awslimitchecker Documentation

Release 0.2.1@0.2.1*

Jason Antman

December 02, 2015

Contents

1 Status **3**

2 What It Does **5**

3 Requirements **7**

4 Installation and Usage **9**

5 Getting Help and Asking Questions **11**

6 Changelog **13**

7 Contents **15**
- 7.1 Getting Started . . . 15
- 7.2 Command Line Usage . . . 17
- 7.3 Python Usage . . . 21
- 7.4 Required IAM Permissions . . . 25
- 7.5 Supported Limits . . . 26
- 7.6 Development . . . 30
- 7.7 Internals . . . 35
- 7.8 awslimitchecker . . . 36
- 7.9 Getting Help . . . 64

8 Indices and tables **65**
- 8.1 License . . . 65

Python Module Index **67**

Master: Develop: A script and python module to check your AWS service limits and usage using boto.

Users building out scalable services in Amazon AWS often run into AWS' service limits - often at the least convenient time (i.e. mid-deploy or when autoscaling fails). Amazon's Trusted Advisor can help this, but even the version that comes with Business and Enterprise support only monitors a small subset of AWS limits and only alerts *weekly*. awslimitchecker provides a command line script and reusable package that queries your current usage of AWS resources and compares it to limits (hard-coded AWS defaults that you can override, API-based limits where available, or data from Trusted Advisor where available), notifying you when you are approaching or at your limits.

CHAPTER 1

Status

This project is currently in very early development. At this time please consider it beta code and not fully tested in all situations; furthermore its API may be changing rapidly. I hope to have this stabilized soon.

CHAPTER 2

What It Does

- Check current AWS resource usage against AWS Service Limits
- Show and inspect current usage
- Override default Service Limits (for accounts with increased limits)
- Compare current usage to limits; return information about limits that exceed thresholds, and (CLI wrapper) exit non-0 if thresholds are exceeded
- Define custom thresholds per-limit
- where possible, pull current limits from Trusted Advisor API
- where possible, pull current limits from each service's API (for services that provide this information)
- Supports explicitly setting the AWS region
- Supports using STS to assume roles in other accounts, including using `external_id`.

CHAPTER 3

Requirements

- Python 2.6 through 3.4. Python 2.x is recommended, as boto (the AWS client library) currently has incomplete Python3 support. See the boto documentation for a list of AWS services that are Python3-compatible.
- Python VirtualEnv and `pip` (recommended installation method; your OS/distribution should have packages for these)
- boto >= 2.32.0

CHAPTER 4

Installation and Usage

See *Getting Started*.

CHAPTER 5

Getting Help and Asking Questions

See *Getting Help*.

CHAPTER 6

Changelog

See https://github.com/jantman/awslimitchecker/blob/develop/CHANGES.rst

CHAPTER 7

Contents

7.1 Getting Started

7.1.1 What It Does

- Check current AWS resource usage against AWS Service Limits
- Show and inspect current usage
- Override default Service Limits (for accounts with increased limits)
- Compare current usage to limits; return information about limits that exceed thresholds, and (CLI wrapper) exit non-0 if thresholds are exceeded
- Define custom thresholds per-limit
- Where possible, pull current limits from Trusted Advisor API
- Supports explicitly setting the AWS region
- Supports using STS to assume roles in other accounts, including using `external_id`.

7.1.2 Nomenclature

Service An AWS Service or Product, such as EC2, VPC, RDS or ElastiCache. More specifically, Services in AwsLimitChecker correspond to distinct APIs for AWS Services.

Limit An AWS-imposed maximum usage for a certain resource type in AWS. See AWS Service Limits. Limits are generally either account-wide or per-region. They have AWS global default values, but can be increased by AWS Support. "Limit" is also the term used within this documentation to describe *`AwsLimit`* objects, which describe a specific AWS Limit within this program.

Usage "Usage" refers to your current usage of a specific resource that has a limit. Usage values/amounts (some integer or floating point number, such as number of VPCs or GB of IOPS-provisioned storage) are represented by instances of the *`AwsLimitUsage`* class. Limits that are measured as a subset of some "parent" resource, such as "Subnets per VPC" or "Read Replicas per Master" have their usage tracked per parent resource, so you can easily determine which ones are problematic.

Threshold The point at which AwsLimitChecker will consider the current usage for a limit to be problematic. Global thresholds default to usage >= 80% of limit for "warning" severity, and usage >= 99% of limit for "critical" severity. Limits which have reached or exceeded their threshold will be reported separately for warning and critical (we generally consider "warning" to be something that will require human intervention in

the near future, and "critical" something that is an immediate problem, i.e. should block automated processes). The `awslimitchecker` command line wrapper can override the default global thresholds. The *`AwsLimitChecker`* class can both override global percentage thresholds, as well as specify per-limit thresholds as a percentage, a fixed usage value, or both.

7.1.3 Requirements

- Python 2.6 or 2.7 (boto currently has incomplete python3 support)
- Python VirtualEnv and `pip` (recommended installation method; your OS/distribution should have packages for these)
- boto

7.1.4 Installing

It's recommended that you install into a virtual environment (virtualenv / venv). See the virtualenv usage documentation for more details, but the gist is as follows (the virtualenv name, "limitchecker" here, can be whatever you want):

```
virtualenv limitchecker
source limitchecker/bin/activate
pip install awslimitchecker
```

7.1.5 Credentials

Aside from STS, awslimitchecker does nothing with AWS credentials, it leaves that to boto itself. You must either have your credentials configured in one of boto's supported config files, or set as environment variables. See boto config and this project's documentation for further information.

When using STS, you will need to specify the `-r` / `--region` option as well as the `-A` / `--sts-account-id` and `-R` / `--sts-account-role` options to specify the Account ID that you want to assume a role in, and the name of the role you want to assume. If an external ID is required, you can specify it with `-E` / `--external-id`.

7.1.6 Regions

To specify the region that `awslimitchecker` connects to, use the `-r` / `--region` command line option. At this time awslimitchecker can only connect to one region at a time; to check limits in multiple regions, simply run the script multiple times, once per region.

7.1.7 Required Permissions

You can view a sample IAM policy listing the permissions required for awslimitchecker to function properly either via the CLI client:

```
awslimitchecker --iam-policy
```

Or as a python dict:

```
from awslimitchecker.checker import AwsLimitChecker
c = AwsLimitChecker()
iam_policy = c.get_required_iam_policy()
```

You can also view the required permissions for the current version of awslimitchecker at *Required IAM Permissions*.

7.2 Command Line Usage

awslimitchecker ships with a command line script for use outside of Python environments. `awslimitchecker` is installed as a setuptools entry point, and will be available wherever you install the package (if you install in a virtual environment as recommended, it will be in the venv's `bin/` directory).

The command line script provides simple access to the most common features, though not full access to all configuration possibilities. In addition, when checking usage, the script will exit 0 of everything is OK, 1 if there are warnings, and 2 if there are critical thresholds exceeded (though the output is not currently suitable for direct use as a Nagios-compatible plugin).

```
(venv)$ awslimitchecker --help
usage: awslimitchecker [-h] [-S SERVICE] [-s] [-l] [--list-defaults]
                       [-L LIMIT] [-u] [--iam-policy] [-W WARNING_THRESHOLD]
                       [-C CRITICAL_THRESHOLD] [-A STS_ACCOUNT_ID]
                       [-R STS_ACCOUNT_ROLE] [-E EXTERNAL_ID] [-r REGION]
                       [--skip-ta] [--no-color] [-v] [-V]
Report on AWS service limits and usage via boto, optionally warn about any
services with usage nearing or exceeding their limits. For further help, see
<http://awslimitchecker.readthedocs.org/>
optional arguments:
  -h, --help            show this help message and exit
  -S SERVICE, --service SERVICE
                        perform action for only the specified service name;
                        see -s|--list-services for valid names
  -s, --list-services   print a list of all AWS service types that
                        awslimitchecker knows how to check
  -l, --list-limits     print all AWS effective limits in
                        "service_name/limit_name" format
  --list-defaults       print all AWS default limits in
                        "service_name/limit_name" format
  -L LIMIT, --limit LIMIT
                        override a single AWS limit, specified in
                        "service_name/limit_name=value" format; can be
                        specified multiple times.
  -u, --show-usage      find and print the current usage of all AWS services
                        with known limits
  --iam-policy          output a JSON serialized IAM Policy listing the
                        required permissions for awslimitchecker to run
                        correctly.
  -W WARNING_THRESHOLD, --warning-threshold WARNING_THRESHOLD
                        default warning threshold (percentage of limit);
                        default: 80
  -C CRITICAL_THRESHOLD, --critical-threshold CRITICAL_THRESHOLD
                        default critical threshold (percentage of limit);
                        default: 99
  -A STS_ACCOUNT_ID, --sts-account-id STS_ACCOUNT_ID
                        for use with STS, the Account ID of the destination
                        account (account to assume a role in)
  -R STS_ACCOUNT_ROLE, --sts-account-role STS_ACCOUNT_ROLE
                        for use with STS, the name of the IAM role to assume
  -E EXTERNAL_ID, --external-id EXTERNAL_ID
                        External ID to use when assuming a role via STS
  -r REGION, --region REGION
                        AWS region name to connect to; required for STS
  --skip-ta             do not attempt to pull *any* information on limits
                        from Trusted Advisor
  --no-color            do not colorize output
```

```
  -v, --verbose         verbose output. specify twice for debug-level output.
  -V, --version         print version number and exit.
awslimitchecker is AGPLv3-licensed Free Software. Anyone using this program,
even remotely over a network, is entitled to a copy of the source code. Use
`--version` for information on the source code location.
```

7.2.1 Examples

In the following examples, **output has been truncated** to simplify documentation. When running with all services enabled, `awslimitchecker` will provide *many* lines of output. `(...)` has been inserted in the output below to denote removed or truncated lines.

Listing Supported Services

View the AWS services currently supported by `awslimitchecker` with the `-s` or `--list-services` option.

```
(venv)$ awslimitchecker -s
AutoScaling
EBS
EC2
ELB
ElastiCache
RDS
VPC
```

Listing Default Limits

To show the hard-coded default limits, ignoring any limit overrides or Trusted Advisor data, run with `--list-defaults`:

```
(venv)$ awslimitchecker --list-defaults
AutoScaling/Auto Scaling groups                          20
AutoScaling/Launch configurations                        100
EBS/Active snapshots                                     10000
EBS/Active volumes                                       5000
EBS/General Purpose (SSD) volume storage (GiB)           20480
(...)
VPC/Rules per network ACL                                20
VPC/Subnets per VPC                                      200
VPC/VPCs                                                 5
```

Viewing Limits

View the limits that `awslimitchecker` currently knows how to check, and what the limit value is set as (if you specify limit overrides, they will be used instead of the default limit) by specifying the `-l` or `--list-limits` option. Limits followed by `(TA)` have been obtained from Trusted Advisor and limits followed by `(API)` have been obtained from the service's API.

```
(venv)$ awslimitchecker -l
AutoScaling/Auto Scaling groups                          500 (API)
AutoScaling/Launch configurations                        500 (API)
EBS/Active snapshots                                     13000 (TA)
EBS/Active volumes                                       5000 (TA)
```

```
EBS/General Purpose (SSD) volume storage (GiB)          163840 (TA)
(...)
VPC/Rules per network ACL                               20
VPC/Subnets per VPC                                     200
VPC/VPCs                                                20 (TA)
```

Disabling Trusted Advisor Checks

Using the `--skip-ta` option will disable attempting to query limit information from Trusted Advisor for all commands.

```
(venv)$ awslimitchecker -l --skip-ta
AutoScaling/Auto Scaling groups                         500 (API)
AutoScaling/Launch configurations                       500 (API)
EBS/Active snapshots                                    10000
EBS/Active volumes                                      5000
EBS/General Purpose (SSD) volume storage (GiB)          20480
(...)
VPC/Rules per network ACL                               20
VPC/Subnets per VPC                                     200
VPC/VPCs                                                5
```

Checking Usage

The `-u` or `--show-usage` options to `awslimitchecker` show the current usage for each limit that `awslimitchecker` knows about. It will connect to the AWS API and determine the current usage for each limit. In cases where limits are per-resource instead of account-wide (i.e. "Rules per VPC security group" or "Security groups per VPC"), the usage will be reported for each possible resource in `resource_id=value` format (i.e. for each VPC security group and each VPC, respectively, using their IDs).

```
(venv)$ awslimitchecker -u
AutoScaling/Auto Scaling groups                         349
AutoScaling/Launch configurations                       388
EBS/Active snapshots                                    11536
EBS/Active volumes                                      2614
EBS/General Purpose (SSD) volume storage (GiB)          26726
(...)
VPC/Rules per network ACL                               max: acl-b1ad1ad5=4 (acl-b1ad1ad5=4, acl-4bd9
VPC/Subnets per VPC                                     max: vpc-c89074a9=19 (vpc-1e5e3c7b=1, vpc-ae7
VPC/VPCs                                                8
```

Overriding Limits

In cases where you've been given a limit increase by AWS Support, you can override the default limits with custom ones. Currently, to do this from the command line, you must specify each limit that you want to override separately (the *`set_limit_overrides()`* Python method accepts a dict for easy bulk overrides of limits) using the `-L` or `--limit` options. Limits are specified in a `service_name/limit_name=value` format, and must be quoted if the limit name contains spaces.

For example, to override the limits of EC2's "EC2-Classic Elastic IPs" and "EC2-VPC Elastic IPs" from their defaults of 5, to 10 and 20, respectively:

```
(venv)$ awslimitchecker -L "AutoScaling/Auto Scaling groups"=321 --limit="AutoScaling/Launch configu
AutoScaling/Auto Scaling groups                         321
```

```
AutoScaling/Launch configurations                        456
EBS/Active snapshots                                     13000 (TA)
EBS/Active volumes                                       5000 (TA)
EBS/General Purpose (SSD) volume storage (GiB)           163840 (TA)
(...)
EC2/Elastic IP addresses (EIPs)                          40 (API)
(...)
VPC/Rules per network ACL                                20
VPC/Subnets per VPC                                      200
VPC/VPCs                                                 20 (TA)
```

This example simply sets the overrides, and then prints the limits for confirmation.

Check Limits Against Thresholds

The default mode of operation for `awslimitchecker` (when no other action-specific options are specified) is to check the usage of all known limits, compare them against the configured limit values, and then output a message and set an exit code depending on thresholds. The limit values used will be (in order of precedence) explicitly-set overrides, Trusted Advisor data, and hard-coded defaults.

Currently, the `awslimitchecker` command line script only supports global warning and critical thresholds, which default to 80% and 99% respectively. If any limit's usage is greater than or equal to 80% of its limit value, this will be included in the output and the program will exit with return code 1. If any limit's usage is greater than or equal to 99%, it will include that in the output and exit 2. When determining exit codes, critical takes priority over warning. The output will include the specifics of which limits exceeded the threshold, and for limits that are per-resource, the resource IDs.

