the function automatically makes sure `local_dir` is not suffixed with `/`, which makes rsync copy `local_dir` into `remote_dir`. With `sync_content=True`, the content of `local_dir` is synced into `remote_dir` instead.

`awsfabrictasks.ec2.api.`**ec2_rsync_download_command**(*instancewrapper*, *remote_dir*, *local_dir*, *rsync_args='-av'*, *sync_content=False*)
Returns the rsync command used by *ec2_rsync_download()*. Takes the same parameters as *ec2_rsync_download()*, except for the first parameter, `instancewrapper`, which is a *Ec2InstanceWrapper* object.

`awsfabrictasks.ec2.api.`**ec2_rsync_upload**(*local_dir*, *remote_dir*, *rsync_args='-av'*, *sync_content=False*)
rsync `local_dir` into `remote_dir` on the current EC2 instance (the one returned by *Ec2InstanceWrapper.get_from_host_string()*).

Parameters **sync_content** – Normally the function automatically makes sure `local_dir` is not suffixed with `/`, which makes rsync copy `local_dir` into `remote_dir`. With `sync_content=True`, the content of `local_dir` is synced into `remote_dir` instead.

`awsfabrictasks.ec2.api.`**ec2_rsync_upload_command**(*instancewrapper*, *local_dir*, *remote_dir*, *rsync_args='-av'*, *sync_content=False*)
Returns the rsync command used by *ec2_rsync_upload()*. Takes the same parameters as *ec2_rsync_upload()*, except for the first parameter, `instancewrapper`, which is a *Ec2InstanceWrapper* object.

awsfabrictasks.ec2.api.**parse_instanceid**(*instanceid_with_optional_region*)
Parse instance id with an optional region-name prefixed. Region name is specified by prefixing the instanceid with `<regionname>:`.

Returns (region, instanceid) where region defaults to `awsfab_settings.DEFAULT_REGION` if not prefixed to the id.

awsfabrictasks.ec2.api.**parse_instancename**(*instancename_with_optional_region*)
Just like *parse_instanceid()*, however this is for instance names. We keep them as separate functions in case they diverge in the future.

Returns (region, instanceid) where region defaults to `awsfab_settings.DEFAULT_REGION` if not prefixed to the name.

awsfabrictasks.ec2.api.**print_ec2_instance**(*instance*, *full=False*, *indentspaces=3*)
Print attributes of an ec2 instance.

Parameters

- **instance** – A `boto.ec2.instance.Instance` object.
- **full** – Print all attributes? If not, a subset of the attributes are printed.
- **indentspaces** – Number of spaces to indent each line in the output.

awsfabrictasks.ec2.api.**wait_for_running_state**(*instanceid*, ***kwargs*)
Shortcut for `wait_for_state(instanceid, 'running', **kwargs)`.

awsfabrictasks.ec2.api.**wait_for_state**(*instanceid, state_name, sleep_intervals=[15, 5], last_sleep_repeat=40*)
Poll the instance with `instanceid` until its `state_name` matches the desired `state_name`.

The first poll is performed without any delay, and the rest of the polls are performed according to `sleep_intervals`.

Parameters

- **instanceid** – ID of an instance.
- **state_name** – The state_name to wait for.
- **sleep_intervals** – List of seconds to wait between each poll for state. The first poll is made immediately, then we wait for sleep_intervals[0] seconds before the next poll, and repeat for each item in sleep_intervals. Then we repeat for `last_sleep_repeat` using the last item in `sleep_intervals` as the timout for each wait.
- **last_sleep_repeat** – Number of times to repeat the last item in `sleep_intervals`. If this is 20, we will wait for a maximum of `sum(sleep_intervals) + sleep_intervals[-1]*20`.

awsfabrictasks.ec2.api.**wait_for_stopped_state**(*instanceid*, ***kwargs*)
Shortcut for `wait_for_state(instanceid, 'stopped', **kwargs)`.

awsfabrictasks.ec2.api.**zipit**(*ss*)
Returns a string containing a user_data compatible gzip-file of the zipped ss input. Note(using zlib alone is not sufficient - we need a zipfile structure)

awsfabrictasks.decorators

awsfabrictasks.decorators.**ec2instance**(*nametag=None*, *instanceid=None*, *tags=None*, *region=None*)
Wraps the decorated function to execute as if it had been invoked with `--ec2names` or `--ec2ids`.

awsfabrictasks.s3.api

exception awsfabrictasks.s3.api.**S3ConnectionError**(*msg='Could not connect S3'*)
Bases: exceptions.Exception

Raised when we fail to connect to S3.

exception awsfabrictasks.s3.api.**S3ErrorBase**
Bases: exceptions.Exception

Base class for all S3 errors. Never raised directly.

exception awsfabrictasks.s3.api.**S3FileDoesNotExist**(*s3file*)
Bases: *awsfabrictasks.s3.api.S3FileErrorBase*

Raised when an *S3File* does not exist.

