awslimitchecker Documentation

Release 0.10.0

Jason Antman

Jun 26, 2017

Contents

Master: Develop: A script and python module to check your AWS service limits and usage, and warn when usage approaches limits.

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.

Full project documentation for the latest release is available at http://awslimitchecker.readthedocs.io/en/latest/.

CHAPTER 1

Status

It's gotten a bunch of downloads on PyPi (when the download counters used to work). As far as I know, it's stable and in use by some pretty large organizations.

Development status is being tracked on a board at waffle.io: https://waffle.io/jantman/awslimitchecker

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`.
- Optionally refresh Trusted Advisor "Service Limits" check before polling Trusted Advisor data, and optionally wait for the refresh to complete (up to an optional maximum time limit). See *Getting Started - Trusted Advisor* for more information.

CHAPTER 3

Requirements

- Python 2.6, 2.7, 3.3+.
- Python VirtualEnv and `pip` (recommended installation method; your OS/distribution should have packages for these)
- boto3 >= 1.2.3

CHAPTER 4

Installation and Usage

See *Getting Started*.

CHAPTER 5

Credentials

See *Credentials*.

CHAPTER 6

Getting Help and Asking Questions

See *Getting Help*.

For paid support and development options, please see the *Enterprise Support Agreements and Contract Development* section of the documentation.

There is also a gitter.im chat channel for support and discussion.

CHAPTER 7

Changelog

See *Changelog*.

CHAPTER 8

Contributions

Pull requests are most definitely welcome. Please cut them against the **develop** branch. For more information, see the *development documentation*. I'm also happy to accept contributions in the form of bug reports, feature requests, testing, etc.

CHAPTER 9

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.

CHAPTER 10

Contents

Getting Started

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`.
- Optionally refresh Trusted Advisor "Service Limits" check before polling Trusted Advisor data, and optionally wait for the refresh to complete (up to an optional maximum time limit). See *Getting Started - Trusted Advisor* for more information.

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. For more information on overriding thresholds, see *Python Usage / Setting a Threshold Override* as well as the documentation for *AwsLimitChecker.check_thresholds()* and *AwsLimitChecker.set_threshold_override()*.

Requirements

- Python 2.6, 2.7, 3.3+.
- Python VirtualEnv and `pip` (recommended installation method; your OS/distribution should have packages for these)
- boto3 >= 1.2.3