The Python class allows setting thresholds per-limit as either a percentage, or an integer usage value, or both; this functionality is not currently present in the command line wrapper.

To check all limits against their thresholds (in this example, one limit has crossed the warning threshold only, and another has crossed the critical threshold):

```
(venv)$ awslimitchecker --no-color
EBS/Active snapshots                                     (limit 13000) WARNING: 11536
EC2/Running On-Demand c3.large instances                 (limit 20) WARNING: 17
EC2/Running On-Demand m3.2xlarge instances               (limit 20) CRITICAL: 29
EC2/Running On-Demand m3.medium instances                (limit 20) CRITICAL: 25
EC2/Running On-Demand m3.xlarge instances                (limit 20) WARNING: 16
(...)
EC2/VPC security groups per elastic network interface    (limit 5) CRITICAL: eni-2751546e=5 WARNING: e
RDS/DB parameter groups                                  (limit 50) WARNING: 42
RDS/Subnet Groups                                        (limit 20) CRITICAL: 84
```

Set Custom Thresholds

To set the warning threshold of 50% and a critical threshold of 75% when checking limits:

```
(venv)$ awslimitchecker -W 97 --critical=98 --no-color
EC2/Running On-Demand m3.2xlarge instances               (limit 20) CRITICAL: 29
EC2/Running On-Demand m3.medium instances                (limit 20) CRITICAL: 25
EC2/Running On-Demand t2.micro instances                 (limit 20) CRITICAL: 28
EC2/Running On-Demand t2.small instances                 (limit 20) CRITICAL: 57
EC2/Security groups per VPC                              (limit 100) CRITICAL: vpc-c89074a9=784
EC2/VPC security groups per elastic network interface    (limit 5) CRITICAL: eni-2751546e=5
RDS/Subnet Groups                                        (limit 20) CRITICAL: 84
```

Required IAM Policy

`awslimitchecker` can also provide the user with an IAM Policy listing the minimum permissions for it to perform all limit checks. This can be viewed with the `--iam-policy` option:

```
(venv)$ awslimitchecker --iam-policy
{
  "Statement": [
    {
      "Action": [
        "autoscaling:DescribeAccountLimits",
(...)
    }
  ],
  "Version": "2012-10-17"
}
```

For the current IAM Policy required by this version of awslimitchecker, see *IAM Policy*.

Connect to a Specific Region

To connect to a specific region (i.e. `us-west-2`), simply specify the region name with the `-r` or `--region` options:

```
(venv)$ awslimitchecker -r us-west-2
```

Assume a Role in Another Account with STS

To assume the "foobar" role in account 123456789012 in region us-west-1, specify the `-r` / `--region` option as well as the `-A` / `--sts-account-id` and `-R` / `--sts-account-role` options:

```
(venv)$ awslimitchecker -r us-west-1 -A 123456789012 -R foobar
```

If you also need to specify an `external_id` of "myid", you can do that with the `-E` / `--external-id` options:

```
(venv)$ awslimitchecker -r us-west-1 -A 123456789012 -R foobar -E myid
```

Please note that this assumes that you already have STS configured and working between your account and the 123456789012 destination account; see the documentation for further information.

7.3 Python Usage

The full feature set of awslimitchecker is available through the Python API. This page attempts to document some examples of usage, but the best resources are *runner*, the command line wrapper, and the API documentation.

7.3.1 Full Jenkins Example

A full example of a wrapper script with limit and threshold overrides, and a Jenkins job to run it, is available in the `docs/examples` directory of awslimitchecker.

See docs/examples/README.rst on GitHub.

7.3.2 Simple Examples

Many of these examples use `pprint` to make output a bit nicer.

Instantiating the Class

Here we import and instantiate the *`AwsLimitChecker`* class; note that we also setup Python's `logging` module, which is used by `awslimitchecker`. We also import `pprint` to make the output nicer.

```
>>> import logging
>>> logging.basicConfig()
>>> logger = logging.getLogger()
>>>
>>> from awslimitchecker.checker import AwsLimitChecker
>>> c = AwsLimitChecker()
```

Specifying a Region

To specify a region ("us-west-2" in this example), specify it as the `region` string parameter to the class constructor:

```
>>> import logging
>>> logging.basicConfig()
>>> logger = logging.getLogger()
>>>
>>> from awslimitchecker.checker import AwsLimitChecker
>>> c = AwsLimitChecker(region='us-west-2')
```

Assuming a Role with STS

To check limits for another account using a Role assumed via STS, specify the `region`, `account_id` and `account_role` parameters to the class constructor. If an external ID is needed, this can be specified by the `external_id` parameter. All are strings:

```
>>> import logging
>>> logging.basicConfig()
>>> logger = logging.getLogger()
>>>
>>> from awslimitchecker.checker import AwsLimitChecker
>>> c = AwsLimitChecker(
...     region='us-west-2',
...     account_id='012345678901',
...     account_role='myRoleName',
...     external_id='myid'
... )
```

Setting a Limit Override

Override EC2's "EC2-Classic Elastic IPs" limit from its default to 20, using *`set_limit_override()`*.

```
>>> c.set_limit_override('EC2', 'EC2-Classic Elastic IPs', 20)
```

Checking Thresholds

To check the current usage against limits, use *check_thresholds()*. The return value is a nested dict of all limits with current usage meeting or exceeding the configured thresholds. Keys are the AWS Service names (string), values are dicts of limit name (string) to *AwsLimit* instances representing the limit and its current usage.

```
>>> result = c.check_thresholds()
>>> pprint.pprint(result)
{'EC2': {'Magnetic volume storage (TiB)': <awslimitchecker.limit.AwsLimit object at 0x7f398db62750>,
         'Running On-Demand EC2 instances': <awslimitchecker.limit.AwsLimit object at 0x7f398db55910>
         'Running On-Demand m3.medium instances': <awslimitchecker.limit.AwsLimit object at 0x7f398db
         'Security groups per VPC': <awslimitchecker.limit.AwsLimit object at 0x7f398db62790>}}
```

Looking at one of the entries, its *get_warnings()* method tells us that the usage did not exceed its warning threshold:

```
>>> result['EC2']['Magnetic volume storage (TiB)'].get_warnings()
[]
```

But its *get_criticals()* method tells us that it did meet or exceed the critical threshold:

```
>>> result['EC2']['Magnetic volume storage (TiB)'].get_criticals()
[<awslimitchecker.limit.AwsLimitUsage object at 0x7f2074dfeed0>]
```

We can then inspect the *AwsLimitUsage* instance for more information about current usage that crossed the threshold:

In this particular case, there is no resource ID associated with the usage, because it is an aggregate (type-, rather than resource-specific) limit:

```
>>> result['EC2']['Magnetic volume storage (TiB)'].get_criticals()[0].resource_id
>>>
```

The usage is of the EC2 Volume resource type (where one exists, we use the CloudFormation Resource Type strings to identify resource types).

```
>>> result['EC2']['Magnetic volume storage (TiB)'].get_criticals()[0].aws_type
'AWS::EC2::Volume'
```

We can query the actual numeric usage value:

```
>>> pprint.pprint(result['EC2']['Magnetic volume storage (TiB)'].get_criticals()[0].get_value())
23.337
```

Or a string description of it:

```
>>> print(str(result['EC2']['Magnetic volume storage (TiB)'].get_criticals()[0]))
23.337
```

The "Security groups per VPC" limit also crossed thresholds, and we can see that it has one critical usage value:

```
>>> len(result['EC2']['Security groups per VPC'].get_warnings())
0
>>> len(result['EC2']['Security groups per VPC'].get_criticals())
1
```

As this limit is per-VPC, our string representation of the current usage includes the VPC ID that crossed the critical threshold:

```
>>> for usage in result['EC2']['Security groups per VPC'].get_criticals():
...     print(str(usage))
```

```
...
vpc-c300b9a6=100
```

Disabling Trusted Advisor

To disable querying Trusted Advisor for limit information, simply call *get_limits()* or *check_thresholds()* with `use_ta=False`:

```
>>> result = c.check_thresholds(use_ta=False)
```

7.3.3 Logging

awslimitchecker uses the python `logging` library for logging, with module-level loggers defined in each file. If you already have a root-level logger defined in your program and are using a simple configuration (i.e. `logging.basicConfig()`), awslimitchecker logs will be emitted at the same level as that which the root logger is configured.

Assuming you have a root-level logger defined and configured, and you only want to see awslimitchecker log messages of WARNING level and above, you can set the level of awslimitchecker's logger before instantiating the class:

```
alc_log = logging.getLogger('awslimitchecker')
alc_log.setLevel(logging.WARNING)
checker = AwsLimitChecker()
```

It's _highly_ recommended that you do not suppress log messages of WARNING or above, as these indicate situations where the checker may not present accurate or complete results.

If your application does not define a root-level logger, this becomes a bit more complicated. Assuming your application has a more complex configuration that uses a top-level logger 'myapp' with its own handlers defined, you can do something like the following. Note that this is highly specific to your logging setup:

```
# setup logging for awslimitchecker
alc_log = logging.getLogger('awslimitchecker')
# WARNING or higher should pass through
alc_log.setLevel(logging.WARNING)
# use myapp's handler(s)
for h in logging.getLogger('cm').handlers:
    alc_log.addHandler(h)
# instantiate the class
checker = AwsLimitChecker()
```

7.3.4 Advanced Examples

CI / Deployment Checks

This example checks usage, logs a message at `WARNING` level for any warning thresholds surpassed, and logs a message at `CRITICAL` level for any critical thresholds passed. If any critical thresholds were passed, it exits the script non-zero, i.e. to fail a CI or build job. In this example, we have multiple critical thresholds crossed.

```
>>> import logging
>>> logging.basicConfig()
>>> logger = logging.getLogger()
>>>
>>> from awslimitchecker.checker import AwsLimitChecker
```

```
>>> c = AwsLimitChecker()
>>> result = c.check_thresholds()
>>>
>>> have_critical = False
>>> for service, svc_limits in result.items():
...     for limit_name, limit in svc_limits.items():
...         for warn in limit.get_warnings():
...             logger.warning("{service} '{limit_name}' usage ({u}) exceeds "
...                            "warning threshold (limit={l})".format(
...                                service=service,
...                                limit_name=limit_name,
...                                u=str(warn),
...                                l=limit.get_limit(),
...                            )
...             )
...         for crit in limit.get_criticals():
...             have_critical = True
...             logger.critical("{service} '{limit_name}' usage ({u}) exceeds "
...                            "critical threshold (limit={l})".format(
...                                service=service,
...                                limit_name=limit_name,
...                                u=str(crit),
...                                l=limit.get_limit(),
...                            )
...             )
...
CRITICAL:root:EC2 'Magnetic volume storage (TiB)' usage (23.417) exceeds critical threshold (limit=20
CRITICAL:root:EC2 'Running On-Demand EC2 instances' usage (97) exceeds critical threshold (limit=20)
WARNING:root:EC2 'Security groups per VPC' usage (vpc-c300b9a6=96) exceeds warning threshold (limit=1
CRITICAL:root:EC2 'Running On-Demand m3.medium instances' usage (53) exceeds critical threshold (limi
CRITICAL:root:EC2 'EC2-Classic Elastic IPs' usage (5) exceeds critical threshold (limit=5)
>>> if have_critical:
...     raise SystemExit(1)
...
(awslimitchecker)$ echo $?
1
```

7.4 Required IAM Permissions

Below is the sample IAM policy from this version of awslimitchecker, listing the IAM permissions required for it to function correctly:

```
{
  "Statement": [
    {
      "Action": [
        "autoscaling:DescribeAccountLimits",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeLaunchConfigurations",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAddresses",
        "ec2:DescribeInstances",
        "ec2:DescribeInternetGateways",
        "ec2:DescribeNetworkAcls",
        "ec2:DescribeNetworkAcls",
        "ec2:DescribeNetworkInterfaces",
```

```
        "ec2:DescribeReservedInstances",
        "ec2:DescribeRouteTables",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSnapshots",
        "ec2:DescribeSnapshots",
        "ec2:DescribeSubnets",
        "ec2:DescribeSubnets",
        "ec2:DescribeVolumes",
        "ec2:DescribeVolumes",
        "ec2:DescribeVpcs",
        "ec2:DescribeVpcs",
        "elasticache:DescribeCacheClusters",
        "elasticache:DescribeCacheParameterGroups",
        "elasticache:DescribeCacheSecurityGroups",
        "elasticache:DescribeCacheSubnetGroups",
        "elasticloadbalancing:DescribeLoadBalancers",
        "rds:DescribeDBInstances",
        "rds:DescribeDBParameterGroups",
        "rds:DescribeDBSecurityGroups",
        "rds:DescribeDBSnapshots",
        "rds:DescribeDBSubnetGroups",
        "rds:DescribeEventSubscriptions",
        "rds:DescribeOptionGroups",
        "rds:DescribeReservedDBInstances",
        "support:*",
        "trustedadvisor:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ],
  "Version": "2012-10-17"
}
```

7.5 Supported Limits

7.5.1 Trusted Advisor Data

So long as the Service and Limit names used by Trusted Advisor (and returned in its API responses) exactly match those shown below, all limits listed in Trusted Advisor "Service Limit" checks should be automatically used by awslimitchecker. The following service limits have been confirmed as being updated from Trusted Advisor:

- AutoScaling
  - Auto Scaling groups
  - Launch configurations
- EBS
  - Active snapshots
  - Active volumes
  - General Purpose (SSD) volume storage (GiB)
  - Magnetic volume storage (GiB)

  - Provisioned IOPS
  - Provisioned IOPS (SSD) storage (GiB)
- EC2
  - Elastic IP addresses (EIPs)
  - VPC Elastic IP addresses (EIPs)
- ELB
  - Active load balancers
- RDS
  - DB instances
  - DB parameter groups
  - DB security groups
  - DB snapshots per user
  - Max auths per security group
  - Storage quota (GB)
- VPC
  - Internet gateways
  - VPCs

7.5.2 Limits Retrieved from Service APIs

The limits listed below can be retrieved directly from their Service's API; this information should be the most accurate, and is used with higher precedence than anything other than explicit limit overrides:

- AutoScaling
  - Auto Scaling groups
  - Launch configurations
- EC2
  - Elastic IP addresses (EIPs)
  - Running On-Demand EC2 instances
  - VPC Elastic IP addresses (EIPs)
  - VPC security groups per elastic network interface

7.5.3 Current Checks

The section below lists every limit that this version of awslimitchecker knows how to check, and its hard-coded default value (per AWS documentation). Limits marked with (TA) are comfirmed as being updated by Trusted Advisor.