Parameters **s3file** – A *S3File* object.

exception awsfabrictasks.s3.api.**S3FileErrorBase**(*s3file*)
Bases: *awsfabrictasks.s3.api.S3ErrorBase*

Base class for all *S3File* errors. Never raised directly.

Parameters **s3file** – A *S3File* object.

exception awsfabrictasks.s3.api.**S3FileExistsError**(*s3file*)
Bases: *awsfabrictasks.s3.api.S3FileErrorBase*

Raised when trying to overwrite an existing *S3File*, unless overwriting is requested.

Parameters **s3file** – A *S3File* object.

exception awsfabrictasks.s3.api.**S3FileNoInfo**(*s3file*)
Bases: *awsfabrictasks.s3.api.S3FileErrorBase*

Raised when trying to use *S3File* metadata before performing a HEAD request.

Parameters **s3file** – A *S3File* object.

class awsfabrictasks.s3.api.**S3ConnectionWrapper**(*connection*)
Bases: object

S3 connection wrapper.

Parameters **bucket** – A boto.rds.bucket.DBInstance object.

classmethod **get_bucket**(*bucketname*)
Get the requested bucket.

Shortcut for:

```
S3ConnectionWrapper.get_connection().connection.get_bucket(bucketname)
```

Parameters **bucketname** – Name of an S3 bucket.

classmethod **get_bucket_using_pattern**(*bucketname*)
Same as *get_bucket()*, however the bucketname is filtered through *settingsformat_bucketname()*.

classmethod **get_connection**()
Connect to S3 using awsfab_settings.AUTH.

Returns S3ConnectionWrapper object.

class awsfabrictasks.s3.api.**S3File**(*bucket*, *key*)
Bases: object

Simplifies working with keys in S3 buckets.

delete()
Delete the key/file from the bucket.

Raises *S3FileDoesNotExist* – If the key does not exist in the bucket.

etag_matches_localfile(*localfile*)
Return True if the file at the path given in localfile has an md5 hex-digested checksum matching the etag of this S3 key.

exists()
Return True if the key/file exists in the S3 bucket.

get_contents_as_string()
Download the file and return it as a string.

get_contents_to_filename(*localfile*)
Download the file to the given localfile.

get_etag()
Return the etag (the md5sum)

set_contents_from_filename(*localfile*, *overwrite=False*)
Upload localfile.

Parameters **overwrite** – If True, overwrite if the key/file exists.

Raises *S3FileExistsError* – If overwrite==True and the key exists in the bucket.

set_contents_from_string(*data*, *overwrite=False*)
Write data to the S3 file.

Parameters **overwrite** – If True, overwrite if the key/file exists.

Raises *S3FileExistsError* – If overwrite==True and the key exists in the bucket.

class awsfabrictasks.s3.api.**S3Sync**(*bucket*, *local_dir*, *s3prefix*)
Bases: object

Makes it easy to sync files to and from S3. This class does not make any changes to the local filesyste, or S3, it only makes it easy to write function that works with hierarkies of files synced locally and on S3.

A good example is the sourcecode for awsfabrictasks.s3.tasks.s3_syncupload_dir().

Parameters

- **bucket** – A boto.rds.bucket.DBInstance object.
- **local_dir** – The local directory.
- **local_dir** – The S3 key prefix that corresponds to local_dir.

iterfiles()
Iterate over all files both local and within the S3 prefix. Yields *S3SyncIterFile* objects.

How it works:

•Uses *dirlist_absfilenames()* to get all local files in the local_dir.

•Uses *s3list_s3filedict()* to get all S3 files in the s3prefix.

•Uses these two sets of information to create *S3SyncIterFile* objects.

class awsfabrictasks.s3.api.**S3SyncIterFile**
Bases: object

Objects of this class is yielded by *S3Sync.iterfiles()*. Contains info about where the file exists, its local and S3 path (even if it does not exist).

both_exists()
Returns True if *localexists* and *s3exists*.

create_localdir()
Create the directory containing *localpath* if it does not exist.

download_s3file_to_localfile()
create_localdir() and download the file at *s3path* to *localpath*.

etag_matches_localfile()
Shortcut for:

```
self.s3file.etag_matches_localfile(self.localpath)
```

localexists = None
Local file exists?

localpath = None
The local path. Always set. Use *localexists* if you want to know if the local file exists.

s3exists = None
S3 file exists?

s3file = None
A *S3File* object. Use *s3exists* if you want to know if the S3 file exists.

s3path = None
The S3 path. Always set. Use *s3exists* if you want to know if the S3 file exists.

awsfabrictasks.s3.api.**dirlist_absfilenames**(*dirpath*)
Get all the files within the given dirpath as a set of absolute filenames.

awsfabrictasks.s3.api.**iter_bucketcontents**(*bucket*, *prefix*, *match*, *delimiter*, *formatter=<function <lambda>>*)
Iterate over items given bucket, yielding items formatted for output.