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
```

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 boto3's supported config files or set as environment variables. If your credentials are in the cross-SDK credentials file (`~/.aws/credentials`) under a named profile section, you can use credentials from that profile by specifying the `-P` / `--profile` command lint option. See boto3 config and this project's documentation for further information.

Please note that version 0.3.0 of awslimitchecker moved from using `boto` as its AWS API client to using `boto3`. This change is mostly transparent, but there is a minor change in how AWS credentials are handled. In `boto`, if the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables were set, and the region was not set explicitly via awslimitchecker, the AWS region would either be taken from the `AWS_DEFAULT_REGION` environment variable or would default to us-east-1, regardless of whether a configuration file (`~/.aws/credentials` or `~/.aws/config`) was present. With boto3, it appears that the default region from the configuration file will be used if present, regardless of whether the credentials come from that file or from environment variables.

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`.

In addition, when assuming a role STS, you can use a MFA device. simply specify the device's serial number with the `-M` / `--mfa-serial-number` option and a token generated by the device with the `-T` / `--mfa-token` option. STS credentials will be cached for the lifetime of the program.

Important Note on Session and Federation (Temporary) Credentials: The temporary credentials granted by the AWS IAM GetFederationToken and GetSessionToken API calls will throw errors when trying to access the IAM API (except for Session tokens, which will work for IAM API calls only if an MFA token is used). Furthermore, Federation tokens cannot make use of the STS AssumeRole functionality. If you attempt to use awslimitchecker with credentials generated by these APIs (commonly used by organizations to hand out limited-lifetime credentials), you will likely encounter errors when checking IAM limits. If this is acceptable, you can use these credentials by setting the `AWS_SESSION_TOKEN` environment variable in addition to `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`, or by otherwise configuring these credentials in a way that's supported by boto3 configuration.

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.

Trusted Advisor

awslimitchecker supports retrieving your current service limits via the Trusted Advisor "Service Limits" performance check , for limits which Trusted Advisor tracks (currently a subset of what awslimitchecker knows about). The results of this check may not be available via the API for all accounts; as of December 2016, the Trusted Advisor documentation states that while this check is available for all accounts, API access is only available to accounts with Business- or Enterprise-level support plans. If your account does not have Trusted Advisor access, the API call will result in a `SubscriptionRequiredException` and awslimitchecker will log a `Cannot check TrustedAdvisor` message at warning level.

Trusted Advisor information is important to awslimitchecker, however, as it provides the current service limit values for a number of limits that cannot be obtained any other way. While you can completely disable Trusted Advisor polling via the `--skip-ta` command-line option, you will then be left with default service limit values for many limits.

As of 0.7.0, awslimitchecker also supports programmatically refreshing the "Service Limits" Trusted Advisor check, in order to get updated limit values. If this is not done, the data provided by Trusted Advisor may not be updated unless a human does so via the AWS Console. The refresh logic operates in one of three modes, controlled by command-line options (these are also exposed in the Python API; see the "Internals" link below):

- `--ta-refresh-wait` - The check will be refreshed and awslimitchecker will poll every 30 seconds waiting for the refresh to complete (or until `ta_refresh_timeout` seconds have elapsed).
- `--ta-refresh-older INTEGER` - This operates like the `--ta-refresh-wait` option, but will only refresh the check if its current result data is at least `INTEGER` seconds old.
- `--ta-refresh-trigger` - The check will be refreshed and the program will continue on immediately, without waiting for the refresh to complete; this will almost certainly result in stale check results in the current run. However, this may be useful if you desire to keep `awslimitchecker` runs short, and run it on a regular schedule (i.e. if you run `awslimitchecker` every 6 hours, and are OK with Trusted Advisor check data being 6 hours old).

Additionally, there is a `--ta-refresh-timeout` option. If this is set (to an integer), refreshes of the check will time out after that number of seconds. If a timeout occurs, a message will be logged at error level, but the program will continue running (most likely using the old result data).

Important: It may take 30 to 60 *minutes* for the Service Limits check to refresh on large accounts. Please be aware of this when enabling the refresh options.

Using the check refresh options will require the `trustedadvisor:RefreshCheck` IAM permission.

See *Internals - Trusted Advisor* for technical information on the implementation of Trusted Advisor polling.

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*.

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 [SERVICE ...]]]
                       [--skip-service SKIP_SERVICE] [-s] [-l]
                       [--list-defaults] [-L LIMIT] [-u] [--iam-policy]
                       [-W WARNING_THRESHOLD] [-C CRITICAL_THRESHOLD]
                       [-P PROFILE_NAME] [-A STS_ACCOUNT_ID]
                       [-R STS_ACCOUNT_ROLE] [-E EXTERNAL_ID]
                       [-M MFA_SERIAL_NUMBER] [-T MFA_TOKEN] [-r REGION]
                       [--skip-ta]
                       [--ta-refresh-wait | --ta-refresh-trigger | --ta-refresh-older␣
↪TA_REFRESH_OLDER]
                       [--ta-refresh-timeout TA_REFRESH_TIMEOUT] [--no-color]
                       [-v] [-V]
Report on AWS service limits and usage via boto3, 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 [SERVICE [SERVICE ...]]
                        perform action for only the specified service name;
                        see -s|--list-services for valid names
  --skip-service SKIP_SERVICE
```

```
                      avoid performing actions for 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
-P PROFILE_NAME, --profile PROFILE_NAME
                      Name of profile in the AWS cross-sdk credentials file
                      to use credentials from; similar to the corresponding
                      awscli option
-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
-M MFA_SERIAL_NUMBER, --mfa-serial-number MFA_SERIAL_NUMBER
                      MFA Serial Number to use when assuming a role via STS
-T MFA_TOKEN, --mfa-token MFA_TOKEN
                      MFA Token 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
--ta-refresh-wait     If applicable, refresh all Trusted Advisor limit-
                      related checks, and wait for the refresh to complete
                      before continuing.
--ta-refresh-trigger  If applicable, trigger refreshes for all Trusted
                      Advisor limit-related checks, but do not wait for them
                      to finish refreshing; trigger the refresh and continue
                      on (useful to ensure checks are refreshed before the
                      next scheduled run).
--ta-refresh-older TA_REFRESH_OLDER
                      If applicable, trigger refreshes for all Trusted
                      Advisor limit-related checks with results more than
                      this number of seconds old. Wait for the refresh to
                      complete before continuing.
--ta-refresh-timeout TA_REFRESH_TIMEOUT
                      If waiting for TA checks to refresh, wait up to this
                      number of seconds before continuing on anyway.
```

```
  --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.
```

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
CloudFormation
EBS
EC2
ELB
(...)
Redshift
S3
SES
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
CloudFormation/Stacks                                   200
EBS/Active snapshots                                    10000
EBS/Active volumes                                      5000
(...)
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                          1000 (API)
AutoScaling/Launch configurations                        1400 (API)
CloudFormation/Stacks                                    1600 (API)
EBS/Active snapshots                                     30000 (TA)
EBS/Active volumes                                       10000 (TA)
(...)
VPC/Rules per network ACL                                20
VPC/Subnets per VPC                                      200
VPC/VPCs                                                 1000 (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                          1000 (API)
AutoScaling/Launch configurations                        1400 (API)
CloudFormation/Stacks                                    1600 (API)
EBS/Active snapshots                                     10000
EBS/Active volumes                                       5000
(...)
VPC/Rules per network ACL                                20
VPC/Subnets per VPC                                      200
VPC/VPCs                                                 5
```

Disabling Specific Services

The `--skip-service` option can be used to completely disable the specified service name(s) (as shown by `-s` / `--list-services`) for services that are problematic or you do not wish to query at all.

For example, you can check usage of all services _except_ for `Firehose` and `EC2`:

```
(venv)$ awslimitchecker --skip-service=Firehose --skip-service EC2
 WARNING:awslimitchecker.checker:Skipping service: Firehose
 WARNING:awslimitchecker.checker:Skipping service: EC2
 ... normal output ...
```

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                          718
AutoScaling/Launch configurations                        834
CloudFormation/Stacks                                    1352
EBS/Active snapshots                                     20337
EBS/Active volumes                                       1979
```

```
(...)
VPC/Rules per network ACL                                   max: acl-bde47dd9=6 (acl-
↪4bd96a2e=4, acl-3f36 (...)
VPC/Subnets per VPC                                         max: vpc-c89074a9=40 (vpc-
↪1e5e3c7b=1, vpc-ae7 (...)
VPC/VPCs                                                    17
```

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 configurations"=456 -l
AutoScaling/Auto Scaling groups                             321
AutoScaling/Launch configurations                           456
CloudFormation/Stacks                                       1600 (API)
EBS/Active snapshots                                        30000 (TA)
EBS/Active volumes                                          10000 (TA)
(...)
VPC/Rules per network ACL                                   20
VPC/Subnets per VPC                                         200
VPC/VPCs                                                    1000 (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
CloudFormation/Stacks                                       (limit 1600) WARNING: 1352
EC2/Security groups per VPC                                 (limit 500) CRITICAL: vpc-
↪c89074a9=960 WARNIN (...)
```

```
EC2/VPC security groups per elastic network interface  (limit 5) CRITICAL: eni-
↪8226ce61=5 WARNING: e (...)
ElastiCache/Parameter Groups                           (limit 20) WARNING: 18
ElasticBeanstalk/Application versions                  (limit 500) CRITICAL: 3375
ElasticBeanstalk/Applications                          (limit 25) CRITICAL: 213
ElasticBeanstalk/Environments                          (limit 200) CRITICAL: 534
RDS/VPC Security Groups                                (limit 5) WARNING: 4
S3/Buckets                                             (limit 100) CRITICAL: 488
VPC/NAT Gateways per AZ                                (limit 5) CRITICAL: us-east-
↪1d=9, us-east-1b= (...)
```

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/Security groups per VPC                            (limit 500) CRITICAL: vpc-
↪c89074a9=960
EC2/VPC security groups per elastic network interface  (limit 5) CRITICAL: eni-
↪8226ce61=5
ElasticBeanstalk/Application versions                  (limit 500) CRITICAL: 3375
ElasticBeanstalk/Applications                          (limit 25) CRITICAL: 213
ElasticBeanstalk/Environments                          (limit 200) CRITICAL: 534
S3/Buckets                                             (limit 100) CRITICAL: 488
VPC/NAT Gateways per AZ                                (limit 5) CRITICAL: us-east-
↪1d=9, us-east-1b= (...)
```

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.

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.

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.

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')
```

Refreshing Trusted Advisor Check Results

Trusted Advisor check refresh behavior is controlled by the `ta_refresh_mode` and `ta_refresh_timeout` parameters on the *`AwsLimitChecker`* constructor, which are passed through to the *`TrustedAdvisor`* constructor. See *Internals - Trusted Advisor* for details of their possible values and meanings.

The below example shows constructing an *`AwsLimitChecker`* class that will refresh Trusted Advisor limit checks only if their data is at least 6 hours (21600 seconds) old, and will allow up to 30 minutes (1800 seconds) for the refresh to complete (if it times out, awslimitchecker will continue on with the old data):

```
>>> import logging
>>> logging.basicConfig()
>>> logger = logging.getLogger()
>>>
>>> from awslimitchecker.checker import AwsLimitChecker
>>> c = AwsLimitChecker(ta_refresh_mode=21600, ta_refresh_timeout=1800)
```

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)
```

Setting a Threshold Override

`awslimitchecker` has two sets of thresholds, warning and critical (intended to be used to trigger different levels of alert/alarm or action). The default thresholds for warning and critical are 80% and 99%, respectively; these program-wide defaults can be overridden by passing the `warning_threshold` and/or `critical_threshold` arguments to the *`AwsLimitChecker`* class constructor.

It is also possible to override these values on a per-limit basis, using the AwsLimitChecker class's *set_threshold_override()* (single limit's threshold override) and *set_threshold_overrides()* (dict of overrides) methods. When setting threshold overrides, you can specify not only the percent threshold, but also a `count` of usage; any limits which have a usage of more than this number will be detected as a warning or critical, respectively.

To warn when our `EC2-Classic Elastic IPs` usage is above 50% (as opposed to the default of 80%) and store a critical alert when it's above 75% (as opposed to 99%):

```
>>> c.set_threshold_override('EC2', 'EC2-Classic Elastic IPs', warn_percent=50, crit_
↪percent=75)
```

Another use could be to warn when certain services are used at all. As of the time of writing, the i2.8xlarge instances cost USD $6.82/hour, or $163/day.