AutoScaling

Limit	Default
Auto Scaling groups (TA) (API)	20
Launch configurations (TA) (API)	100

EBS

Limit	Default
Active snapshots (TA)	10000
Active volumes (TA)	5000
General Purpose (SSD) volume storage (GiB) (TA)	20480
Magnetic volume storage (GiB) (TA)	20480
Provisioned IOPS (SSD) storage (GiB) (TA)	20480
Provisioned IOPS (TA)	40000

EC2

Limit	Default
Elastic IP addresses (EIPs) (TA) (API)	5
Rules per VPC security group	50
Running On-Demand EC2 instances (API)	20
Running On-Demand c1.medium instances	20
Running On-Demand c1.xlarge instances	20
Running On-Demand c3.2xlarge instances	20
Running On-Demand c3.4xlarge instances	20
Running On-Demand c3.8xlarge instances	20
Running On-Demand c3.large instances	20
Running On-Demand c3.xlarge instances	20
Running On-Demand c4.2xlarge instances	20
Running On-Demand c4.4xlarge instances	10
Running On-Demand c4.8xlarge instances	5
Running On-Demand c4.large instances	20
Running On-Demand c4.xlarge instances	20
Running On-Demand cc2.8xlarge instances	20
Running On-Demand cg1.4xlarge instances	2
Running On-Demand cr1.8xlarge instances	2
Running On-Demand d2.2xlarge instances	20
Running On-Demand d2.4xlarge instances	10
Running On-Demand d2.8xlarge instances	5
Running On-Demand d2.xlarge instances	20
Running On-Demand g2.2xlarge instances	5
Running On-Demand g2.8xlarge instances	2
Running On-Demand hi1.4xlarge instances	2
Running On-Demand hs1.8xlarge instances	2
Running On-Demand i2.2xlarge instances	8
Running On-Demand i2.4xlarge instances	4
Running On-Demand i2.8xlarge instances	2
Running On-Demand i2.xlarge instances	8

Continued on next page

Table 7.1 – continued from previous page

Limit	Default
Running On-Demand m1.large instances	20
Running On-Demand m1.medium instances	20
Running On-Demand m1.small instances	20
Running On-Demand m1.xlarge instances	20
Running On-Demand m2.2xlarge instances	20
Running On-Demand m2.4xlarge instances	20
Running On-Demand m2.xlarge instances	20
Running On-Demand m3.2xlarge instances	20
Running On-Demand m3.large instances	20
Running On-Demand m3.medium instances	20
Running On-Demand m3.xlarge instances	20
Running On-Demand m4.2xlarge instances	20
Running On-Demand m4.4xlarge instances	20
Running On-Demand m4.8xlarge instances	20
Running On-Demand m4.large instances	20
Running On-Demand m4.xlarge instances	20
Running On-Demand r3.2xlarge instances	20
Running On-Demand r3.4xlarge instances	10
Running On-Demand r3.8xlarge instances	5
Running On-Demand r3.large instances	20
Running On-Demand r3.xlarge instances	20
Running On-Demand t1.micro instances	20
Running On-Demand t2.large instances	20
Running On-Demand t2.medium instances	20
Running On-Demand t2.micro instances	20
Running On-Demand t2.small instances	20
Security groups per VPC	100
VPC Elastic IP addresses (EIPs) (TA) (API)	5
VPC security groups per elastic network interface (API)	5

ELB

Limit	Default
Active load balancers (TA)	20
Listeners per load balancer	100

ElastiCache

Limit	Default
Clusters	50
Nodes	50
Nodes per Cluster	20
Parameter Groups	20
Security Groups	50
Subnet Groups	50

RDS

Limit	Default
DB instances (TA)	40
DB parameter groups (TA)	50
DB security groups (TA)	25
DB snapshots per user (TA)	50
Event Subscriptions	20
Max auths per security group (TA)	20
Option Groups	20
Read replicas per master	5
Reserved Instances	40
Storage quota (GB) (TA)	100000
Subnet Groups	20
Subnets per Subnet Group	20
VPC Security Groups	5

VPC

Limit	Default
Entries per route table	50
Internet gateways (TA)	5
Network ACLs per VPC	200
Route tables per VPC	200
Rules per network ACL	20
Subnets per VPC	200
VPCs (TA)	5

7.6 Development

7.6.1 Pull Requests

Please cut all pull requests against the "develop" branch. I'll do my best to merge them as quickly as possible. If they pass all unit tests and have 100% coverage, it'll certainly be easier. I work on this project only in my personal time, so I can't always get things merged as quickly as I'd like. That being said, I'm committed to doing my best, and please call me out on it if you feel like I'm not.

7.6.2 Installing for Development

To setup awslimitchecker for development:

1. Fork the awslimitchecker repository on GitHub
2. Create a *virtualenv* to run the code in:

```
$ virtualenv awslimitchecker
$ cd awslimitchecker
$ source bin/activate
```

3. Install your fork in the virtualenv as an editable git clone

```
$ pip install -e git+git@github.com:YOUR_NAME/awslimitchecker.git#egg=awslimitchecker
$ cd src/awslimitchecker
```

4. Check out a new git branch. If you're working on a GitHub issue you opened, your branch should be called "issues/N" where N is the issue number.

7.6.3 Guidelines

- pep8 compliant with some exceptions (see pytest.ini)
- 100% test coverage with pytest (with valid tests)
- each *_AwsService* subclass should only connect to boto once, and should save the connection as `self.conn`. They *must not* connect in the class constructor.
- All modules should have (and use) module-level loggers.
- See the section on the AGPL license below.
- **Commit messages** should be meaningful, and reference the Issue number if you're working on a GitHub issue (i.e. "issue #x - <message>"). Please refrain from using the "fixes #x" notation unless you are *sure* that the the issue is fixed in that commit.
- Unlike many F/OSS projects on GitHub, there is **no reason to squash your commits**; this just loses valuable history and insight into the development process, which could prove valuable if a bug is introduced by your work. Until GitHub fixes this, we'll live with a potentially messy git log in order to keep the history.

7.6.4 Adding New Limits and Checks to Existing Services

First, note that all calls to AWS APIs should be handled through *boto_query_wrapper()*, which handles retries (with exponential backoff) when API request rate limits are hit, and also handles paginated responses if they're not handled by boto.

Second, note that queries which may be paginated **and return a dict** instead of a ResultSet object must have the proper parameters for *_paginate_dict()* passed in to *boto_query_wrapper()*.

1. Add a new *AwsLimit* instance to the return value of the Service class's *get_limits()* method.
2. In the Service class's *find_usage()* method (or a method called by that, in the case of large or complex services), get the usage information via *boto* and pass it to the appropriate AwsLimit object via its *_add_current_usage()* method. For anything more than trivial services (those with only 2-3 limits), `find_usage()` should be broken into multiple methods, generally one per AWS API call.
3. If the service has an API call that retrieves current limit values, and its results include your new limit, ensure that this value is updated in the limit via its *_set_api_limit()* method. This should be done in the Service class's `_update_limits_from_api()` method.
4. Ensure complete test coverage for the above.

7.6.5 Adding New Services

All Services are sublcasses of *_AwsService* using the `abc` module.

First, note that all calls to AWS APIs should be handled through *boto_query_wrapper()*, which handles retries (with exponential backoff) when API request rate limits are hit, and also handles paginated responses if they're not handled by boto.

1. The new service name should be in CamelCase, preferably one word (if not one word, it should be underscore-separated). In `awslimitchecker/services`, use the `addservice` script; this will create a templated service class in the current directory, and create a templated (but far from complete) unit test file in `awslimitchecker/tests/services`:

```
./addservice ServiceName
```

2. Find all "TODO" comments in the newly-created files; these have instructions on things to change for new services. Add yourself to the Authors section in the header if desired.
3. Add an import line for the new service in `awslimitchecker/services/__init__.py`.
4. Ensure that the *`connect()`* method is properly defined; if `self.conn` is not None, then it should return None. If `self.region` is None, it should set `self.conn` to the return value of the appropriate `boto.connect_*()` method for the service, specifically the connected connection class for the service. Otherwise, it should call `self.connect_via()` (*`connect_via()`*) passing in the service's `connect_to_region()` function as the argument. This is done to centralize region and STS connection logic in *`_AwsService`*.
5. Write at least high-level tests; TDD is greatly preferred.
6. Implement all abstract methods from *`_AwsService`* and any other methods you need; small, easily-testable methods are preferred. Ensure all methods have full documentation. For simple services, you need only to search for "TODO" in the new service class you created (#1). See *Adding New Limits* for further information.
7. If your service has an API action to retrieve limit/quota information (i.e. `DescribeAccountAttributes` for EC2 and RDS), ensure that the service class has an `_update_limits_from_api()` method which makes this API call and updates each relevant AwsLimit via its *`_set_api_limit()`* method.
8. Test your code; 100% test coverage is expected, and mocks should be using `autospec` or `spec_set`.
9. Ensure the *`required_iam_permissions()`* method of your new class returns a list of all IAM permissions required for it to work.
10. Write integration tests. (currently not implemented; see issue #21)
11. Run all tox jobs, or at least one python version, docs and coverage.
12. Commit the updated documentation to the repository.
13. As there is no programmatic way to validate IAM policies, once you are done writing your service, grab the output of `awslimitchecker --iam-policy`, login to your AWS account, and navigate to the IAM page. Click through to create a new policy, paste the output of the `--iam-policy` command, and click the "Validate Policy" button. Correct any errors that occur; for more information, see the AWS IAM docs on Using Policy Validator. It would also be a good idea to run any policy changes through the Policy Simulator.
14. Submit your pull request.

7.6.6 Trusted Advisor Checks

So long as the `Service` and `Limit` name strings returned by the Trusted Advisor (Support) API exactly match how they are set on the corresponding *`_AwsService`* and *`AwsLimit`* objects, no code changes are needed to support new limit checks from TA.

For further information, see *Internals / Trusted Advisor*.

7.6.7 Unit Testing

Testing is done via pytest, driven by tox.

- testing is as simple as:
  - `pip install tox`
  - `tox`
- If you want to see code coverage: `tox -e cov`
  - this produces two coverage reports - a summary on STDOUT and a full report in the `htmlcov/` directory
- If you want to pass additional arguments to pytest, add them to the tox command line after "–". i.e., for verbose pytext output on py27 tests: `tox -e py27 -- -v`

Note that while boto currently doesn't have python3 support, we still run tests against py3 to ensure that this package is ready for it when boto is.

7.6.8 Integration Testing

currently not implemented; see issue #21

7.6.9 Building Docs

Much like the test suite, documentation is build using tox:

```
$ tox -e docs
```

Output will be in the `docs/build/html` directory under the project root.

7.6.10 AGPL License

awslimitchecker is licensed under the GNU Affero General Public License, version 3 or later.

Pursuant to Sections 5(b) and 13 of the license, all users of awslimitchecker - including those interacting with it remotely over a network - have a right to obtain the exact, unmodified running source code. We have done as much as possible to make this transparent to developers, with no additional work needed. See the guidelines below for information.

- If you're simply *running* awslimitchecker via the command line, there's nothing to worry about; just use it like any other software.
- If you're using awslimitchecker in your own software in a way that allows users to interact with it over the network (i.e. in your deployment or monitoring systems), but not modifying it, you also don't need to do anything special; awslimitchecker will log a WARNING-level message indicating where the source code of the currently-running version can be obtained. So long as you've installed awslimitchecker via Python's packaging system (i.e. with *pip*), its current version and source will be automatically detected. This suffices for the AGPL source code offer provision, so long as it's displayed to users and the currently-running source is unmodified.
- If you wish to modify the source code of awslimitchecker, you need to do is ensure that *`_get_version_info()`* always returns correct and accutate information (a publicly-accessible URL to the exact version of the running source code, and a version number). If you install your modified version directly from an editable (i.e. `pip install -e`), publicly-accessible Git repository, and ensure that changes are available in the repository before they are present in the code running for your users, this should be automatically detected by awslimitchecker and the correct URL provided. It is strongly recommended that any such repository is a fork of the project's original GitHub repository. It is solely your responsibility to ensure that the URL and version information presented to users is accurate and reflects source code identical to what is running.

- If you're distributing awslimitchecker with modifications or as part of your own software (as opposed to simply an editable requirement that gets installed with pip), please read the license and ensure that you comply with its terms.
- If you are running awslimitchecker as part of a hosted service that users somehow interact with, please ensure that the source code URL and version is correct and visible in the output given to users.

7.6.11 Handling Issues and PRs

All PRs and new work should be based off of the `develop` branch.

PRs can be merged if they look good, and `CHANGES.rst` updated after the merge.

For issues:

1. Cut a `issues/number` branch off of `develop`.
2. Work the issue, come up with a fix. Commit early and often, and mention "issue #x - <message>" in your commit messages.
3. When you believe you have a working fix, build docs (`tox -e docs`) and push to origin. Ensure all Travis tests pass.
4. Ensure that coverage has increased or stayed the same.
5. Update `CHANGES.rst` for the fix; commit this with a message like "fixes #x - <message>" and push to origin.
6. Open a new pull request **against the develop branch** for this change; once all tests pass, merge it to develop.
7. Assign the "unreleased fix" label to the issue. It should be closed automatically when develop is merged to master for a release, but this lets us track which issues have unreleased fixes.

7.6.12 Release Checklist

1. Open an issue for the release; cut a branch off `develop` for that issue.
2. Build docs (`tox -e docs`) and ensure they're current; commit any changes.
3. Ensure that Travis tests are passing in all environments. If there were any changes to `awslimitchecker.versioncheck`, manually run the `-versioncheck` tox environments (these are problematic in Travis and with PRs).
4. Ensure that test coverage is no less than the last release (ideally, 100%).
5. Create or update an actual IAM user with the policy from `awslimitchecker --iam-policy`; run the command line wrapper and ensure that the policy works and contains all needed permissions.
6. Build docs for the branch (locally) and ensure they look correct.
7. Increment the version number in awslimitchecker/version.py and add version and release date to CHANGES.rst. Ensure that there are CHANGES.rst entries for all major changes since the last release. Mention the issue in the commit for this, and push to GitHub.
8. Confirm that README.rst renders correctly on GitHub.
9. Upload package to testpypi, confirm that README.rst renders correctly.
    - Make sure your ~/.pypirc file is correct
    - `python setup.py register -r https://testpypi.python.org/pypi`
    - `python setup.py sdist upload -r https://testpypi.python.org/pypi`

    - Check that the README renders at https://testpypi.python.org/pypi/awslimitchecker
10. Create a pull request for the release to be merge into master. Upon successful Travis build, merge it.
11. Tag the release in Git, push tag to GitHub:
    - tag the release. for now the message is quite simple: `git tag -a vX.Y.Z -m 'X.Y.Z released YYYY-MM-DD'`
    - push the tag to GitHub: `git push origin vX.Y.Z`
12. Upload package to live pypi:
    - `python setup.py sdist upload`
13. make sure any GH issues fixed in the release were closed.
14. merge master back into develop
15. Blog, tweet, etc. about the new version.