Parameters

- **bucket** – A class:*boto.s3.bucket.Bucket* object.
- **prefix** – The prefix to list. Defaults to empty string, which lists all items in the root directory.
- **match** – A Unix shell style pattern to match. Matches against the entire key name (in filesystem terms: the absolute path).

  Uses the fnmatch python module. The match is case-sensitive.

  Examples:

  ```
  *.jpg
  *2012*example*.log
  icon-*.png
  ```

- **delimiter** – The delimiter to use. Defaults to /.
- **formatter** – Formatter callback to use to format each key. Not used on Prefix-keys (directories). The callback should take a key as input, and return a string.

See also:

http://docs.amazonwebservices.com/AmazonS3/latest/dev/ListingKeysHierarchy.html

`awsfabrictasks.s3.api.`**`localpath_to_s3path`**(*localdir*, *localpath*, *s3prefix*)
Convert a local filepath into a S3 path within the given `s3prefix`.

Parameters

- **`localdir`** – The local directory that corresponds to `s3prefix`.
- **`localpath`** – Path to a file within `localdir`.
- **`s3prefix`** – Prefix to use for the file on S3.

Example:: >>> localpath_to_s3path('/mydir', '/mydir/hello/world.txt', 'my/test') 'my/test/hello/world.txt'

`awsfabrictasks.s3.api.`**`s3list_s3filedict`**(*bucket*, *prefix*)
Get all the keys with the given `prefix` as a dict with key-name as key and the key-object wrappen in a *`S3File`* as value.

`awsfabrictasks.s3.api.`**`s3path_to_localpath`**(*s3prefix*, *s3path*, *localdir*)
Convert a s3 filepath into a local filepath within the given `localdir`.

Parameters

- **`s3prefix`** – Prefix used for the file on S3.
- **`s3path`** – Path to a file within `s3prefix`.
- **`localdir`** – The local directory that corresponds to `s3prefix`.

Example:: >>> s3path_to_localpath('mydir/', 'mydir/hello/world.txt', '/my/test') '/my/test/hello/world.txt'

`awsfabrictasks.s3.api.`**`settingsformat_bucketname`**(*bucketname*)
Returns `awsfab_settings.S3_BUCKET_PATTERN.format(bucketname=bucketname)`.

See also:

`awsfabrictasks.default_settings.S3_BUCKET_PATTERN`.

awsfab_settings.py — Settings

Required setup

awsfabrictasks uses a settings system similar to the Django settings module. You add your settings to `awsfab_settings.py`.

Required configuration

Create a file named `awsfab_settings.py`, and add your AWS Security credentials:

```
AUTH = {'aws_access_key_id': 'Access Key ID',
        'aws_secret_access_key': 'Secret Access Key'}
```

You find these under `My account -> Security Credentials` on http://aws.amazon.com/.

.pem-key

The `.pem`-keys (key pairs) for you instances (the ones used for SSH login) must be added to `~/.ssh/` or the current directory. You can change these directories with the `awsfab_settings.KEYPAIR_PATH` variable (see *Default settings*).

Local override

You may override settings in `awsfab_settings_local.py`, which is typically used to store authentication credentials outside your version control system (i.e: with git you would add awsfab_settings_local.py to .gitignore).

Override name of settings module

`awsfab_settings.py` defines the `awsfab_settings` python module. Importing the module works because awsfab adds the current working directory to your `PYTHONPATH`. You can override the name of this module using `--awsfab-settings <modulename>`. When you override the settings module, you also override the name of the corresponding local override. The local override is always `<settings-module-name>_local`.

To sum this up, you do the following to create a custom settings file named `my_awsfab_settings.py` with a local override:

- Create `my_awsfab_settings.py`.
- Create `my_awsfab_settings_local.py` (optional).
- Run with:

```
$ awsfab --awsfab-settings my_awsfab_settings
```

Warning: The `--awsfab-settings` module can NOT be a dotted path (it can be `my_test_module`, but NOT `my.test.module`).

