
awsauthhelper Documentation

Release 1.5.1

Drew J. Sonne

Apr 03, 2018

Contents

1	Introduction	3
1.1	Changes	3
1.2	License	3
1.3	Dependencies	3
1.4	Installation	3
1.5	Documentation	3
1.6	Finally...	4
2	Installing aws-auth-helper	5
2.1	Locating the software	5
2.2	Source Release Packages	5
2.3	Python Eggs	6
2.4	Final Words	6
3	Tutorial	7
4	API Reference	9
4.1	AWSArgumentParser	9
4.2	validate_creds	10
4.3	Credentials	10
4.4	Password generation	13
5	What's new in aws-auth-helper 1.5.0	15
5.1	Release: 1.5.5	15
5.2	Release: 1.5.4	15
5.3	Release: 1.5.2	15
5.4	Release: 1.5.1	16
5.5	Release: 1.5.0	16
5.6	Release: 1.4.0	16
6	Indices and tables	17

Helper library providing ArgumentParser and Credentials class for AWS authentication

A Python library for simplifying authentication to Amazon Web Services APIs.

1.1 Changes

For details on the latest updates and changes, see *What's new in aws-auth-helper 1.5.0*

1.2 License

This software is released under the GPLv2.

See the licence for full text.

1.3 Dependencies

- Python 2.7.10

1.4 Installation

See *Installing aws-auth-helper* for details.

1.5 Documentation

This library has comprehensive docstrings and a full set of project documentation (including tutorials):

- <http://aws-auth-helper.readthedocs.io/en/latest/>

1.6 Finally...

Share and enjoy!

Installing aws-auth-helper

aws-auth-helper is available in various packaged and non-packaged forms :

- source code repository access
- source release packages (tarball and zip formats)
- Python eggs

You can also build your own RPM packages, using bdist_rpm with setup.py available in the source tarball.

2.1 Locating the software

aws-auth-helper is available directly from the public git source code repository.

Details on how to check out the source code can be found here :

<http://github.com/drewsonne/aws-auth-helper/>

Official milestone releases can be found here :

<http://github.com/drewsonne/aws-auth-helper/downloads>

2.2 Source Release Packages

Download the latest release tarball/zip file and extract it to a temporary location or check out the source from the code hosting site into a local working copy directory.

Run the setup file in the root directory like this:

```
python setup.py install
```

This automatically places the required files in the `lib/site-packages` directory of the Python version you used to run the setup script, may be part of a virtualenv or similar.

2.3 Python Eggs

You can build and install eggs with aws-auth-helper using the `setup_egg.py` file provided in the source distribution.

All the usual commands are supported e.g.:

```
python setup_egg.py develop
python setup_egg.py bdist_egg
...
```

This requires that you install distribute or setuptools which is not part of the Python standard library.

See the following URL for details :-

- distribute - <http://guide.python-distribute.org/>
- setuptools (old) - <http://peak.telecommunity.com/DevCenter/setuptools>

Warning: setuptools is now very long in the tooth and full of bugs! Just use distribute, or pip instead.

Download and install the latest `easy_install` script and run the following command

```
easy_install aws-auth-helper
```

This will go to the Python Package Index and automatically find the appropriate version of aws-auth-helper for your Python setup.

Alternatively, you can use pip instead of easy_install.

Just download the latest version of pip from PyPI found here - <http://pypi.python.org/pypi/pip> and run the following command

```
pip install aws-auth-helper
```

2.4 Final Words

Always be sure you verify your downloads against the checksums on the code hosting site's download page!

CHAPTER 3

Tutorial

This tutorial will take you through a quick example of a normal `ArgumentParser` class, and then show you how to integrate the `AWSArgumentParser` into this.

First import the default `ArgumentParser`

```
>>> from argparse import ArgumentParser
```

First, let's create an argument parser for the rest of the options in our new utility.

```
>>> # Instantiate an argument parser, add an argument, and print the help text
>>> my_aws_app = ArgumentParser(description='Lists EC2 instances', prog='my_app')
>>> my_aws_app.add_argument('--name', required=True)
>>> my_aws_app.print_help()
usage: my_app [-h] --name NAME

Lists EC2 instances

optional arguments:
  -h, --help            show this help message and exit
  --name NAME
```

```
>>> my_aws_app.parse_args(args=['--name', 'Hello, World!'])
Namespace(name='Hello, World!')
```

Now that we have a parser for the arguments of our utility, we can add the `AWSArgumentParser`.

```
>>> from awsauthhelper import AWSArgumentParser
>>> aws_options = AWSArgumentParser(role_session_name='ec2_audit')
```

Note: You **must** set a `role_session_name` parameter in-case the user does not provide one on the cli.