7.7 Internals

7.7.1 Overall Program Flow

AwsLimitChecker provides the full and only public interface to this project; it's used by the `awslimitchecker` command line script (entry point to *runner*) and should be the only portion directly used by external code.

Each AWS Service is represented by a subclass of the *_AwsService* abstract base class; these Service Classes are responsible for knowing which limits exist for the service they represent, what the default values for these limits are, querying current limits from the service's API (if supported), and how to check the current usage via the AWS API (via `boto`). When the Service Classes are instantiated, they build a dict of all of their limits, correlating a string key (the "limit name") with an *AwsLimit* object. The Service Class constructors *must not* make any network connections; connections are created lazily as needed and stored as a class attribute. This allows us to inspect the services, limits and default limit values without ever connecting to AWS (this is also used to generate the *Supported Limits* documentation automatically).

All calls to the AWS APIs should be made through *boto_query_wrapper()*. This function encapsulates both retrying queries with an exponential backoff when queries are throttled due to your account hitting the request rate limit (via *invoke_with_throttling_retries()*) and automatically paginating query responses that aren't automatically handled by boto.

When *AwsLimitChecker* is instantiated, it imports *services* which in turn creates instances of all `awslimitchecker.services.*` classes and adds them to a dict mapping the string Service Name to the Service Class instance. These instances are used for all interaction with the services.

So, once an instance of *AwsLimitChecker* is created, we should have instant access to the services and limits without any connection to AWS. This is utilized by the `--list-services` and `--list-defaults` options for the *command line client*.

7.7.2 Trusted Advisor

When *AwsLimitChecker* is initialized, it also initializes an instance of *TrustedAdvisor*. In *get_limits()*, *find_usage()* and *check_thresholds()*, when called with `use_ta == True` (the default), *update_limits()* is called on the TrustedAdvisor instance.

update_limits() polls Trusted Advisor data is from the Support API via *_poll()*; this will retrieve the limits for all "flaggedResources" items in the `Service Limits` Trusted Advisor check result for the current AWS

account. It then calls *_update_services()*, passing in the Trusted Advisor check results and the dict of *_AwsService* objects it was called with (from *AwsLimitChecker*).

_update_services() iterates over the Services in the Trusted Advisor check result and attempts to find a matching *_AwsService* (by string service name) in the dict passed in from *AwsLimitChecker*. If a match is found, it iterates over all limits for that service in the TA result and attempts to call the `Service`'s *_set_ta_limit()* method. If a matching Service is not found, or if `_set_ta_limit` raises a ValueError (matching Limit not found for that Service), an error is logged.

Using this methodology, no additional code is needed to support new/additional Trusted Advisor limit checks; *so long as* the Service and Limit name strings match between the Trusted Advisor API response and their corresponding *_AwsService* and *AwsLimit* instances, the TA limits will be automatically added to the corresponding `AwsLimit`.

7.7.3 Service API Limit Information

Some services provide API calls to retrieve at least some of the current limits, such as the `DescribeAccountAttributes` API calls for RDS and EC2. Services that support such calls should make them in a `_update_limits_from_api()` method, which will be automatically called from *get_limits()*. The `_update_limits_from_api()` method should make the API call, and then update all relevant limits via the *AwsLimit* class's *_set_api_limit()* method.

7.7.4 Limit Value Precedence

The value used for a limit is the first match in the following list:

1. Limit Override (set at runtime)
2. API Limit
3. Trusted Advisor
4. Hard-coded default

7.8 awslimitchecker

7.8.1 awslimitchecker package

Subpackages

awslimitchecker.services package

Submodules

awslimitchecker.services.autoscaling module

class `awslimitchecker.services.autoscaling.`**`_AutoscalingService`**(*warning_threshold*, *critical_threshold*, *account_id=None*, *account_role=None*, *region=None*, *external_id=None*)

Bases: *awslimitchecker.services.base._AwsService*

Describes an AWS service and its limits, and provides methods to query current utilization.

Constructors of _AwsService subclasses *must not* make any external connections; these must be made lazily as needed in other methods. _AwsService subclasses should be usable without any external network connections.

Parameters

- **warning_threshold** (*int*) – the default warning threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **critical_threshold** (*int*) – the default critical threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **account_id** (*str*) – AWS Account ID (12-digit string, currently numeric) for the account to connect to (destination) via STS
- **account_role** (*str*) – the name of an IAM Role (in the destination account) to assume
- **region** (*str*) – AWS region name to connect to
- **external_id** (*str*) – (optional) the External ID string to use when assuming a role via STS.

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.autoscaling'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 25

_abc_registry = <_weakrefset.WeakSet object>

_update_limits_from_api()
Query EC2's DescribeAccountAttributes API action, and update limits with the quotas returned. Updates `self.limits`.

connect()
Connect to API if not already connected; set self.conn.

find_usage()
Determine the current usage for each limit of this service, and update corresponding Limit via *_add_current_usage()*.

get_limits()
Return all known limits for this service, as a dict of their names to *AwsLimit* objects.

Returns dict of limit names to *AwsLimit* objects

Return type dict

required_iam_permissions()
Return a list of IAM Actions required for this Service to function properly. All Actions will be shown with an Effect of "Allow" and a Resource of "*".

Returns list of IAM Action strings

Return type list

service_name = 'AutoScaling'

awslimitchecker.services.base module

class awslimitchecker.services.base.**_AwsService**(*warning_threshold*, *critical_threshold*, *account_id=None*, *account_role=None*, *region=None*, *external_id=None*)

Bases: *awslimitchecker.connectable.Connectable*

Describes an AWS service and its limits, and provides methods to query current utilization.

Constructors of _AwsService subclasses *must not* make any external connections; these must be made lazily as needed in other methods. _AwsService subclasses should be usable without any external network connections.

Parameters

- **warning_threshold** (*int*) – the default warning threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **critical_threshold** (*int*) – the default critical threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **account_id** (*str*) – AWS Account ID (12-digit string, currently numeric) for the account to connect to (destination) via STS
- **account_role** (*str*) – the name of an IAM Role (in the destination account) to assume
- **region** (*str*) – AWS region name to connect to
- **external_id** (*str*) – (optional) the External ID string to use when assuming a role via STS.

__abstractmethods__ = frozenset(['get_limits', 'find_usage', 'required_iam_permissions', 'connect'])

__init__(*warning_threshold*, *critical_threshold*, *account_id=None*, *account_role=None*, *region=None*, *external_id=None*)

Describes an AWS service and its limits, and provides methods to query current utilization.

Constructors of _AwsService subclasses *must not* make any external connections; these must be made lazily as needed in other methods. _AwsService subclasses should be usable without any external network connections.

Parameters

- **warning_threshold** (*int*) – the default warning threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **critical_threshold** (*int*) – the default critical threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **account_id** (*str*) – AWS Account ID (12-digit string, currently numeric) for the account to connect to (destination) via STS
- **account_role** (*str*) – the name of an IAM Role (in the destination account) to assume
- **region** (*str*) – AWS region name to connect to
- **external_id** (*str*) – (optional) the External ID string to use when assuming a role via STS.

__metaclass__

alias of ABCMeta

__module__ = 'awslimitchecker.services.base'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 25

_abc_registry = <_weakrefset.WeakSet object>

_set_ta_limit(*limit_name*, *value*)
Set the value for the limit as reported by Trusted Advisor, for the specified limit.

This method should only be called by *TrustedAdvisor*.

Parameters

- **limit_name** (*string*) – the name of the limit to override the value for
- **value** (*int*) – the Trusted Advisor limit value

Raises ValueError if limit_name is not known to this service

check_thresholds()
Checks current usage against configured thresholds for all limits for this service.

Returns a dict of limit name to *AwsLimit* instance for all limits that crossed one or more of their thresholds.

Return type dict of *AwsLimit*

connect()
If not already done, establish a connection to the relevant AWS service and save as `self.conn`. If `self.region` is defined, call `self.connect_via()` (*connect_via()*) passing the appripriate boto `connect_to_region()` function as the argument, else call the boto.connect_SERVICE_NAME() method directly.

find_usage()
Determine the current usage for each limit of this service, and update the `current_usage` property of each corresponding *AwsLimit* instance.

This method MUST set `self._have_usage = True`. This method MUST make all API calls through *boto_query_wrapper()*.

get_limits()
Return all known limits for this service, as a dict of their names to *AwsLimit* objects.

All limits must have `self.warning_threshold` and `self.critical_threshold` passed into them.

Returns dict of limit names to *AwsLimit* objects

Return type dict

required_iam_permissions()
Return a list of IAM Actions required for this Service to function properly. All Actions will be shown with an Effect of "Allow" and a Resource of "*".

Returns list of IAM Action strings

Return type list

service_name = 'baseclass'

set_limit_override(*limit_name*, *value*, *override_ta=True*)
Set a new limit `value` for the specified limit, overriding the default. If `override_ta` is True, also use this value instead of any found by Trusted Advisor. This method simply passes the data through to the *set_limit_override()* method of the underlying *AwsLimit* instance.

Parameters

- **limit_name** (*string*) – the name of the limit to override the value for

- **value** (*int*) – the new value to set for the limit
- **override_ta** (*bool*) – whether or not to also override Trusted Advisor information

Raises ValueError if limit_name is not known to this service

set_threshold_override(*limit_name*, *warn_percent=None*, *warn_count=None*, *crit_percent=None*, *crit_count=None*)

Override the default warning and critical thresholds used to evaluate the specified limit's usage. Thresholds can be specified as a percentage of the limit, or as a usage count, or both.

Parameters

- **warn_percent** (*int*) – new warning threshold, percentage used
- **warn_count** (*int*) – new warning threshold, actual count/number
- **crit_percent** (*int*) – new critical threshold, percentage used
- **crit_count** (*int*) – new critical threshold, actual count/number

awslimitchecker.services.ebs module

class awslimitchecker.services.ebs.**_EbsService**(*warning_threshold*, *critical_threshold*, *account_id=None*, *account_role=None*, *region=None*, *external_id=None*)

Bases: *awslimitchecker.services.base._AwsService*

Describes an AWS service and its limits, and provides methods to query current utilization.

Constructors of _AwsService subclasses *must not* make any external connections; these must be made lazily as needed in other methods. _AwsService subclasses should be usable without any external network connections.

Parameters

- **warning_threshold** (*int*) – the default warning threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **critical_threshold** (*int*) – the default critical threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **account_id** (*str*) – AWS Account ID (12-digit string, currently numeric) for the account to connect to (destination) via STS
- **account_role** (*str*) – the name of an IAM Role (in the destination account) to assume
- **region** (*str*) – AWS region name to connect to
- **external_id** (*str*) – (optional) the External ID string to use when assuming a role via STS.

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.ebs'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 25

_abc_registry = <_weakrefset.WeakSet object>

_find_usage_ebs()

calculate usage for all EBS limits and update Limits

_find_usage_snapshots()
find snapshot usage

_get_limits_ebs()
Return a dict of EBS-related limits only. This method should only be used internally by :py:meth:~.get_limits‘.

Return type dict

connect()
Connect to API if not already connected; set self.conn.

find_usage()
Determine the current usage for each limit of this service, and update corresponding Limit via *_add_current_usage()*.

get_limits()
Return all known limits for this service, as a dict of their names to *AwsLimit* objects.

Returns dict of limit names to *AwsLimit* objects

Return type dict

required_iam_permissions()
Return a list of IAM Actions required for this Service to function properly. All Actions will be shown with an Effect of “Allow” and a Resource of “*”.

Returns list of IAM Action strings

Return type list

service_name = 'EBS'

awslimitchecker.services.ec2 module

class awslimitchecker.services.ec2.**_Ec2Service**(*warning_threshold*, *critical_threshold*, *account_id=None*, *account_role=None*, *region=None*, *external_id=None*)
Bases: *awslimitchecker.services.base._AwsService*

Describes an AWS service and its limits, and provides methods to query current utilization.

Constructors of _AwsService subclasses *must not* make any external connections; these must be made lazily as needed in other methods. _AwsService subclasses should be usable without any external network connections.

Parameters

- **warning_threshold** (*int*) – the default warning threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **critical_threshold** (*int*) – the default critical threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **account_id** (*str*) – AWS Account ID (12-digit string, currently numeric) for the account to connect to (destination) via STS
- **account_role** (*str*) – the name of an IAM Role (in the destination account) to assume
- **region** (*str*) – AWS region name to connect to
- **external_id** (*str*) – (optional) the External ID string to use when assuming a role via STS.

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.ec2'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 25

_abc_registry = <_weakrefset.WeakSet object>

_find_usage_instances()
calculate On-Demand instance usage for all types and update Limits

_find_usage_networking_eips()

_find_usage_networking_eni_sg()

_find_usage_networking_sgs()
calculate usage for VPC-related things

_get_limits_instances()
Return a dict of limits for EC2 instances only. This method should only be used internally by :py:meth:~.get_limits'.

Return type dict

_get_limits_networking()
Return a dict of VPC-related limits only. This method should only be used internally by :py:meth:~.get_limits'.

Return type dict

_get_reserved_instance_count()
For each availability zone, get the count of current instance reservations of each instance type. Return as a nested dict of AZ name to dict of instance type to reservation count.

Return type dict

_instance_types()
Return a list of all known EC2 instance types

Returns list of all valid known EC2 instance types

Return type list

_instance_usage()
Find counts of currently-running EC2 Instances (On-Demand or Reserved) by placement (Availability Zone) and instance type (size). Return as a nested dict of AZ name to dict of instance type to count.

Return type dict

_update_limits_from_api()
Query EC2's DescribeAccountAttributes API action, and update limits with the quotas returned. Updates `self.limits`.

connect()
Connect to API if not already connected; set self.conn.

find_usage()
Determine the current usage for each limit of this service, and update corresponding Limit via *_add_current_usage()*.

get_limits()
Return all known limits for this service, as a dict of their names to *AwsLimit* objects.

Returns dict of limit names to *AwsLimit* objects

Return type dict

required_iam_permissions()
Return a list of IAM Actions required for this Service to function properly. All Actions will be shown with an Effect of "Allow" and a Resource of "*".

Returns list of IAM Action strings

Return type list

service_name = 'EC2'

awslimitchecker.services.elasticache module

class awslimitchecker.services.elasticache.**_ElastiCacheService**(*warning_threshold*, *critical_threshold*, *account_id=None*, *account_role=None*, *region=None*, *external_id=None*)

Bases: *awslimitchecker.services.base._AwsService*

Describes an AWS service and its limits, and provides methods to query current utilization.