To report a critical status if *any* i2.8xlarge instances are running:

```
>>> c.set_threshold_override('EC2', 'Running On-Demand i2.8xlarge instances', crit_
↪count=1)
```

You do not need to also override the percent thresholds. Because of how *check_thresholds()* evaluates thresholds, *any* crossed threshold will be considered an error condition.

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 0x7f398db55a10>,
         '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)
```

Skipping Specific Services

You can completely disable all interaction with specific Services with the *remove_services()* method. This method takes a list of string Service names to remove from AwsLimitChecker's internal `services` dict, which will prevent those services from being queried or reported on.

To remove the Firehose and EC2 services:

```
c.remove_services(['Firehose', 'EC2'])
```

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()
```

Advanced Examples

For more examples, see docs/examples/README.rst on GitHub.

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=100)
CRITICAL:root:EC2 'Running On-Demand m3.medium instances' usage (53) exceeds critical␣
↪threshold (limit=20)
CRITICAL:root:EC2 'EC2-Classic Elastic IPs' usage (5) exceeds critical threshold␣
↪(limit=5)
>>> if have_critical:
...     raise SystemExit(1)
...
(awslimitchecker)$ echo $?
1
```

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",
        "cloudformation:DescribeAccountLimits",
        "cloudformation:DescribeStacks",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAddresses",
        "ec2:DescribeInstances",
        "ec2:DescribeInternetGateways",
        "ec2:DescribeNatGateways",
        "ec2:DescribeNetworkAcls",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeReservedInstances",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSecurityGroups",
```

```
        "ec2:DescribeSnapshots",
        "ec2:DescribeSpotDatafeedSubscription",
        "ec2:DescribeSpotFleetInstances",
        "ec2:DescribeSpotFleetRequestHistory",
        "ec2:DescribeSpotFleetRequests",
        "ec2:DescribeSpotInstanceRequests",
        "ec2:DescribeSpotPriceHistory",
        "ec2:DescribeSubnets",
        "ec2:DescribeVolumes",
        "ec2:DescribeVpcs",
        "elasticache:DescribeCacheClusters",
        "elasticache:DescribeCacheParameterGroups",
        "elasticache:DescribeCacheSecurityGroups",
        "elasticache:DescribeCacheSubnetGroups",
        "elasticbeanstalk:DescribeApplicationVersions",
        "elasticbeanstalk:DescribeApplications",
        "elasticbeanstalk:DescribeEnvironments",
        "elasticloadbalancing:DescribeLoadBalancers",
        "firehose:ListDeliveryStreams",
        "iam:GetAccountSummary",
        "rds:DescribeAccountAttributes",
        "rds:DescribeDBInstances",
        "rds:DescribeDBParameterGroups",
        "rds:DescribeDBSecurityGroups",
        "rds:DescribeDBSnapshots",
        "rds:DescribeDBSubnetGroups",
        "rds:DescribeEventSubscriptions",
        "rds:DescribeOptionGroups",
        "rds:DescribeReservedDBInstances",
        "redshift:DescribeClusterSnapshots",
        "redshift:DescribeClusterSubnetGroups",
        "s3:ListAllMyBuckets",
        "ses:GetSendQuota",
        "support:*",
        "trustedadvisor:Describe*",
        "trustedadvisor:RefreshCheck"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ],
  "Version": "2012-10-17"
}
```

Supported Limits

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
- CloudFormation
    - Stacks
- 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)
    - Running On-Demand c3.2xlarge instances
    - Running On-Demand c3.4xlarge instances
    - Running On-Demand c3.large instances
    - Running On-Demand c3.xlarge instances
    - Running On-Demand c4.2xlarge instances
    - Running On-Demand c4.4xlarge instances
    - Running On-Demand c4.large instances
    - Running On-Demand c4.xlarge instances
    - Running On-Demand m3.2xlarge instances
    - Running On-Demand m3.large instances
    - Running On-Demand m3.medium instances
    - Running On-Demand m3.xlarge instances
    - Running On-Demand m4.2xlarge instances
    - Running On-Demand m4.large instances
    - Running On-Demand m4.xlarge instances
    - Running On-Demand r3.2xlarge instances
    - Running On-Demand r3.4xlarge instances
    - Running On-Demand r3.large instances
    - Running On-Demand r3.xlarge instances
    - Running On-Demand r4.large instances
    - Running On-Demand t1.micro instances
    - Running On-Demand t2.large instances
    - Running On-Demand t2.medium instances
    - Running On-Demand t2.micro instances

  - Running On-Demand t2.nano instances
  - Running On-Demand t2.small instances
  - Running On-Demand t2.xlarge instances
  - VPC Elastic IP addresses (EIPs)
- ELB
  - Active load balancers
- IAM
  - Groups
  - Instance profiles
  - Policies
  - Roles
  - Server certificates
  - Users
- RDS
  - DB Cluster Parameter Groups
  - DB Clusters
  - DB instances
  - DB parameter groups
  - DB security groups
  - DB snapshots per user
  - Event Subscriptions
  - Max auths per security group
  - Read replicas per master
  - Storage quota (GB)
  - Subnet Groups
  - Subnets per Subnet Group
- SES
  - Daily sending quota
- VPC
  - Internet gateways
  - VPCs