Example awsfab_settings.py

```
# Config file for awsfabrictasks.
#
# This is a Python module, and it is imported just as a regular Python module.
# Every variable with an uppercase-only name is a setting.

AUTH = {'aws_access_key_id': 'XXXXXXXXXXXXXXXXXXXXX',
        'aws_secret_access_key': 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'}

DEFAULT_REGION = 'eu-west-1'


###################################################################
# Self documenting map of AMIs
# - You are not required to use this, but it makes it easier to read
#   EC2_LAUNCH_CONFIGS.
###################################################################
ami = {
    'ubuntu-10.04-lts': 'ami-fb665f8f'
}
```

```
################################################################
# Example user_data
# This script will be passed to the new instance at boot
# time and run late in the boot sequence.
# It can be used to do arbitrarily complex setup tasks.
# info: http://ubuntu-smoser.blogspot.co.uk/2010/03/introducing-cloud-inits-cloud-
↪config.html
################################################################
user_data_example = """#!/bin/sh
echo ========== Hello World: $(date) ==========
echo "I have been up for $(cut -d\  -f 1 < /proc/uptime) sec"
"""


############################################################
# Configuration for ec2_launch_instance
############################################################
EC2_LAUNCH_CONFIGS = {
    'ubuntu-10.04-lts-micro': {
        'description': 'Ubuntu 10.04 on the least expensive instance type.',

        # Ami ID (E.g.: ami-fb665f8f)
        'ami': ami['ubuntu-10.04-lts'],

        # One of: m1.small, m1.large, m1.xlarge, c1.medium, c1.xlarge, m2.xlarge, m2.
↪2xlarge, m2.4xlarge, cc1.4xlarge, t1.micro
        'instance_type': 't1.micro',

        # List of security groups
        'security_groups': ['allowssh'],

        # Use the ``list_regions`` task to see all available regions
        'region': DEFAULT_REGION,

        # The name of the key pair to use for instances (See http://console.aws.
↪amazon.com -> EC2 -> Key Pairs)
        'key_name': 'awstestkey',

        # The availability zone in which to launch the instances. This is
        # automatically prefixed by ``region``.
        'availability_zone': 'b',

        # Tags to add to the instances. You can use the ``ec2_*_tag`` tasks or
        # the management interface to manage tags. Special tags:
        #   - Name: Should not be in this dict. It is specified when launching
        #           an instance (needs to be unique for each instance).
        #   - awsfab-ssh-user: The ``awsfab`` tasks use this user to log into your␣
↪instance.
        'tags': {
            'awsfab-ssh-user': 'ubuntu'
        },
        'user_data': user_data_example
    }
}
```

```
######################################################
# Add your own settings here
######################################################

MYCOOLSTUFF_REMOTE_DIR = '/var/www/stuff'
```

Default settings

awsfabrictasks.default_settings.**AUTH = {}**
The AWS access key. Should look something like this:

```
AUTH = {'aws_access_key_id': 'XXXXXXXXXXXXXXXXX',
        'aws_secret_access_key': 'aaaaaaaaaaaa\BBBBBBBBB\dsaddad'}
```

awsfabrictasks.default_settings.**DEFAULT_REGION = 'eu-west-1'**
The default AWS region to use with the commands where REGION is supported.

awsfabrictasks.default_settings.**EC2_INSTANCE_DEFAULT_SSHUSER = 'root'**
Default ssh user if the `awsfab-ssh-user` tag is not set

awsfabrictasks.default_settings.**EC2_LAUNCH_CONFIGS = {}**
Configuration for ec2_launch_instance (see the docs)

awsfabrictasks.default_settings.**EXTRA_SSH_ARGS = '-o StrictHostKeyChecking=no'**
Extra SSH arguments. Used with `ssh` and `rsync`.

awsfabrictasks.default_settings.**KEYPAIR_PATH = ['.', '~/.ssh/']**
Directories to search for "<key_name>.pem". These paths are filtered through os.path.expanduser, so paths like `~/.ssh/` works.

awsfabrictasks.default_settings.**S3_BUCKET_PATTERN = '{bucketname}'**
S3 bucket suffix. This is used for all tasks taking bucketname as parameter. The actual bucketname used become:

```
S3_BUCKET_PATTERN.format(bucketname=bucketname)
```

This is typically used to add your domain name or company name to all bucket names, but avoid having to type the entire name for each task. Examples:

```
S3_BUCKET_PATTERN = '{bucketname}.example.com'
S3_BUCKET_PATTERN = 'example.com.{bucketname}'
```

The default, "`{bucketname}`", uses the bucket name as provided by the user without any changes.

See also:

awsfabrictasks.s3.api.S3ConnectionWrapper.get_bucket_using_pattern(), *awsfabrictasks.s3.api.settingsformat_bucketname()*

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

F

G

I

K

L

M

N

P

R

S

T

W

Z