restfulgrok Documentation

Release 1.0.6

Espen Angell Kristiansen

May 20, 2012

CONTENTS

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

CHAPTER

ONE

INSTALL

```
$ pip install awsfabrictasks
```

CHAPTER

TWO

CODE AND ISSUES

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

CHAPTER

THREE

GETTING STARTED

3.1 Fabric

Learn how to use Fabric.

3.2 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.

3.3 Required settings

See *awsfab_settings.py — Settings*.

CHAPTER

FOUR

MAKING A FABFILE.PY AND USE AWSFABRICTASKS

4.1 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 *
```

4.2 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.

4.3 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

FIVE

MORE TASK-EXAMPLES

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

CHAPTER

SIX

DOCUMENTATION

6.1 Tasks

6.1.1 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.
- **noconfirm** – Do not require the user to confirm creating the instance? Defaults to `False`.

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.

6.1.2 awsfabrictasks.ec2.regions

6.2 API

6.2.1 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.

pprint()
Prettyprint the settings.

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

6.2.2 awsfabrictasks.utils

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()`.

6.2.3 awsfabrictasks.ubuntu

Ubuntu utilities.

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

6.2.4 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.**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.
- **LookupError** – 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.
- **LookupError** – If no matching instance was found in the region.

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

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>"

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

awsfabrictasks.ec2.api.**create_hostsfile_from_nametags**(*nametags=[]*, *suffix='.ec2private'*, *hostsfile_template='\n127.0.0.1 localhost\n\n# The following lines are desirable for IPv6 capable hosts\n::1 ip6-localhost ip6-loopback\nfe00::0 ip6-localnet\nff00::0 ip6-mcastprefix\nff02::1 ip6-allnodes\nff02::2 ip6-allrouters\nff02::3 ip6-allhosts\n\n{custom_hosts}\n'*)

Create a `/etc/hosts` file from the list of `nametags`. The result will be a hostsfile with private_ip_address as IP, and `nametag+suffix` as hostname.

awsfabrictasks.ec2.api.**ec2_rsync**(*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.**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)`.

6.2.5 awsfabrictasks.decorators

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

6.3 awsfab_settings.py — Settings

6.3.1 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*).

6.3.2 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).

6.3.3 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': 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'}

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'
}


############################################################
# Configuration for ec2_launch_instance
############################################################
EC2_LAUNCH_CONFIGS = {
    'ubuntu-10.04-lts-micro': {
        # 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, m2.4xl
        '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_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'
        }
    }
}
```

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

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

6.3.4 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.

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

a

awsfabrictasks.conf, **??**
awsfabrictasks.decorators, **??**
awsfabrictasks.default_settings, **??**
awsfabrictasks.ec2.api, **??**
awsfabrictasks.ec2.tasks, **??**
awsfabrictasks.regions, **??**
awsfabrictasks.ubuntu, **??**
awsfabrictasks.utils, **??**