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
- CloudFormation
  - Stacks
- EC2
  - Elastic IP addresses (EIPs)
  - Running On-Demand EC2 instances
  - VPC Elastic IP addresses (EIPs)
  - VPC security groups per elastic network interface
- IAM
  - Groups
  - Instance profiles
  - Policies
  - Policy Versions In Use
  - Roles
  - Server certificates
  - Users
- RDS
  - DB Cluster Parameter Groups
  - DB Clusters
  - DB instances
  - DB parameter groups
  - DB security groups
  - DB snapshots per user
  - Event Subscriptions
  - Max auths per security group
  - Option Groups
  - Read replicas per master
  - Reserved Instances
  - Storage quota (GB)
  - Subnet Groups
  - Subnets per Subnet Group
- SES
  - Daily sending quota

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

CloudFormation

Limit	Default
Stacks (TA) (API)	200

EBS

Limit	Default
Active snapshots (TA)	10000
Active volumes (TA)	5000
Cold (HDD) volume storage (GiB)	20480
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
Throughput Optimized (HDD) volume storage (GiB)	20480

EC2

Note on On-Demand vs Reserved Instances: The EC2 limits for "Running On-Demand" EC2 Instances apply only to On-Demand instances, not Reserved Instances. If you list all EC2 instances that are running in the Console or API, you'll get back instances of all types (On-Demand, Reserved, etc.). The value that awslimitchecker reports for Running On-Demand Instances current usage will *not* match the number of instances you see in the Console or API.

Limit	Default
Elastic IP addresses (EIPs) (TA) (API)	5
Max active spot fleets per region	1000
Max launch specifications per spot fleet	50
Max spot instance requests per region	20
Max target capacity for all spot fleets in region	5000
Max target capacity per spot fleet	3000
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 (TA)	20
Running On-Demand c3.4xlarge instances (TA)	20

Continued on next page

Table 10.1 – continued from previous page

Limit	Default
Running On-Demand c3.8xlarge instances	20
Running On-Demand c3.large instances (TA)	20
Running On-Demand c3.xlarge instances (TA)	20
Running On-Demand c4.2xlarge instances (TA)	20
Running On-Demand c4.4xlarge instances (TA)	10
Running On-Demand c4.8xlarge instances	5
Running On-Demand c4.large instances (TA)	20
Running On-Demand c4.xlarge instances (TA)	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 f1.16xlarge instances	20
Running On-Demand f1.2xlarge 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
Running On-Demand i3.16xlarge instances	2
Running On-Demand i3.2xlarge instances	2
Running On-Demand i3.4xlarge instances	2
Running On-Demand i3.8xlarge instances	2
Running On-Demand i3.large instances	2
Running On-Demand i3.xlarge instances	2
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 (TA)	20
Running On-Demand m3.large instances (TA)	20
Running On-Demand m3.medium instances (TA)	20
Running On-Demand m3.xlarge instances (TA)	20
Running On-Demand m4.10xlarge instances	5
Running On-Demand m4.16xlarge instances	5
Running On-Demand m4.2xlarge instances (TA)	20
Running On-Demand m4.4xlarge instances	10
Running On-Demand m4.large instances (TA)	20
Running On-Demand m4.xlarge instances (TA)	20
Running On-Demand p2.16xlarge instances	1

Continued on next page

Table 10.1 – continued from previous page

Limit	Default
Running On-Demand p2.8xlarge instances	1
Running On-Demand p2.xlarge instances	1
Running On-Demand r3.2xlarge instances (TA)	20
Running On-Demand r3.4xlarge instances (TA)	10
Running On-Demand r3.8xlarge instances	5
Running On-Demand r3.large instances (TA)	20
Running On-Demand r3.xlarge instances (TA)	20
Running On-Demand r4.16xlarge instances	20
Running On-Demand r4.2xlarge instances	20
Running On-Demand r4.4xlarge instances	20
Running On-Demand r4.8xlarge instances	20
Running On-Demand r4.large instances (TA)	20
Running On-Demand r4.xlarge instances	20
Running On-Demand t1.micro instances (TA)	20
Running On-Demand t2.2xlarge instances	20
Running On-Demand t2.large instances (TA)	20
Running On-Demand t2.medium instances (TA)	20
Running On-Demand t2.micro instances (TA)	20
Running On-Demand t2.nano instances (TA)	20
Running On-Demand t2.small instances (TA)	20
Running On-Demand t2.xlarge instances (TA)	20
Running On-Demand x1.16xlarge instances	20
Running On-Demand x1.32xlarge instances	20
Security groups per VPC	500
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
Nodes	100
Nodes per Cluster	20
Parameter Groups	20
Security Groups	50
Subnet Groups	50
Subnets per subnet group	20

ElasticBeanstalk

Limit	Default
Application versions	500
Applications	25
Environments	200

Firehose

Limit	Default
Delivery streams per region	20

IAM

Limit	Default
Groups (TA) (API)	100
Instance profiles (TA) (API)	100
Policies (TA) (API)	1000
Policy Versions In Use (API)	10000
Roles (TA) (API)	250
Server certificates (TA) (API)	20
Users (TA) (API)	5000

RDS

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

Redshift

Limit	Default
Redshift manual snapshots	20
Redshift subnet groups	20

S3

Limit	Default
Buckets	100

SES

Limit	Default
Daily sending quota (TA) (API)	200

VPC

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

Getting Help

If you have a quick question or need some simple assistance, you can try the gitter.im chat channel.