Now let's recreate our app options, so that we can chain the `AWSArgumentParser`.

```
>>> my_aws_app = argparse.ArgumentParser(
>>>     prog='my_app',
>>>     description='Lists EC2 instances',
>>>     parents=[
>>>         aws_options
>>>     ]
>>> )
>>> my_aws_app.add_argument('--max-instances', type=int)
>>> my_aws_app.print_help()
usage: my_app [-h] [--aws-access-key-id AWS_ACCESS_KEY_ID]
              [--aws-secret-access-key AWS_SECRET_ACCESS_KEY]
              [--aws-session-token AWS_SESSION_TOKEN] [--region REGION]
              [--profile PROFILE] [--role ROLE] [--config-path CONFIG_PATH]
              [--credentials-path CREDENTIALS_PATH] [--auth-debug]
              [--role-session-name ROLE_SESSION_NAME]
              [--max-instances MAX_INSTANCES]

Lists EC2 instances

optional arguments:
  -h, --help            show this help message and exit
  --max-instances MAX_INSTANCES

AWS credentials:
  --aws-access-key-id AWS_ACCESS_KEY_ID
                        AWS access key
  --aws-secret-access-key AWS_SECRET_ACCESS_KEY
                        Access and secret key variables override credentials
                        stored in credential and config files
  --aws-session-token AWS_SESSION_TOKEN
                        A session token is only required if you are using
                        temporary security credentials.
  --region REGION      This variable overrides the default region of the in-
                        use profile, if set.
  --profile PROFILE    This can be the name of a profile stored in a
                        credential or config file, or default to use the
                        default profile.
  --role ROLE          Fully qualified role arn to assume
  --config-path CONFIG_PATH
                        Specify a custom location for ~/.aws/config
  --credentials-path CREDENTIALS_PATH
                        Specify a custom location for ~/.aws/credentials
  --auth-debug         Enter debug mode, which will print credentials and
                        then exist at `create_session`.
  --role-session-name ROLE_SESSION_NAME
                        If you have assigned a role, set a --role-session-name
```

Note: It is possible to use the `AWSArgumentParser` as your main `ArgumentParser` object, and like you would with a normal `ArgumentParser` object, but if you chain the `ArgumentParser` and `AWSArgumentParser`, you can segment your options in the help text, as you can see here. Furthermore, if you set the `AWSArgumentParser` as the parent, the `aws` options will be rendered at the end of the help.

4.1 AWSArgumentParser

This class provides a prepackaged set of cli options for AWS authentication.

CLI Option	Default	Description
<code>--aws-access-key-id</code>	<code>\$AWS_ACCESS_KEY_ID</code>	
<code>--aws-secret-access-key</code>	<code>\$AWS_SECRET_ACCESS_KEY</code>	
<code>--aws-session-token</code>	<code>\$AWS_SESSION_TOKEN</code>	
<code>--region</code>	<code>\$AWS_DEFAULT_REGION</code>	
<code>--profile</code>	<code>\$AWS_DEFAULT_PROFILE</code>	
<code>--role</code>		
<code>--config-path</code>	<code>\$AWS_CONFIG_FILE</code>	Custom path to an AWS config file
<code>--credentials-path</code>	<code>\$AWS_SHARED_CREDENTIALS_FILE</code>	Custom path to an AWS credentials path
<code>--auth-debug</code>		If this flag is enabled, execution of the application will stop when <code>create_session()</code> is called.

The `AWSArgumentParser` class takes all the arguments of a `argparser.ArgumentParser` class in addition to:

- `role_session_name` is a default value in case `--role_session_name` is not provided by the user.
- `region` is a default value in case `--region` is not provided by the user.
- `profile` is a default value in case `--profile` is not provided by the user.
- `enforce_auth_type` enforces the type of arguments which must be passed to this utility. Can be one of:

Argument	Description
keys	Both <code>aws_access_key_id</code> and <code>aws_secret_access_key</code> must be provided by the user.
keys_with_session	All of <code>aws_access_key_id</code> , <code>aws_secret_access_key</code> , and <code>aws_session_token</code> must be provided by the user.
profile	Only profile must be provided by the user.
profile_role	Both profile, and role must be provided by the user.
config	Only <code>config_path</code> must be provided by the user.
credentials	Only <code>credentials_path</code> must be provided by the user.

Like `argparse.ArgumentParser`, `AWSArgumentParser` allows chaining/inclusion of multiple `ArgumentParser` objects through the `list[ArgumentParser]: parents` constructor argument. The child `ArgumentParser` appears last in the list of options when `--help` is called, so it's best to add other `ArgumentParser` objects to `AWSArgumentParser`, rather than the reverse.

class `awsauthhelper.AWSArgumentParser` (*role_session_name*, *region=None*, *profile=None*, *enforce_auth_type=None*, ***kwargs*)

Helper Class containing a preset set of cli arguments for parsing into the `Credentials` object. If not explicitly set, arguments are read from the environment variables.