Constructors of _AwsService subclasses *must not* make any external connections; these must be made lazily as needed in other methods. _AwsService subclasses should be usable without any external network connections.

Parameters

- **warning_threshold** (*int*) – the default warning threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **critical_threshold** (*int*) – the default critical threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **account_id** (*str*) – AWS Account ID (12-digit string, currently numeric) for the account to connect to (destination) via STS
- **account_role** (*str*) – the name of an IAM Role (in the destination account) to assume
- **region** (*str*) – AWS region name to connect to
- **external_id** (*str*) – (optional) the External ID string to use when assuming a role via STS.

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.elasticache'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 25

_abc_registry = <_weakrefset.WeakSet object>

_find_usage_nodes()
find usage for cache nodes

_find_usage_parameter_groups()
find usage for elasticache parameter groups

_find_usage_security_groups()
find usage for elasticache security groups

_find_usage_subnet_groups()
find usage for elasticache subnet groups

connect()
Connect to API if not already connected; set self.conn.

find_usage()
Determine the current usage for each limit of this service, and update corresponding Limit via *_add_current_usage()*.

get_limits()
Return all known limits for this service, as a dict of their names to *AwsLimit* objects.

Returns dict of limit names to *AwsLimit* objects

Return type dict

required_iam_permissions()
Return a list of IAM Actions required for this Service to function properly. All Actions will be shown with an Effect of "Allow" and a Resource of "*".

Returns list of IAM Action strings

Return type list

service_name = 'ElastiCache'

awslimitchecker.services.elb module

class awslimitchecker.services.elb.**_ElbService**(*warning_threshold*, *critical_threshold*, *account_id=None*, *account_role=None*, *region=None*, *external_id=None*)
Bases: *awslimitchecker.services.base._AwsService*

Describes an AWS service and its limits, and provides methods to query current utilization.

Constructors of _AwsService subclasses *must not* make any external connections; these must be made lazily as needed in other methods. _AwsService subclasses should be usable without any external network connections.

Parameters

- **warning_threshold** (*int*) – the default warning threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **critical_threshold** (*int*) – the default critical threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **account_id** (*str*) – AWS Account ID (12-digit string, currently numeric) for the account to connect to (destination) via STS
- **account_role** (*str*) – the name of an IAM Role (in the destination account) to assume
- **region** (*str*) – AWS region name to connect to
- **external_id** (*str*) – (optional) the External ID string to use when assuming a role via STS.

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.elb'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 25

_abc_registry = <_weakrefset.WeakSet object>

connect()
Connect to API if not already connected; set self.conn.

find_usage()
Determine the current usage for each limit of this service, and update corresponding Limit via *_add_current_usage()*.

get_limits()
Return all known limits for this service, as a dict of their names to *AwsLimit* objects.

Returns dict of limit names to *AwsLimit* objects

Return type dict

required_iam_permissions()
Return a list of IAM Actions required for this Service to function properly. All Actions will be shown with an Effect of "Allow" and a Resource of "*".

Returns list of IAM Action strings

Return type list

service_name = 'ELB'

awslimitchecker.services.rds module

class awslimitchecker.services.rds.**_RDSService**(*warning_threshold*, *critical_threshold*, *account_id=None*, *account_role=None*, *region=None*, *external_id=None*)
Bases: *awslimitchecker.services.base._AwsService*

Describes an AWS service and its limits, and provides methods to query current utilization.

Constructors of _AwsService subclasses *must not* make any external connections; these must be made lazily as needed in other methods. _AwsService subclasses should be usable without any external network connections.

Parameters

- **warning_threshold** (*int*) – the default warning threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **critical_threshold** (*int*) – the default critical threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **account_id** (*str*) – AWS Account ID (12-digit string, currently numeric) for the account to connect to (destination) via STS
- **account_role** (*str*) – the name of an IAM Role (in the destination account) to assume
- **region** (*str*) – AWS region name to connect to
- **external_id** (*str*) – (optional) the External ID string to use when assuming a role via STS.

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.rds'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 25

_abc_registry = <_weakrefset.WeakSet object>

_find_usage_event_subscriptions()
find usage for event subscriptions

_find_usage_instances()
find usage for DB Instances and related limits

_find_usage_option_groups()
find usage for option groups

_find_usage_param_groups()
find usage for parameter groups

_find_usage_reserved_instances()
find usage for reserved instances

_find_usage_security_groups()
find usage for security groups

_find_usage_snapshots()
find usage for (manual) DB snapshots

_find_usage_subnet_groups()
find usage for subnet groups

connect()
Connect to API if not already connected; set self.conn.

find_usage()
Determine the current usage for each limit of this service, and update corresponding Limit via *_add_current_usage()*.

get_limits()
Return all known limits for this service, as a dict of their names to *AwsLimit* objects.

Returns dict of limit names to *AwsLimit* objects

Return type dict

required_iam_permissions()
Return a list of IAM Actions required for this Service to function properly. All Actions will be shown with an Effect of "Allow" and a Resource of "*".

Returns list of IAM Action strings

Return type list

service_name = 'RDS'

awslimitchecker.services.vpc module

class awslimitchecker.services.vpc.**_VpcService**(*warning_threshold*, *critical_threshold*, *account_id=None*, *account_role=None*, *region=None*, *external_id=None*)

Bases: *awslimitchecker.services.base._AwsService*

Describes an AWS service and its limits, and provides methods to query current utilization.

Constructors of _AwsService subclasses *must not* make any external connections; these must be made lazily as needed in other methods. _AwsService subclasses should be usable without any external network connections.

Parameters

- **warning_threshold** (*int*) – the default warning threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **critical_threshold** (*int*) – the default critical threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **account_id** (*str*) – AWS Account ID (12-digit string, currently numeric) for the account to connect to (destination) via STS
- **account_role** (*str*) – the name of an IAM Role (in the destination account) to assume
- **region** (*str*) – AWS region name to connect to
- **external_id** (*str*) – (optional) the External ID string to use when assuming a role via STS.

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.vpc'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 25

_abc_registry = <_weakrefset.WeakSet object>

_find_usage_ACLs()
find usage for ACLs

_find_usage_gateways()
find usage for Internet Gateways

_find_usage_route_tables()
find usage for route tables

_find_usage_subnets()
find usage for Subnets

_find_usage_vpcs()
find usage for VPCs

connect()
Connect to API if not already connected; set self.conn.

find_usage()
Determine the current usage for each limit of this service, and update corresponding Limit via *_add_current_usage()*.

get_limits()
Return all known limits for this service, as a dict of their names to *AwsLimit* objects.

Returns dict of limit names to *AwsLimit* objects

Return type dict

required_iam_permissions()
Return a list of IAM Actions required for this Service to function properly. All Actions will be shown with an Effect of "Allow" and a Resource of "*".

Returns list of IAM Action strings

Return type list

service_name = 'VPC'

Submodules

awslimitchecker.checker module

class awslimitchecker.checker.**AwsLimitChecker**(*warning_threshold=80*, *critical_threshold=99*, *account_id=None*, *account_role=None*, *region=None*, *external_id=None*)

Bases: object

Main AwsLimitChecker class - this should be the only externally-used portion of awslimitchecker.

Constructor builds `self.services` as a dict of service_name (str) to *_AwsService* instance, and sets limit thresholds.

Parameters

- **warning_threshold** (*int*) – the default warning threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **critical_threshold** (*int*) – the default critical threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **account_id** (*str*) – AWS Account ID (12-digit string, currently numeric) for the account to connect to (destination) via STS
- **account_role** (*str*) – the name of an IAM Role (in the destination account) to assume
- **region** (*str*) – AWS region name to connect to
- **external_id** (*str*) – (optional) the External ID string to use when assuming a role via STS.

__dict__ = dict_proxy({'set_limit_override': <function set_limit_override at 0x7f53c0d5b398>, '__module__': 'awslimi

__init__(*warning_threshold=80*, *critical_threshold=99*, *account_id=None*, *account_role=None*, *region=None*, *external_id=None*)

Main AwsLimitChecker class - this should be the only externally-used portion of awslimitchecker.

Constructor builds `self.services` as a dict of service_name (str) to *_AwsService* instance, and sets limit thresholds.

Parameters

- **warning_threshold** (*int*) – the default warning threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **critical_threshold** (*int*) – the default critical threshold, as an integer percentage, for any limits without a specifically-set threshold.
- **account_id** (*str*) – AWS Account ID (12-digit string, currently numeric) for the account to connect to (destination) via STS
- **account_role** (*str*) – the name of an IAM Role (in the destination account) to assume
- **region** (*str*) – AWS region name to connect to
- **external_id** (*str*) – (optional) the External ID string to use when assuming a role via STS.

__module__ = 'awslimitchecker.checker'

__weakref__

list of weak references to the object (if defined)

check_thresholds(*service=None*, *use_ta=True*)
Check all limits and current usage against their specified thresholds; return all *AwsLimit* instances that have crossed one or more of their thresholds.

If `service` is specified, the returned dict has one element, the service name, whose value is a nested dict as described below; otherwise it includes all known services.

The returned *AwsLimit* objects can be interrogated for their limits (*get_limit()*) as well as the details of usage that crossed the thresholds (*get_warnings()* and *get_criticals()*).

Parameters

- **service** (*string*) – the name of one service to return results for
- **use_ta** (*bool*) – check Trusted Advisor for information on limits

Returns dict of service name (string) to nested dict of limit name (string) to limit (*AwsLimit*)

Return type dict

find_usage(*service=None*, *use_ta=True*)
For each limit in the specified service (or all services if `service` is `None`), query the AWS API via `boto` and find the current usage amounts for that limit.

This method updates the `current_usage` attribute of the *AwsLimit* objects for each service, which can then be queried using *get_limits()*.

Parameters

- **service** (`None`, or `string` service name to get) – *_AwsService* name, or `None` to check all services.
- **use_ta** (*bool*) – check Trusted Advisor for information on limits

get_limits(*service=None*, *use_ta=True*)
Return all *AwsLimit* objects for the given service name, or for all services if `service` is None.

If `service` is specified, the returned dict has one element, the service name, whose value is a nested dict as described below.

Parameters

- **service** (*string*) – the name of one service to return limits for
- **use_ta** (*bool*) – check Trusted Advisor for information on limits

Returns dict of service name (string) to nested dict of limit name (string) to limit (*AwsLimit*)

Return type dict

get_project_url()
Return the URL for the awslimitchecker project.

Returns URL of where to find awslimitchecker

Return type string

get_required_iam_policy()
Return an IAM policy granting all of the permissions needed for awslimitchecker to fully function. This returns a dict suitable for json serialization to a valid IAM policy.

Internally, this calls *required_iam_permissions()* on each *_AwsService* instance.

Returns dict representation of IAM Policy

Return type dict

get_service_names()
Return a list of all known service names

Returns list of service names

Return type list

get_version()
Return the version of awslimitchecker currently running.

Returns current awslimitchecker version

Return type string

set_limit_override(*service_name*, *limit_name*, *value*, *override_ta=True*)
Set a manual override on an AWS service limits, i.e. if you had limits increased by AWS support.

This method calls *set_limit_override()* on the corresponding _AwsService instance.

Explicitly set limit overrides using this method will take precedence over default limits. They will also take precedence over limit information obtained via Trusted Advisor, unless `override_ta` is set to `False`.

Parameters

- **service_name** (*string*) – the name of the service to override limit for
- **limit_name** (*string*) – the name of the limit to override:
- **value** (*int*) – the new (overridden) limit value)
- **override_ta** (*bool*) – whether or not to use this value even if Trusted Advisor supplies limit information

Raises ValueError if limit_name is not known to the service instance

set_limit_overrides(*override_dict*, *override_ta=True*)
Set manual overrides on AWS service limits, i.e. if you had limits increased by AWS support. This takes a dict in the same form as that returned by *get_limits()*, i.e. service_name (str) keys to nested dict of limit_name (str) to limit value (int) like:

```
{
    'EC2': {
      'Running On-Demand t2.micro Instances': 1000,
      'Running On-Demand r3.4xlarge Instances': 1000,
    }
}
```

Internally, for each limit override for each service in `override_dict`, this method calls *set_limit_override()* on the corresponding _AwsService instance.

Explicitly set limit overrides using this method will take precedence over default limits. They will also take precedence over limit information obtained via Trusted Advisor, unless `override_ta` is set to `False`.

Parameters

- **override_dict** (*dict*) – dict of overrides to default limits
- **override_ta** (*bool*) – whether or not to use this value even if Trusted Advisor supplies limit information

Raises `exceptions.ValueError` if limit_name is not known to the service instance

set_threshold_override(*service_name*, *limit_name*, *warn_percent=None*, *warn_count=None*, *crit_percent=None*, *crit_count=None*)
Set a manual override on the threshold (used for determining warning/critical status) for a specific limit. See *AwsLimitChecker* for information on Warning and Critical thresholds.

See *set_threshold_override()*.

Parameters

- **service_name** (*string*) – the name of the service to override limit for
- **limit_name** (*string*) – the name of the limit to override:
- **warn_percent** (*int*) – new warning threshold, percentage used
- **warn_count** (*int*) – new warning threshold, actual count/number
- **crit_percent** (*int*) – new critical threshold, percentage used
- **crit_count** (*int*) – new critical threshold, actual count/number

set_threshold_overrides(*override_dict*)

Set manual overrides on the threshold (used for determining warning/critical status) a dict of limits. See *AwsLimitChecker* for information on Warning and Critical thresholds.

Dict is composed of service name keys (string) to dict of limit names (string), to dict of threshold specifications. Each threhold specification dict can contain keys 'warning' or 'critical', each having a value of a dict containing keys 'percent' or 'count', to an integer value.

Example:

```
{
    'EC2': {
        'SomeLimit': {
            'warning': {
                'percent': 80,
                'count': 8,
            },
            'critical': {
                'percent': 90,
                'count': 9,
            }
        }
    }
}
```

See *set_threshold_override()*.

Parameters **override_dict** (*dict*) – nested dict of threshold overrides

awslimitchecker.connectable module

class awslimitchecker.connectable.**Connectable**

Bases: object

Mix-in helper class for connecting to AWS APIs. Centralizes logic of connecting via regions and/or STS.

__dict__ = dict_proxy({'connect_via': <function connect_via at 0x7f53c0e8e410>, '__module__': 'awslimitchecker.conn

__module__ = 'awslimitchecker.connectable'

__weakref__

list of weak references to the object (if defined)

_get_sts_token()

Assume a role via STS and return the credentials.

First connect to STS via boto.sts.connect_to_region(), then assume a role using boto.sts.STSConnection.assume_role() using self.account_id and self.account_role (and optionally self.external_id). Return the resulting boto.sts.credentials.Credentials object.