Enterprise Support Agreements and Contract Development

For Commercial or Enterprise support agreements for awslimitchecker, or for paid as-needed feature development or bug fixes, please contact Jason Antman at jason@jasonantman.com.

Reporting Bugs and Asking Questions

Questions, bug reports and feature requests are happily accepted via the GitHub Issue Tracker. Pull requests are welcome; see the *Development* documentation for information on PRs. Issues that don't have an accompanying pull request will be worked on as my time and priority allows, and 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.

Guidelines for Reporting Issues

Opening a new issue on GitHub should pre-populate the issue description with a template of the following:

Feature Requests

If your feature request is for support of a service or limit not currently supported by awslimitchecker, you can simply title the issue `add support for <name of service, or name of service and limit>` and add a simple description. For anything else, please follow these guidelines:

1. Describe in detail the feature you would like to see implemented, especially how it would work from a user perspective and what benefits it adds. Your description should be detailed enough to be used to determine if code written for the feature adequately solves the problem.
2. Describe one or more use cases for why this feature will be useful.
3. Indicate whether or not you will be able to assist in testing pre-release code for the feature.

Bug Reports

When reporting a bug in awslimitchecker, please provide all of the following information, as well as any additional details that may be useful in reproducing or fixing the issue:

1. awslimitchecker version, as reported by `awslimitchecker --version`.
2. How was awslimitchecker installed (provide as much detail as possible, ideally the exact command used and whether it was installed in a virtualenv or not).
3. The output of `python --version` and `virtualenv --version` in the environment that awslimitchecker is running in.
4. Your operating system type and version.
5. The output of awslimitchecker, run with the `-vv` (debug-level output) flag that shows the issue.
6. The output that you expected (what's wrong).
7. If the bug/issue is related to TrustedAdvisor, which support contract your account has.
8. Whether or not you are willing and able to assist in testing pre-release code intended to fix the issue.

Development

Any and all contributions to awslimitchecker are welcome. Guidelines for submitting code contributions in the form of pull requests on GitHub can be found below. For guidelines on submitting bug reports or feature requests, please see the *Getting Help* documentation. For any contributions that don't fall into the above categories, please open an issue for further assistance.

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.

Pull Request Guidelines

- All pull requests should be made against the `develop` branch, NOT master.

- If you have not contributed to the project before, all pull requests must include a statement that your contribution is being made under the same license as the awslimitchecker project (or any subsequent version of that license if adopted by awslimitchecker), may perpetually be included in and distributed with awslimitchecker, and that you have the legal power to agree to these terms.
- Code should conform to the *Guidelines* below.
- If you have difficulty writing tests for the code, feel free to ask for help or submit the PR without tests. This will increase the amount of time it takes to get merged, but I'd rather write tests for your code than write all the code myself.
- You've rebuilt the documentation using `tox -e docs`

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 and install development dependencies:

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

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.

Guidelines

- pep8 compliant with some exceptions (see pytest.ini)
- 100% test coverage with pytest (with valid tests)
- Complete, correctly-formatted documentation for all classes, functions and methods.
- Connections to the AWS services should only be made by the class's *`connect()`* and *`connect_resource()`* methods, inherited from the *`Connectable`* mixin.
- 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.

Adding New Limits and Checks to Existing Services

First, note that all calls to boto3 client ("low-level") methods that return a dict response that can include 'NextToken' or another pagination marker, should be called through *paginate_dict()* with the appropriate parameters.

1. Add a new *AwsLimit* instance to the return value of the Service class's *get_limits()* method. If Trusted Advisor returns data for this limit, be sure the service and limit names match those returned by Trusted Advisor.
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 `self.conn` and/or `self.resource_conn` 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.

In cases where the AWS service API has a different name than what is reported by Trusted Advisor, or legacy cases where Trusted Advisor support is retroactively added to a limit already in awslimitchecker, you must pass the `ta_service_name` and `ta_limit_name` parameters to the *AwsLimit* constructor, specifying the string values that are returned by Trusted Advisor.

Adding New Services

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

First, note that all calls to boto3 client ("low-level") methods that return a dict response that can include 'NextToken' or another pagination marker, should be called through *paginate_dict()* with the appropriate parameters.

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. Be sure to set the class's `api_name` attribute to the correct name of the AWS service API (i.e. the parameter passed to boto3.client). This string can typically be found at the top of the Service page in the boto3 docs.
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. Run all tox jobs, or at least one python version, docs and coverage.
11. Commit the updated documentation to the repository.
12. 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.
13. Submit your pull request.

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*.