Create our arguments and determine if we need to enforce an auth method.

Parameters

- **role_session_name** (*str*) – Default name for the role session, in case a user does not provide one.
- **region** (*str*) – AWS Region
- **profile** (*str*) – Name of the profile in the AWS profile to use as the base configuration.
- **enforce_auth_type** (*str*) – The Authentication method can be locked to one of {'keys', 'keys_with_session', 'profile', 'profile_role', 'config', 'credentials'}
- **kwargs** (*dict*) –

Return `awsauthhelper.AWSArgumentParser`

4.2 validate_creds

Helper function validate your credential combinations

4.3 Credentials

The `Credentials` class allows us to encapsulate and hide all the aws auth operations, exposing three key methods:

- `has_role()`
- `assume_role()`
- `create_session()`

The arguments this class takes are the same format as `libawsauth.ArgumentParser()`, so the `Namespace` object returned from `argparse.ArgumentParser.parse_args()` can be wrapped in `vars(...)` and injected as *kwargs* into the `Credentials(...)` constructor.

```

>>> configs = aws_options.parse_args()
>>> credentials = awsauthhelper.Credentials(
...     **vars(configs)
... )

>>> if credentials.has_role():
>>>     credentials.assume_role()
>>> boto3_session = credentials.create_session()

>>> s3 = boto3_session().resource('s3')
>>> for bucket in s3.buckets.all():
>>>     print(bucket.name)

>>> for region in regions:
>>>     # The session object can be 're-authorized' across regions.
>>>     print(
...         boto3_session(region=region['RegionName']).client('ec2').describe_
↪instances()
...     )

```

```

class awsauthhelper.Credentials(region=None,                aws_secret_access_key=None,
                                aws_access_key_id=None,      aws_session_token=None,
                                profile=None, role=None, role_session_name=None, con-
                                fig_path=None, credentials_path=None, mfa_serial=None,
                                mfa_session_life=900, mfa_token=None, force_mfa=False,
                                auth_debug=False, **kwargs)

```

Encapsulates processing of AWS credentials.

Handle the assumption of roles, and creation of Session objects.

Parameters

- **region** (*str*) – AWS region
- **aws_secret_access_key** (*str*) – AWS_SECRET_ACCESS_KEY to use for the base credentials.
- **aws_access_key_id** (*str*) – AWS_ACCESS_KEY_ID to use for the base credentials.
- **aws_session_token** (*str*) – AWS_SESSION_TOKEN to use for the base credentials. Generally this should not be needed as roles are assumed through providing a role argument.
- **profile** (*str*) – Name of the profile in the AWS profile to use as the base configuration.
- **role** (*str*) – ARN of the AWS IAM Role to assume.
- **role_session_name** (*str*) – Custom name of the role session to override the default.
- **config_path** (*str*) – Custom path to the aws config file if it is not in a location botocore expects.
- **credentials_path** (*str*) – Custom path to the aws credentials file if it is not in a path botocore expects.
- **mfa_serial** (*str*) – Identification number of the MFA device. If you set this argument, your will be prompted for your MFA token.
- **mfa_session_life** (*str*) – The duration, in seconds, that the mfa credentials should remain valid.
- **mfa_token** (*str*) – MFA token to authentication to AWS with.

- **auth_debug** (*bool*) – Whether or not to print debug information. If True, `exit()` is throw at `create_session()`
- **kwargs** (*dict*) – catcher to allow arbitrary `**var(my_args.parse_args(...))` to be passed in. Arguments in `**kwargs` not used at all.

Return awsauthhelper.Credentials

assume_role()

Check if we have a role, and assume it if we do. Otherwise, raise exception.

Raises ValueError – If a role has not be specified.

Return awsauthhelper.Credentials Allow chaining.

assume_temp_session()

Retrieve some temporary credentials from AWS

Return awsauthhelper.Credentials Allow chaining.

authenticate_mfa()

Use the provided `mfa_serial`, the existing credentials, and get an mfa session token

Returns

create_session (*internal=False*)

DEPRECATED. Use `awsauthhelper.Credentials.get_session_generator()` instead.

Returns

freeze()

Take a snapshot of the credentials and remember them.

Return awsauthhelper.Credentials

get_session_generator (*internal=False*)

Return a callable which will generate a boto3 Session

Parameters internal (*bool*) – Whether or not this method was called from internal or external to the class

Return callable(region)

has_keys()

Do we have key credentials?