Returns STS assumed role credentials

Return type boto.sts.credentials.Credentials

connect_via(*driver*)

Connect to an AWS API and return the connection object. If self.account_id is None, call driver(self.region). Otherwise, call *_get_sts_token()* to get STS token credentials using boto.sts.STSConnection.assume_role() and call driver() with those credentials to use an assumed role.

Parameters **driver** (function) – the connect_to_region() function of the boto submodule to use to create this connection

Returns connected boto service class instance

awslimitchecker.limit module

class awslimitchecker.limit.**AwsLimit**(*name*, *service*, *default_limit*, *def_warning_threshold*, *def_critical_threshold*, *limit_type=None*, *limit_subtype=None*)

Bases: object

Describes one specific AWS service limit, as well as its current utilization, default limit, thresholds, and any Trusted Advisor information about this limit.

Parameters

- **name** (*string*) – the name of this limit (may contain spaces); if possible, this should be the name used by AWS, i.e. TrustedAdvisor
- **service** – the *_AwsService* class that this limit is for
- **default_limit** (*int*) – the default value of this limit for new accounts
- **def_warning_threshold** (*int*) – the default warning threshold, as an integer percentage.
- **def_critical_threshold** (*int*) – the default critical threshold, as an integer percentage.
- **limit_type** – the type of resource this limit describes, specified as one of the type names used in CloudFormation # noqa such as "AWS::EC2::Instance" or "AWS::RDS::DBSubnetGroup".
- **limit_subtype** – resource sub-type for this limit, if applicable, such as "t2.micro" or "SecurityGroup"

Raises ValueError

__dict__ = dict_proxy({'__module__': 'awslimitchecker.limit', 'get_current_usage_str': <function get_current_usage_s

__init__(*name*, *service*, *default_limit*, *def_warning_threshold*, *def_critical_threshold*, *limit_type=None*, *limit_subtype=None*)

Describes one specific AWS service limit, as well as its current utilization, default limit, thresholds, and any Trusted Advisor information about this limit.

Parameters

- **name** (*string*) – the name of this limit (may contain spaces); if possible, this should be the name used by AWS, i.e. TrustedAdvisor
- **service** – the *_AwsService* class that this limit is for
- **default_limit** (*int*) – the default value of this limit for new accounts
- **def_warning_threshold** (*int*) – the default warning threshold, as an integer percentage.
- **def_critical_threshold** (*int*) – the default critical threshold, as an integer percentage.
- **limit_type** – the type of resource this limit describes, specified as one of the type names used in CloudFormation # noqa such as "AWS::EC2::Instance" or "AWS::RDS::DBSubnetGroup".
- **limit_subtype** – resource sub-type for this limit, if applicable, such as "t2.micro" or "SecurityGroup"

Raises ValueError

__module__ = 'awslimitchecker.limit'

__weakref__
list of weak references to the object (if defined)

_add_current_usage(*value*, *resource_id=None*, *aws_type=None*)
Add a new current usage value for this limit.

Creates a new *AwsLimitUsage* instance and appends it to the internal list. If more than one usage value is given to this service, they should have `id` and `aws_type` set.

This method should only be called from the *_AwsService* instance that created and manages this Limit.

Parameters

- **value** (`int` or `float`) – the numeric usage value
- **resource_id** (*string*) – If there can be multiple usage values for one limit, an AWS ID for the resource this instance describes
- **aws_type** (*string*) – if `id` is not None, the AWS resource type that ID represents. As a convention, we use the AWS Resource Type names used by CloudFormation # noqa

_get_thresholds()
Get the warning and critical thresholds for this Limit.

Return type is a 4-tuple of:

1. warning integer (usage) threshold, or None
2. warning percent threshold
3. critical integer (usage) threshold, or None
4. critical percent threshold

Return type tuple

_reset_usage()
Discard all current usage data.

_set_api_limit(*limit_value*)
Set the value for the limit as reported by the service's API.

This method should only be called from the Service class.

Parameters **limit_value** (*int*) – the API limit value

_set_ta_limit(*limit_value*)
Set the value for the limit as reported by Trusted Advisor.

This method should only be called by *TrustedAdvisor*.

Parameters **limit_value** (*int*) – the Trusted Advisor limit value

check_thresholds()
Check this limit's current usage against the specified default thresholds, and any custom theresholds that have been set on the instance. Return True if usage is within thresholds, or false if warning or critical thresholds have been surpassed.

This method sets internal variables in this instance which can be queried via *get_warnings()* and *get_criticals()* to obtain further details about the thresholds that were crossed.

Parameters

- **default_warning** (int or float percentage) – default warning threshold in percentage; usage higher than this percent of the limit will be considered a warning
- **default_critical** (int or float percentage) – default critical threshold in percentage; usage higher than this percent of the limit will be considered a critical

Returns False if any thresholds crossed, True otherwise

Return type bool

get_criticals()
Return a list of *AwsLimitUsage* instances that crossed the critical threshold. These objects are comparable and can be sorted.

Return type list

get_current_usage()
Get the current usage for this limit, as a list of *AwsLimitUsage* instances.

Returns list of current usage values

Return type list of *AwsLimitUsage*

get_current_usage_str()
Get the a string describing the current usage for this limit.

If no usage has been added for this limit, the result will be "<unknown>".

If the limit has only one current usage instance, this will be that instance's *__str__()* value.

If the limit has more than one current usage instance, this will be the a string of the form `max: X (Y)` where `X` is the *__str__()* value of the instance with the maximum value, and `Y` is a comma-separated list of the *__str__()* values of all usage instances in ascending order.

Returns representation of current usage

Return type string

get_limit()
Returns the effective limit value for this Limit, taking into account limit overrides and Trusted Advisor data.

Returns effective limit value

Return type int

get_limit_source()
Return *SOURCE_DEFAULT* if *get_limit()* returns the default limit, *SOURCE_OVERRIDE* if it returns a manually-overridden limit, *SOURCE_TA* if it returns a limit from Trusted Advisor, or *SOURCE_API* if it returns a limit retrieved from the service's API.

Returns one of *SOURCE_DEFAULT*, *SOURCE_OVERRIDE*, or *SOURCE_TA*, or *SOURCE_API*

Return type int

get_warnings()
Return a list of *AwsLimitUsage* instances that crossed the warning threshold. These objects are comparable and can be sorted.

Return type list

set_limit_override(*limit_value*, *override_ta=True*)
Set a new value for this limit, to override the default (such as when AWS Support has increased a limit of yours). If `override_ta` is True, this value will also supersede any found through Trusted Advisor.

Parameters

- **limit_value** (*int*) – the new limit value
- **override_ta** (*bool*) – whether or not to also override Trusted Advisor information

set_threshold_override(*warn_percent=None*, *warn_count=None*, *crit_percent=None*, *crit_count=None*)
Override the default warning and critical thresholds used to evaluate this limit's usage. Theresholds can be specified as a percentage of the limit, or as a usage count, or both.

Parameters

- **warn_percent** (*int*) – new warning threshold, percentage used
- **warn_count** (*int*) – new warning threshold, actual count/number
- **crit_percent** (*int*) – new critical threshold, percentage used
- **crit_count** (*int*) – new critical threshold, actual count/number

class awslimitchecker.limit.**AwsLimitUsage**(*limit*, *value*, *resource_id=None*, *aws_type=None*)
Bases: object

This object describes the usage of an AWS resource, with the capability of containing information about the resource beyond an integer usage.

The simplest case is an account- / region-wide count, such as the number of running EC2 Instances, in which case a simple integer value is sufficient. In this case, the *AwsLimit* would have one instance of this class for the single value.

In more complex cases, such as the "Subnets per VPC", the limit is applied by AWS on multiple resources (once per VPC). In this case, the *AwsLimit* should have one instance of this class per VPC, so we can determine *which* VPCs have crossed thresholds.

AwsLimitUsage objects are comparable based on their numeric `value`.

Parameters

- **limit** (*AwsLimit*) – the AwsLimit that this instance describes
- **value** (`int` or `float`) – the numeric usage value

- **resource_id** (*string*) – If there can be multiple usage values for one limit, an AWS ID for the resource this instance describes
- **aws_type** (*string*) – if `id` is not None, the AWS resource type that ID represents. As a convention, we use the AWS Resource Type names used by CloudFormation # noqa

__dict__ = dict_proxy({'__ne__': <function __ne__ at 0x7f53c0e9baa0>, '__module__': 'awslimitchecker.limit', '__wea

__eq__(*other*)

__ge__(*other*)

__gt__(*other*)

__init__(*limit*, *value*, *resource_id=None*, *aws_type=None*)

This object describes the usage of an AWS resource, with the capability of containing information about the resource beyond an integer usage.

The simplest case is an account- / region-wide count, such as the number of running EC2 Instances, in which case a simple integer value is sufficient. In this case, the *AwsLimit* would have one instance of this class for the single value.

In more complex cases, such as the "Subnets per VPC", the limit is applied by AWS on multiple resources (once per VPC). In this case, the *AwsLimit* should have one instance of this class per VPC, so we can determine *which* VPCs have crossed thresholds.

AwsLimitUsage objects are comparable based on their numeric `value`.

Parameters

- **limit** (*AwsLimit*) – the AwsLimit that this instance describes
- **value** (`int` or `float`) – the numeric usage value
- **resource_id** (*string*) – If there can be multiple usage values for one limit, an AWS ID for the resource this instance describes
- **aws_type** (*string*) – if `id` is not None, the AWS resource type that ID represents. As a convention, we use the AWS Resource Type names used by CloudFormation # noqa

__lt__(*other*)

__module__ = 'awslimitchecker.limit'

__ne__(*other*)

__str__()

Return a string representation of this object.

If `id` is not set, return `value` formatted as a string; otherwise, return a string of the format `id=value`.

Return type string

__weakref__

list of weak references to the object (if defined)

get_value()

Get the current usage value

Returns current usage value

Return type `int` or `float`

`awslimitchecker.limit.`**SOURCE_API = 3**

indicates a limit value that came from the service's API

awslimitchecker.limit.**SOURCE_DEFAULT = 0**
indicates a limit value that came from hard-coded defaults in awslimitchecker

awslimitchecker.limit.**SOURCE_OVERRIDE = 1**
indicates a limit value that came from user-defined limit overrides

awslimitchecker.limit.**SOURCE_TA = 2**
indicates a limit value that came from Trusted Advisor

awslimitchecker.runner module

class awslimitchecker.runner.**Runner**
Bases: object

__dict__ = dict_proxy({'__module__': 'awslimitchecker.runner', 'color_output': <function color_output at 0x7f53c0d5

__init__()

__module__ = 'awslimitchecker.runner'

__weakref__
list of weak references to the object (if defined)

check_thresholds()

color_output(*s*, *color*)

console_entry_point()

iam_policy()

list_defaults()

list_limits()

list_services()

parse_args(*argv*)
parse arguments/options

Parameters **argv** (*list*) – argument list to parse, usually sys.argv[1:]

Returns parsed arguments

Return type argparse.Namespace

print_issue(*service_name*, *limit*, *crits*, *warns*)

Parameters

- **service_name** (*str*) – the name of the service
- **limit** (*AwsLimit*) – the Limit this relates to
- **crits** – the specific usage values that crossed the critical threshold
- **crits** – the specific usage values that crossed the warning threshold

set_limit_overrides(*overrides*)

show_usage()

awslimitchecker.runner.**console_entry_point**()

awslimitchecker.trustedadvisor module

class awslimitchecker.trustedadvisor.**TrustedAdvisor**(*account_id=None*, *account_role=None*, *region=None*, *external_id=None*)

Bases: *awslimitchecker.connectable.Connectable*

Class to contain all TrustedAdvisor-related logic.

Parameters

- **account_id** (*str*) – AWS Account ID (12-digit string, currently numeric) for the account to connect to (destination) via STS
- **account_role** (*str*) – the name of an IAM Role (in the destination account) to assume
- **region** (*str*) – AWS region name to connect to
- **external_id** (*str*) – (optional) the External ID string to use when assuming a role via STS.

__init__(*account_id=None*, *account_role=None*, *region=None*, *external_id=None*)

Class to contain all TrustedAdvisor-related logic.

Parameters

- **account_id** (*str*) – AWS Account ID (12-digit string, currently numeric) for the account to connect to (destination) via STS
- **account_role** (*str*) – the name of an IAM Role (in the destination account) to assume
- **region** (*str*) – AWS region name to connect to
- **external_id** (*str*) – (optional) the External ID string to use when assuming a role via STS.

__module__ = 'awslimitchecker.trustedadvisor'

_get_limit_check_id()

Query currently-available TA checks, return the check ID and metadata of the 'performance/Service Limits' check.

Returns 2-tuple of Service Limits TA check ID (string), metadata (list), or (None, None).

Return type tuple

_poll()

Poll Trusted Advisor (Support) API for limit checks.

Return a dict of service name (string) keys to nested dict vals, where each key is a limit name and each value the current numeric limit.

e.g.:

```
{
    'EC2': {
        'SomeLimit': 10,
    }
}
```

_update_services(*ta_results*, *services*)

Given a dict of TrustedAdvisor check results from *_poll()* and a dict of Service objects passed in to *update_limits()*, updated the TrustedAdvisor limits for all services.

Parameters

- **ta_results** (*dict*) – results returned by *_poll()*
- **services** (*dict*) – dict of service names to _AwsService objects

connect()
Connect to API if not already connected; set self.conn.

service_name = 'TrustedAdvisor'

update_limits(*services*)
Poll 'Service Limits' check results from Trusted Advisor, if possible. Iterate over all *AwsLimit* objects for the given services and update their limits from TA if present in TA checks.

Parameters **services** (*dict*) – dict of service name (string) to *_AwsService* objects

awslimitchecker.utils module

class awslimitchecker.utils.**StoreKeyValuePair**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)
Bases: argparse.Action

Store key=value options in a dict as {'key': 'value'}.

Supports specifying the option multiple times, but NOT with nargs.

See Action.

__call__(*parser*, *namespace*, *values*, *option_string=None*)

__init__(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

__module__ = 'awslimitchecker.utils'

awslimitchecker.utils.**_get_dict_value_by_path**(*d*, *path*)
Given a dict (d) and a list specifying the hierarchical path to a key in that dict (path), return the value at that path or None if it does not exist.

Parameters

- **d** (*dict*) – the dict to search in
- **path** (*list*) – the path to the key in the dict

awslimitchecker.utils.**_paginate_dict**(*result*, *function_ref*, **argv*, ***kwargs*)
Paginate through a query that returns a dict result, and return the combined result.