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.

Integration Testing

Integration tests are automatically run in TravisCI for all **non-pull request** branches. You can run them manually from your local machine using:

```
tox -r -e integration,integration3
```

These tests simply run `awslimitchecker`'s CLI script for both usage and limits, for all services and each service individually. Note that this covers a very small amount of the code, as the account that I use for integration tests has virtually no resources in it.

If integration tests fail, check the required IAM permissions. The IAM user for Travis integration tests is configured via Terraform, which must be re-run after policy changes.

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.

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.

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.

Release Checklist

Note that to perform releases, you will need:

- Your Github access token exported as the `GITHUB_TOKEN` environment variable.
- pandoc installed on your local machine and in your `PATH`.

1. Open an issue for the release (the checklist below may help); cut a branch off `develop` for that issue.
2. Build docs locally (`tox -e localdocs`) and ensure they're current; commit any changes.
3. Run `dev/terraform.py` in the awslimitchecker source directory to update the integration test user's IAM policy with what is actually being reported by the current code.
4. Ensure that Travis tests are passing in all environments.
5. Ensure that test coverage is no less than the last release (ideally, 100%).
6. Build docs for the branch (locally) and ensure they look correct. Commit any changes.
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, and that any breaking changes or new required IAM permissions are explicitly mentioned.
8. Run `dev/release.py gist` to convert the CHANGES.rst entry for the current version to Markdown and upload it as a Github Gist. View the gist and ensure that the Markdown rendered properly and all links are valid. Iterate on this until the rendered version looks correct.
9. Commit all changes, mention the issue in the commit, and push to GitHub.
10. Confirm that README.rst renders correctly on GitHub.
11. Upload package to testpypi, confirm that README.rst renders correctly.
    - Make sure your ~/.pypirc file is correct (a repo called `test` for https://testpypi.python.org/pypi).
    - `rm -Rf dist`
    - `python setup.py register -r https://testpypi.python.org/pypi`
    - `python setup.py sdist bdist_wheel`
    - `twine upload -r test dist/*`
    - Check that the README renders at https://testpypi.python.org/pypi/awslimitchecker
12. Create a pull request for the release to be merged into master. Upon successful Travis build, merge it.
13. Tag the release in Git, push tag to GitHub:
    - tag the release with a signed tag: `git tag -s -a X.Y.Z -m 'X.Y.Z released YYYY-MM-DD'`
    - Verify the signature on the tag, just to be sure: `git tag -v X.Y.Z`
    - push the tag to GitHub: `git push origin X.Y.Z`
14. Upload package to live pypi:

    - `twine upload dist/*`

15. make sure any GH issues fixed in the release were closed.
16. merge master back into develop
17. Run `dev/release.py release` to create the release on GitHub.
18. Ensure that the issues are moved to Done on the waffle.io board
19. Blog, tweet, etc. about the new version.

Release Issue Template

Issue title: `x.y.z Release`

Issue content:

```
* [ ] Cut a branch off ``develop`` for this issue.
* [ ] Build docs locally (``tox -e localdocs``) and ensure they're current; commit␣
↪any changes.
* [ ] Run ``dev/terraform.py`` in the awslimitchecker source directory to update the␣
↪integration test user's IAM policy with what is actually being reported by the␣
↪current code.
* [ ] Ensure that Travis tests are passing in all environments.
* [ ] Ensure that test coverage is no less than the last release (ideally, 100%).
* [ ] Build docs for the branch (locally) and ensure they look correct. Commit any␣
↪changes.
* [ ] 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, and that any breaking changes or new required␣
↪IAM permissions are explicitly mentioned.
* [ ] Run ``dev/release.py gist`` to convert the CHANGES.rst entry for the current␣
↪version to Markdown and upload it as a Github Gist. View the gist and ensure that␣
↪the Markdown rendered properly and all links are valid. Iterate on this until the␣
↪rendered version looks correct.
* [ ] Commit all changes, mention the issue in the commit, and push to GitHub.
* [ ] Confirm that README.rst renders correctly on GitHub.
* [ ] Upload package to testpypi, confirm that README.rst renders correctly.

   * Make sure your ~/.pypirc file is correct (a repo called ``test`` for https://
↪testpypi.python.org/pypi).
   * ``rm -Rf dist``
   * ``python setup.py register -r https://testpypi.python.org/pypi``
   * ``python setup.py sdist bdist_wheel``
   * ``twine upload -r test dist/*``
   * Check that the README renders at https://testpypi.python.org/pypi/awslimitchecker

* [ ] Create a pull request for the release to be merged into master. Upon successful␣
↪Travis build, merge it.
* [ ] Continue at [#13 on the Release Checklist](http://awslimitchecker.readthedocs.
↪io/en/develop/development.html#release-checklist).
```

Internals

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 (`boto3`). 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 boto3 client ("low-level") methods that return a dict response that can include 'NextToken' or another pagination marker, should be called through *paginate_dict()* with the appropriate parameters.

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*.

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 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.