Return bool

has_mfa()

Have we been provided an `mfa_serial` to use?

Return bool

has_profile()

Do we have profile credentials?

Return bool

has_role()

Do we have a role to assume?

Return bool

has_session_keys()

Do we have temporal key credentials?

Return bool

reset()

Reset Credentials object back to original state, pre any role assumptions.

Return `awsauthhelper.Credentials`

use_as_global()

Set this object to use its current credentials as the global boto3 settings. If a role has been assumed, the assumed credentials will be used. If a role is set but has not been assumed, the base credentials will be used. WARNING: This will affect all calls made to boto3.

Return `awsauthhelper.Credentials`

using_role()

If we have a role and either a set of credentials or a profile, then we should assume the role.

Return `bool`

4.4 Password generation

`awsauthhelper.password.generate(password_policy)`

Builds a password based on the password policy provided `password_policy` should be an object with the attributes:

- **minimum_password_length** (*int*) – Minimum length of password. Maximum length of password will be the ceiling of 1.3 times this value.
- **require_symbols** (*bool*) – Make sure password contains `!@#$%^&*()_+-=[]{}|'.`
- **require_lowercase_characters** (*bool*) – Make sure password contains `abcdefghijklmnopqrstuvwxyz.`
- **require_uppercase_characters** (*bool*) – Make sure password contains `ABCDEFGHIJKLMNOPQRSTUVWXYZ.`
- **require_numbers** (*bool*) – Make sure password contains `0123456789.`

Parameters `password_policy` (*iam.AccountPasswordPolicy*) – boto password policy

Return `basestring` New password

What's new in aws-auth-helper 1.5.0

5.1 Release: 1.5.5

Date: 7 June 2017

5.1.1 Changes since 1.5.4

- Fixed a bug where region were not being exported when using the global session

5.2 Release: 1.5.4

Date: 7 June 2017

- Fixed a bug where profiles were not being exported when using the global session

5.3 Release: 1.5.2

Date: 7 March 2017

5.3.1 Changes since 1.5.1

- Added MFA functionality to `awsauthhelper.Credentials`. Special thanks to [Ahmed](<https://github.com/Ashex>) for testing!
- Added `aws-auth-helper` cli utility to provide an interface to test the library.
- Added `-shell-init` to `aws-auth-helper` to allow shell initialisation of AWS environment variables.

5.4 Release: 1.5.1

Date: 21 June 2016

5.4.1 Changes since 1.5.0

- Fixed a bug where the previous ‘README.rst’ was not found by setup.py, causing travis build to fail.

5.5 Release: 1.5.0

Date: 17 May 2016

5.5.1 Changes since 1.4.1

- **Added filtering for cli options. Specific types of authentication cli options can be turned on and off using** `AWSArgumentParser(..., enforce_auth_type=None, ...)`.

5.6 Release: 1.4.0

Date: 4 Sep 2015

5.6.1 Changes since 1.3.3

- Added method `Credentials.use_as_global()` to allow the use of credentials with calls directly in the boto3 namespace.
- Added more test cases
- Changed default, profile, region, and role_session name parameter names in `awsauthhelper.Credentials.__init__`

Specific bug fixes addressed in this release

FIXED 10: <https://github.com/drewsonne/aws-auth-helper/issues/10>

- `__init__.py` - Bad key

CHAPTER 6

Indices and tables

A

`assume_role()` (`awsauthhelper.Credentials` method), [12](#)
`assume_temp_session()` (`awsauthhelper.Credentials`
method), [12](#)
`authenticate_mfa()` (`awsauthhelper.Credentials` method),
[12](#)
`AWSArgumentParser` (class in `awsauthhelper`), [10](#)

C

`create_session()` (`awsauthhelper.Credentials` method), [12](#)
`Credentials` (class in `awsauthhelper`), [11](#)

F

`freeze()` (`awsauthhelper.Credentials` method), [12](#)

G

`generate()` (in module `awsauthhelper.password`), [13](#)
`get_session_generator()` (`awsauthhelper.Credentials`
method), [12](#)

H

`has_keys()` (`awsauthhelper.Credentials` method), [12](#)
`has_mfa()` (`awsauthhelper.Credentials` method), [12](#)
`has_profile()` (`awsauthhelper.Credentials` method), [12](#)
`has_role()` (`awsauthhelper.Credentials` method), [12](#)
`has_session_keys()` (`awsauthhelper.Credentials` method),
[12](#)

R

`reset()` (`awsauthhelper.Credentials` method), [12](#)

U

`use_as_global()` (`awsauthhelper.Credentials` method), [13](#)
`using_role()` (`awsauthhelper.Credentials` method), [13](#)