Note that this function requires some special kwargs to be passed in:

- __alc_marker_path__ - The dictionary path to the Marker for the next result set. If this path does not exist, the raw result will be returned.
- __alc_data_path__ - The dictionary path to the list containing the query results. This will be updated with the results of subsequent queries.
- __alc_marker_param__ - The parameter name to pass to function_ref with the marker value.

These paths should be lists, in a form usable by *_get_dict_value_by_path()*.

All function calls are passed through *invoke_with_throttling_retries()*.

Parameters

- **result** (*dict*) – the first result from the query
- **function_ref** (*function*) – the function to call
- **argv** (*tuple*) – the parameters to pass to the function
- **kwargs** (*dict*) – keyword arguments to pass to the function (*invoke_with_throttling_retries()*)

awslimitchecker.utils.**_paginate_resultset**(*result*, *function_ref*, **argv*, ***kwargs*)

Paginate through a query that returns a boto.resultset.ResultSet object, and return the combined result.

All function calls are passed through *invoke_with_throttling_retries()*.

Parameters

- **result** (boto.resultset.ResultSet) – the first ResultSet from the query
- **function_ref** (*function*) – the function to call
- **argv** (*tuple*) – the parameters to pass to the function
- **kwargs** (*dict*) – keyword arguments to pass to the function (*invoke_with_throttling_retries()*)

awslimitchecker.utils.**_set_dict_value_by_path**(*d*, *val*, *path*)

Given a dict (d), a value (val), and a list specifying the hierarchical path to a key in that dict (path), set the value in d at path to val.

Parameters

- **d** (*dict*) – the dict to search in
- **path** (*list*) – the path to the key in the dict

Raises TypeError if the path is too short

Returns the modified dict

awslimitchecker.utils.**boto_query_wrapper**(*function_ref*, **argv*, ***kwargs*)

Function to wrap all boto query method calls, for throttling and pagination.

Calls *paginate_query()* and returns the result.

Parameters

- **function_ref** (*function*) – the function to call
- **argv** (*tuple*) – the parameters to pass to the function
- **kwargs** (*dict*) – keyword arguments to pass to the function (*paginate_query()*)

Returns return value of function_ref

awslimitchecker.utils.**dict2cols**(*d*, *spaces=2*, *separator=' '*)

Take a dict of string keys and string values, and return a string with them formatted as two columns separated by at least spaces number of separator characters.

Parameters

- **d** (*dict*) – dict of string keys, string values
- **spaces** (*int*) – number of spaces to separate columns by
- **separator** (*string*) – character to fill in between columns

awslimitchecker.utils.**invoke_with_throttling_retries**(*function_ref*, **argv*, ***kwargs*)
Invoke a Boto operation using an exponential backoff in the case of API request throttling.

This is taken from: https://github.com/47lining/ansible-modules-core/blob/2d189f0d192717f83e3c6d37d3fe0988fc329b5a/cloud/a
see: https://github.com/ansible/ansible-modules-core/pull/224 and https://github.com/ansible/ansible-modules-core/pull/569

To use, transform:

```
conn.action(args)
```

into:

```
invoke_with_throttling_retries(conn.action, args)
```

Parameters

- **function_ref** (*function*) – the function to call
- **argv** (*tuple*) – the parameters to pass to the function
- **kwargs** (*dict*) – keyword arguments to pass to the function. Any arguments with names starting with `alc_` will be removed for internal use.

awslimitchecker.utils.**paginate_query**(*function_ref*, **argv*, ***kwargs*)
Invoke a Boto operation, automatically paginating through all responses.

If `function_ref` returns a `boto.resultset.ResultSet` object and its `next_token` attribute is not None, pass it through to *`_paginate_resultset()`* and return the result.

Else if `function_ref` returns a dict, pass it through to *`_paginate_dict()`* and return the result.

Else, return the result.

Parameters

- **function_ref** (*function*) – the function to call
- **argv** (*tuple*) – the parameters to pass to the function
- **kwargs** (*dict*) – keyword arguments to pass to the function (*`invoke_with_throttling_retries()`*)

awslimitchecker.version module

class awslimitchecker.version.**AWSLimitCheckerVersion**(*release*, *url*, *commit=None*, *tag=None*)
Bases: `object`

__dict__ = dict_proxy({'__module__': 'awslimitchecker.version', '__str__': <function __str__ at 0x7f53c2980b18>, '__

__init__(*release*, *url*, *commit=None*, *tag=None*)

__module__ = 'awslimitchecker.version'

__repr__()
Return a representation of this object that is valid Python and will create an idential AWSLimitCheckerVersion object.

Return type string

__str__()
Human-readable string representation of this version object, in the format: "version_str <url>"

Return type string

__weakref__
list of weak references to the object (if defined)

version_str
The version string for the currently-running awslimitchecker; includes git branch and tag information.

Return type string

awslimitchecker.version.**_get_version_info**()
Returns the currently-installed awslimitchecker version, and a best-effort attempt at finding the origin URL and commit/tag if installed from an editable git clone.

Returns awslimitchecker version

Return type string

awslimitchecker.versioncheck module

class awslimitchecker.versioncheck.**AGPLVersionChecker**
Bases: object

__dict__ = dict_proxy({'__module__': 'awslimitchecker.versioncheck', '__doc__': None, '_find_git_info': <function _fi

__module__ = 'awslimitchecker.versioncheck'

__weakref__
list of weak references to the object (if defined)

_dist_version_url(*dist*)
Get version and homepage for a pkg_resources.Distribution

Parameters **dist** – the pkg_resources.Distribution to get information for

Returns 2-tuple of (version, homepage URL)

Return type tuple

_find_git_info()
Find information about the git repository, if this file is in a clone.

Returns information about the git clone

Return type dict

_find_pip_info()
Try to find information about the installed awslimitchecker from pip. This should be wrapped in a try/except.

Returns information from pip about 'awslimitchecker'

Return type dict

_find_pkg_info()
Find information about the installed awslimitchecker from pkg_resources.

Returns information from pkg_resources about 'awslimitchecker'

Return type dict

_is_git_clone
Attempt to determine whether this package is installed via git or not.

Return type bool

Returns True if installed via git, False otherwise

_is_git_dirty()
Determine if the git clone has uncommitted changes or is behind origin

Returns True if clone is dirty, False otherwise

Return type bool

find_package_version()
Find the installed version of the specified package, and as much information about it as possible (source URL, git ref or tag, etc.)

This attempts, to the best of our ability, to find out if the package was installed from git, and if so, provide information on the origin of that git repository and status of the clone. Otherwise, it uses pip and pkg_resources to find the version and homepage of the installed distribution.

This class is not a sure-fire method of identifying the source of the distribution or ensuring AGPL compliance; it simply helps with this process _iff_ a modified version is installed from an editable git URL _and_ all changes are pushed up to the publicly-visible origin.

Returns a dict with keys 'version', 'tag', 'commit', and 'url'. Values are strings or None.

Parameters **package_name** (*str*) – name of the package to find information for

Returns information about the installed version of the package

Return type dict

awslimitchecker.versioncheck.**_check_output**(*args*, *stderr=None*)
Python version compatibility wrapper for subprocess.check_output

Parameters **stderr** – what to do with STDERR - None or an appropriate argument to subprocess.check_output / subprocess.Popen

Raises subprocess.CalledProcessError

Returns command output

Return type string

awslimitchecker.versioncheck.**_get_git_commit**()
Get the current (short) git commit hash of the current directory.

Returns short git hash

Return type string

awslimitchecker.versioncheck.**_get_git_tag**(*commit*)
Get the git tag for the specified commit, or None

Parameters **commit** (*string*) – git commit hash to get the tag for

Returns tag name pointing to commit

Return type string

awslimitchecker.versioncheck.**_get_git_url**()
Get the origin URL for the git repository.

Returns repository origin URL

Return type string

7.9 Getting Help

7.9.1 Reporting Bugs and Asking Questions

Questions, bug reports and feature requests are happily accepted via the GitHub Issue Tracker. Pull requests are welcome. Issues that don't have an accompanying pull request will be worked on as my time and priority allows, but I'll do my best to complete feature requests as quickly as possible. Please take into account that I work on this project solely in my personal time, I don't get paid to work on it and I can't work on it for my day job, so there may be some delay in getting things implemented.

7.9.2 IRC

I can often be found on freenode as "jantman".

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

8.1 License

awslimitchecker is licensed under the GNU Affero General Public License, version 3 or later. This shouldn't be much of a concern to most people; see *Development / AGPL* for more information.

a

awslimitchecker, 36
awslimitchecker.checker, 48
awslimitchecker.connectable, 51
awslimitchecker.limit, 52
awslimitchecker.runner, 57
awslimitchecker.services, 36
awslimitchecker.services.autoscaling, 36
awslimitchecker.services.base, 38
awslimitchecker.services.ebs, 40
awslimitchecker.services.ec2, 41
awslimitchecker.services.elasticache, 43
awslimitchecker.services.elb, 44
awslimitchecker.services.rds, 45
awslimitchecker.services.vpc, 46
awslimitchecker.trustedadvisor, 58
awslimitchecker.utils, 59
awslimitchecker.version, 61
awslimitchecker.versioncheck, 62

Symbols

_AutoscalingService (class in awslimitchecker.services.autoscaling), 36
_AwsService (class in awslimitchecker.services.base), 38
_EbsService (class in awslimitchecker.services.ebs), 40
_Ec2Service (class in awslimitchecker.services.ec2), 41
_ElastiCacheService (class in awslimitchecker.services.elasticache), 43
_ElbService (class in awslimitchecker.services.elb), 44
_RDSService (class in awslimitchecker.services.rds), 45
_VpcService (class in awslimitchecker.services.vpc), 46
__abstractmethods__ (awslimitchecker.services.autoscaling._AutoscalingService attribute), 37
__abstractmethods__ (awslimitchecker.services.base._AwsService attribute), 38
__abstractmethods__ (awslimitchecker.services.ebs._EbsService attribute), 40
__abstractmethods__ (awslimitchecker.services.ec2._Ec2Service attribute), 41
__abstractmethods__ (awslimitchecker.services.elasticache._ElastiCacheService attribute), 43
__abstractmethods__ (awslimitchecker.services.elb._ElbService attribute), 44
__abstractmethods__ (awslimitchecker.services.rds._RDSService attribute), 45
__abstractmethods__ (awslimitchecker.services.vpc._VpcService attribute), 47
__call__() (awslimitchecker.utils.StoreKeyValuePair method), 59
__dict__ (awslimitchecker.checker.AwsLimitChecker attribute), 48
__dict__ (awslimitchecker.connectable.Connectable attribute), 51
__dict__ (awslimitchecker.limit.AwsLimit attribute), 52
__dict__ (awslimitchecker.limit.AwsLimitUsage attribute), 56
__dict__ (awslimitchecker.runner.Runner attribute), 57
__dict__ (awslimitchecker.version.AWSLimitCheckerVersion attribute), 61
__dict__ (awslimitchecker.versioncheck.AGPLVersionChecker attribute), 62
__eq__() (awslimitchecker.limit.AwsLimitUsage method), 56
__ge__() (awslimitchecker.limit.AwsLimitUsage method), 56
__gt__() (awslimitchecker.limit.AwsLimitUsage method), 56
__init__() (awslimitchecker.checker.AwsLimitChecker method), 48
__init__() (awslimitchecker.limit.AwsLimit method), 52
__init__() (awslimitchecker.limit.AwsLimitUsage method), 56
__init__() (awslimitchecker.runner.Runner method), 57
__init__() (awslimitchecker.services.base._AwsService method), 38
__init__() (awslimitchecker.trustedadvisor.TrustedAdvisor method), 58
__init__() (awslimitchecker.utils.StoreKeyValuePair method), 59
__init__() (awslimitchecker.version.AWSLimitCheckerVersion method), 61
__lt__() (awslimitchecker.limit.AwsLimitUsage method), 56
__metaclass__ (awslimitchecker.services.base._AwsService attribute), 38
__module__ (awslimitchecker.checker.AwsLimitChecker attribute), 48
__module__ (awslimitchecker.connectable.Connectable attribute), 51
__module__ (awslimitchecker.limit.AwsLimit attribute), 53
__module__ (awslimitchecker.limit.AwsLimitUsage at-

tribute), 56
__module__ (awslimitchecker.runner.Runner attribute), 57
__module__ (awslimitchecker.services.autoscaling._AutoscalingService attribute), 37
__module__ (awslimitchecker.services.base._AwsService attribute), 38
__module__ (awslimitchecker.services.ebs._EbsService attribute), 40
__module__ (awslimitchecker.services.ec2._Ec2Service attribute), 41
__module__ (awslimitchecker.services.elasticache._ElastiCacheService attribute), 43
__module__ (awslimitchecker.services.elb._ElbService attribute), 44
__module__ (awslimitchecker.services.rds._RDSService attribute), 45
__module__ (awslimitchecker.services.vpc._VpcService attribute), 47
__module__ (awslimitchecker.trustedadvisor.TrustedAdvisor attribute), 58
__module__ (awslimitchecker.utils.StoreKeyValuePair attribute), 59
__module__ (awslimitchecker.version.AWSLimitCheckerVersion attribute), 61
__module__ (awslimitchecker.versioncheck.AGPLVersionChecker attribute), 62
__ne__() (awslimitchecker.limit.AwsLimitUsage method), 56
__repr__() (awslimitchecker.version.AWSLimitCheckerVersion method), 61
__str__() (awslimitchecker.limit.AwsLimitUsage method), 56
__str__() (awslimitchecker.version.AWSLimitCheckerVersion method), 61
__weakref__ (awslimitchecker.checker.AwsLimitChecker attribute), 48
__weakref__ (awslimitchecker.connectable.Connectable attribute), 51
__weakref__ (awslimitchecker.limit.AwsLimit attribute), 53
__weakref__ (awslimitchecker.limit.AwsLimitUsage attribute), 56
__weakref__ (awslimitchecker.runner.Runner attribute), 57
__weakref__ (awslimitchecker.version.AWSLimitCheckerVersion attribute), 62
__weakref__ (awslimitchecker.versioncheck.AGPLVersionChecker attribute), 62
_abc_cache (awslimitchecker.services.autoscaling._AutoscalingService attribute), 37
_abc_cache (awslimitchecker.services.base._AwsService attribute), 38
_abc_cache (awslimitchecker.services.ebs._EbsService attribute), 40
_abc_cache (awslimitchecker.services.ec2._Ec2Service attribute), 42
_abc_cache (awslimitchecker.services.elasticache._ElastiCacheService attribute), 43
_abc_cache (awslimitchecker.services.elb._ElbService attribute), 44
_abc_cache (awslimitchecker.services.rds._RDSService attribute), 45
_abc_cache (awslimitchecker.services.vpc._VpcService attribute), 47
_abc_negative_cache (awslimitchecker.services.autoscaling._AutoscalingService attribute), 37
_abc_negative_cache (awslimitchecker.services.base._AwsService attribute), 38
_abc_negative_cache (awslimitchecker.services.ebs._EbsService attribute), 40
_abc_negative_cache (awslimitchecker.services.ec2._Ec2Service attribute), 42
_abc_negative_cache (awslimitchecker.services.elasticache._ElastiCacheService attribute), 43
_abc_negative_cache (awslimitchecker.services.elb._ElbService attribute), 44
_abc_negative_cache (awslimitchecker.services.rds._RDSService attribute), 45
_abc_negative_cache (awslimitchecker.services.vpc._VpcService attribute), 47
_abc_negative_cache_version (awslimitchecker.services.autoscaling._AutoscalingService attribute), 37
_abc_negative_cache_version (awslimitchecker.services.base._AwsService attribute), 38
_abc_negative_cache_version (awslimitchecker.services.ebs._EbsService attribute), 40
_abc_negative_cache_version (awslimitchecker.services.ec2._Ec2Service attribute), 42
_abc_negative_cache_version (awslimitchecker.services.elasticache._ElastiCacheService attribute), 43
_abc_negative_cache_version (awslimitchecker.services.elb._ElbService attribute), 44
_abc_negative_cache_version (awslim-

itchecker.services.rds._RDSService attribute), 45
_abc_negative_cache_version (awslimitchecker.services.vpc._VpcService attribute), 47
_abc_registry (awslimitchecker.services.autoscaling._AutoscalingService attribute), 37
_abc_registry (awslimitchecker.services.base._AwsService attribute), 39
_abc_registry (awslimitchecker.services.ebs._EbsService attribute), 40
_abc_registry (awslimitchecker.services.ec2._Ec2Service attribute), 42
_abc_registry (awslimitchecker.services.elasticache._ElastiCacheService attribute), 43
_abc_registry (awslimitchecker.services.elb._ElbService attribute), 44
_abc_registry (awslimitchecker.services.rds._RDSService attribute), 45
_abc_registry (awslimitchecker.services.vpc._VpcService attribute), 47
_add_current_usage() (awslimitchecker.limit.AwsLimit method), 53
_check_output() (in module awslimitchecker.versioncheck), 63
_dist_version_url() (awslimitchecker.versioncheck.AGPLVersionChecker method), 62
_find_git_info() (awslimitchecker.versioncheck.AGPLVersionChecker method), 62
_find_pip_info() (awslimitchecker.versioncheck.AGPLVersionChecker method), 62
_find_pkg_info() (awslimitchecker.versioncheck.AGPLVersionChecker method), 62
_find_usage_ACLs() (awslimitchecker.services.vpc._VpcService method), 47
_find_usage_ebs() (awslimitchecker.services.ebs._EbsService method), 40
_find_usage_event_subscriptions() (awslimitchecker.services.rds._RDSService method), 46
_find_usage_gateways() (awslimitchecker.services.vpc._VpcService method), 47
_find_usage_instances() (awslimitchecker.services.ec2._Ec2Service method), 42
_find_usage_instances() (awslimitchecker.services.rds._RDSService method), 46
_find_usage_networking_eips() (awslimitchecker.services.ec2._Ec2Service method), 42
_find_usage_networking_eni_sg() (awslimitchecker.services.ec2._Ec2Service method), 42
_find_usage_networking_sgs() (awslimitchecker.services.ec2._Ec2Service method), 42
_find_usage_nodes() (awslimitchecker.services.elasticache._ElastiCacheService method), 43
_find_usage_option_groups() (awslimitchecker.services.rds._RDSService method), 46
_find_usage_param_groups() (awslimitchecker.services.rds._RDSService method), 46
_find_usage_parameter_groups() (awslimitchecker.services.elasticache._ElastiCacheService method), 43
_find_usage_reserved_instances() (awslimitchecker.services.rds._RDSService method), 46
_find_usage_route_tables() (awslimitchecker.services.vpc._VpcService method), 47
_find_usage_security_groups() (awslimitchecker.services.elasticache._ElastiCacheService method), 43
_find_usage_security_groups() (awslimitchecker.services.rds._RDSService method), 46
_find_usage_snapshots() (awslimitchecker.services.ebs._EbsService method), 40
_find_usage_snapshots() (awslimitchecker.services.rds._RDSService method), 46
_find_usage_subnet_groups() (awslimitchecker.services.elasticache._ElastiCacheService method), 44
_find_usage_subnet_groups() (awslimitchecker.services.rds._RDSService method), 46
_find_usage_subnets() (awslimitchecker.services.vpc._VpcService method), 47
_find_usage_vpcs() (awslimitchecker.services.vpc._VpcService method), 47
_get_dict_value_by_path() (in module awslimitchecker.utils), 59

_get_git_commit() (in module awslimitchecker.versioncheck), 63
_get_git_tag() (in module awslimitchecker.versioncheck), 63
_get_git_url() (in module awslimitchecker.versioncheck), 63
_get_limit_check_id() (awslimitchecker.trustedadvisor.TrustedAdvisor method), 58
_get_limits_ebs() (awslimitchecker.services.ebs._EbsService method), 41
_get_limits_instances() (awslimitchecker.services.ec2._Ec2Service method), 42
_get_limits_networking() (awslimitchecker.services.ec2._Ec2Service method), 42
_get_reserved_instance_count() (awslimitchecker.services.ec2._Ec2Service method), 42
_get_sts_token() (awslimitchecker.connectable.Connectable method), 51
_get_thresholds() (awslimitchecker.limit.AwsLimit method), 53
_get_version_info() (in module awslimitchecker.version), 62
_instance_types() (awslimitchecker.services.ec2._Ec2Service method), 42
_instance_usage() (awslimitchecker.services.ec2._Ec2Service method), 42
_is_git_clone (awslimitchecker.versioncheck.AGPLVersionChecker attribute), 62
_is_git_dirty() (awslimitchecker.versioncheck.AGPLVersionChecker method), 63
_paginate_dict() (in module awslimitchecker.utils), 59
_paginate_resultset() (in module awslimitchecker.utils), 60
_poll() (awslimitchecker.trustedadvisor.TrustedAdvisor method), 58
_reset_usage() (awslimitchecker.limit.AwsLimit method), 53
_set_api_limit() (awslimitchecker.limit.AwsLimit method), 53
_set_dict_value_by_path() (in module awslimitchecker.utils), 60
_set_ta_limit() (awslimitchecker.limit.AwsLimit method), 54
_set_ta_limit() (awslimitchecker.services.base._AwsService method), 39
_update_limits_from_api() (awslimitchecker.services.autoscaling._AutoscalingService method), 37
_update_limits_from_api() (awslimitchecker.services.ec2._Ec2Service method), 42
_update_services() (awslimitchecker.trustedadvisor.TrustedAdvisor method), 58

A

AGPLVersionChecker (class in awslimitchecker.versioncheck), 62
AwsLimit (class in awslimitchecker.limit), 52
AwsLimitChecker (class in awslimitchecker.checker), 48
awslimitchecker (module), 36
awslimitchecker.checker (module), 48
awslimitchecker.connectable (module), 51
awslimitchecker.limit (module), 52
awslimitchecker.runner (module), 57
awslimitchecker.services (module), 36
awslimitchecker.services.autoscaling (module), 36
awslimitchecker.services.base (module), 38
awslimitchecker.services.ebs (module), 40
awslimitchecker.services.ec2 (module), 41
awslimitchecker.services.elasticache (module), 43
awslimitchecker.services.elb (module), 44
awslimitchecker.services.rds (module), 45
awslimitchecker.services.vpc (module), 46
awslimitchecker.trustedadvisor (module), 58
awslimitchecker.utils (module), 59
awslimitchecker.version (module), 61
awslimitchecker.versioncheck (module), 62
AwsLimitCheckerVersion (class in awslimitchecker.version), 61
AwsLimitUsage (class in awslimitchecker.limit), 55

B

boto_query_wrapper() (in module awslimitchecker.utils), 60

C

check_thresholds() (awslimitchecker.checker.AwsLimitChecker method), 48
check_thresholds() (awslimitchecker.limit.AwsLimit method), 54
check_thresholds() (awslimitchecker.runner.Runner method), 57
check_thresholds() (awslimitchecker.services.base._AwsService method), 39

color_output() (awslimitchecker.runner.Runner method), 57
connect() (awslimitchecker.services.autoscaling._AutoscalingService method), 37
connect() (awslimitchecker.services.base._AwsService method), 39
connect() (awslimitchecker.services.ebs._EbsService method), 41
connect() (awslimitchecker.services.ec2._Ec2Service method), 42
connect() (awslimitchecker.services.elasticache._ElastiCacheService method), 44
connect() (awslimitchecker.services.elb._ElbService method), 45
connect() (awslimitchecker.services.rds._RDSService method), 46
connect() (awslimitchecker.services.vpc._VpcService method), 47
connect() (awslimitchecker.trustedadvisor.TrustedAdvisor method), 59
connect_via() (awslimitchecker.connectable.Connectable method), 52
Connectable (class in awslimitchecker.connectable), 51
console_entry_point() (awslimitchecker.runner.Runner method), 57
console_entry_point() (in module awslimitchecker.runner), 57

D

dict2cols() (in module awslimitchecker.utils), 60

F

find_package_version() (awslimitchecker.versioncheck.AGPLVersionChecker method), 63
find_usage() (awslimitchecker.checker.AwsLimitChecker method), 49
find_usage() (awslimitchecker.services.autoscaling._AutoscalingService method), 37
find_usage() (awslimitchecker.services.base._AwsService method), 39
find_usage() (awslimitchecker.services.ebs._EbsService method), 41
find_usage() (awslimitchecker.services.ec2._Ec2Service method), 42
find_usage() (awslimitchecker.services.elasticache._ElastiCacheService method), 44
find_usage() (awslimitchecker.services.elb._ElbService method), 45
find_usage() (awslimitchecker.services.rds._RDSService method), 46
find_usage() (awslimitchecker.services.vpc._VpcService method), 47

G

get_criticals() (awslimitchecker.limit.AwsLimit method), 54
get_current_usage() (awslimitchecker.limit.AwsLimit method), 54
get_current_usage_str() (awslimitchecker.limit.AwsLimit method), 54
get_limit() (awslimitchecker.limit.AwsLimit method), 54
get_limit_source() (awslimitchecker.limit.AwsLimit method), 55
get_limits() (awslimitchecker.checker.AwsLimitChecker method), 49
get_limits() (awslimitchecker.services.autoscaling._AutoscalingService method), 37
get_limits() (awslimitchecker.services.base._AwsService method), 39
get_limits() (awslimitchecker.services.ebs._EbsService method), 41
get_limits() (awslimitchecker.services.ec2._Ec2Service method), 42
get_limits() (awslimitchecker.services.elasticache._ElastiCacheService method), 44
get_limits() (awslimitchecker.services.elb._ElbService method), 45
get_limits() (awslimitchecker.services.rds._RDSService method), 46
get_limits() (awslimitchecker.services.vpc._VpcService method), 47
get_project_url() (awslimitchecker.checker.AwsLimitChecker method), 49
get_required_iam_policy() (awslimitchecker.checker.AwsLimitChecker method), 49
get_service_names() (awslimitchecker.checker.AwsLimitChecker method), 49
get_value() (awslimitchecker.limit.AwsLimitUsage method), 56
get_version() (awslimitchecker.checker.AwsLimitChecker method), 50
get_warnings() (awslimitchecker.limit.AwsLimit method), 55

I

iam_policy() (awslimitchecker.runner.Runner method), 57
invoke_with_throttling_retries() (in module awslimitchecker.utils), 60

L

list_defaults() (awslimitchecker.runner.Runner method), 57
list_limits() (awslimitchecker.runner.Runner method), 57

list_services() (awslimitchecker.runner.Runner method), 57

P

paginate_query() (in module awslimitchecker.utils), 61
parse_args() (awslimitchecker.runner.Runner method), 57
print_issue() (awslimitchecker.runner.Runner method), 57

R

required_iam_permissions() (awslimitchecker.services.autoscaling._AutoscalingService method), 37
required_iam_permissions() (awslimitchecker.services.base._AwsService method), 39
required_iam_permissions() (awslimitchecker.services.ebs._EbsService method), 41
required_iam_permissions() (awslimitchecker.services.ec2._Ec2Service method), 43
required_iam_permissions() (awslimitchecker.services.elasticache._ElastiCacheService method), 44
required_iam_permissions() (awslimitchecker.services.elb._ElbService method), 45
required_iam_permissions() (awslimitchecker.services.rds._RDSService method), 46
required_iam_permissions() (awslimitchecker.services.vpc._VpcService method), 47
Runner (class in awslimitchecker.runner), 57

S

service_name (awslimitchecker.services.autoscaling._AutoscalingService attribute), 37
service_name (awslimitchecker.services.base._AwsService attribute), 39
service_name (awslimitchecker.services.ebs._EbsService attribute), 41
service_name (awslimitchecker.services.ec2._Ec2Service attribute), 43
service_name (awslimitchecker.services.elasticache._ElastiCacheService attribute), 44
service_name (awslimitchecker.services.elb._ElbService attribute), 45
service_name (awslimitchecker.services.rds._RDSService attribute), 46
service_name (awslimitchecker.services.vpc._VpcService attribute), 47
service_name (awslimitchecker.trustedadvisor.TrustedAdvisor attribute), 59
set_limit_override() (awslimitchecker.checker.AwsLimitChecker method), 50
set_limit_override() (awslimitchecker.limit.AwsLimit method), 55
set_limit_override() (awslimitchecker.services.base._AwsService method), 39
set_limit_overrides() (awslimitchecker.checker.AwsLimitChecker method), 50
set_limit_overrides() (awslimitchecker.runner.Runner method), 57
set_threshold_override() (awslimitchecker.checker.AwsLimitChecker method), 50
set_threshold_override() (awslimitchecker.limit.AwsLimit method), 55
set_threshold_override() (awslimitchecker.services.base._AwsService method), 40
set_threshold_overrides() (awslimitchecker.checker.AwsLimitChecker method), 51
show_usage() (awslimitchecker.runner.Runner method), 57
SOURCE_API (in module awslimitchecker.limit), 56
SOURCE_DEFAULT (in module awslimitchecker.limit), 56
SOURCE_OVERRIDE (in module awslimitchecker.limit), 57
SOURCE_TA (in module awslimitchecker.limit), 57
StoreKeyValuePair (class in awslimitchecker.utils), 59

T

TrustedAdvisor (class in awslimitchecker.trustedadvisor), 58

U

update_limits() (awslimitchecker.trustedadvisor.TrustedAdvisor method), 59

V

version_str (awslimitchecker.version.AWSLimitCheckerVersion attribute), 62