When *AwsLimitChecker* initializes *TrustedAdvisor*, it passes in the `self.services` dictionary of all services and limits. At initialization time, *TrustedAdvisor* iterates all services and limits, and builds a new dictionary mapping the limit objects by the return values of their *ta_service_name()* and *ta_limit_name()* properties. This allows limits to override the Trusted Advisor service and limit name that their data comes from. In the default case, their service and limit names will be used as they are set in the awslimitchecker code, and limits which have matching Trusted Advisor data will be automatically populated.

In the *TrustedAdvisor* class's *_poll()* method, *_get_refreshed_check_result()* is used to retrieve the check result data from Trusted Advisor. This method also implements the check refresh logic. See the comments in the source code for the specific logic. There are three methods of refreshing checks (refresh modes), which are controlled by the `ta_refresh_mode` parameter to *TrustedAdvisor*:

- If `ta_refresh_mode` is the string "wait", the check will be refreshed and awslimitchecker will poll for the refresh result every 30 seconds, waiting for the refresh to complete (or until `ta_refresh_timeout` seconds have elapsed). This is exposed via the CLI as the `--ta-refresh-wait` option.
- If `ta_refresh_mode` is an integer, it will operate like the "wait" mode above, but only if the current result data for the check is more than `ta_refresh_mode` seconds old. This is exposed via the CLI as the `--ta-refresh-older` option.
- If `ta_refresh_mode` is the string "trigger", the check will be refreshed and the program will continue on immediately, without waiting for the refresh to complete; this will almost certainly result in stale check results in the current run. However, this may be useful if you desire to keep `awslimitchecker` runs short, and run it on a regular schedule (i.e. if you run `awslimitchecker` every 6 hours, and are OK with Trusted Advisor check data being 6 hours old). This is exposed via the CLI as the `--ta-refresh-trigger` option.

Additionally, *`TrustedAdvisor`* has a `ta_refresh_timeout` parameter. If this is set to a non-`None` value (an integer), refreshes of the check will time out after that number of seconds. If a timeout occurs, a message will be logged at error level, but the program will continue running (most likely using the old result data). This parameter is exposed via the CLI as the `--ta-refresh-timeout` option.

Important: It may take 30 to 60 *minutes* for the Service Limits check to refresh on large accounts. Please be aware of this when enabling the refresh options.

Using the check refresh options will require the `trustedadvisor:RefreshCheck` IAM permission.

For use via Python, these same parameters (`ta_refresh_mode` and `ta_refresh_timeout`) are exposed as parameters on the *`AwsLimitChecker`* constructor.

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.

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

Threshold Overrides

For more information on overriding thresholds, see *Python Usage / Setting a Threshold Override* as well as the documentation for *`AwsLimitChecker.check_thresholds()`* and *`AwsLimitChecker.set_threshold_override()`*.

awslimitchecker

awslimitchecker package

Subpackages

awslimitchecker.services package

Submodules

awslimitchecker.services.autoscaling module

class awslimitchecker.services.autoscaling.**_AutoscalingService**(*warning_threshold*, *critical_threshold*, *boto_connection_kwargs={}*)

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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* 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 = 30

_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`.

api_name = 'autoscaling'

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*, *boto_connection_kwargs={}*)
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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* string to use when assuming a role via STS.

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

__init__(*warning_threshold*, *critical_threshold*, *boto_connection_kwargs={}*)
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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* 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 = 30

_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

api_name = 'baseclass'

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*

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.

If the boto3 method being called returns a dict response that can include 'NextToken' or another pagination marker, it should be called through *paginate_dict()* with the appropriate parameters.

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.cloudformation module

class awslimitchecker.services.cloudformation.**_CloudformationService**(*warning_threshold*, *critical_threshold*, *boto_connection_kwargs={}*)

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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* string to use when assuming a role via STS.

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.cloudformation'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 30

_abc_registry = <_weakrefset.WeakSet object>

_update_limits_from_api()
Call the service's API action to retrieve limit/quota information, and update AwsLimit objects in `self.limits` with this information.

api_name = 'cloudformation'

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 = 'CloudFormation'

awslimitchecker.services.ebs module

class awslimitchecker.services.ebs.**_EbsService**(*warning_threshold*, *critical_threshold*, *boto_connection_kwargs={}*)
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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* 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 = 30

_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

api_name = ‘ec2’

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*, *boto_connection_kwargs={}*)
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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.

- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* 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 = 30

_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

_find_usage_spot_fleets()
calculate spot fleet request usage and update Limits

_find_usage_spot_instances()
calculate spot instance request usage and update Limits

_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_limits_spot()
Return a dict of limits for spot requests 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`.

api_name = 'ec2'

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*, *boto_connection_kwargs={}*)
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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* 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 = 30

_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

api_name = 'elasticache'

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.elasticbeanstalk module

class awslimitchecker.services.elasticbeanstalk.**_ElasticBeanstalkService**(*warning_threshold*, *critical_threshold*, *boto_connection_kwargs={}*)

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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* string to use when assuming a role via STS.

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.elasticbeanstalk'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 30

_abc_registry = <_weakrefset.WeakSet object>

_find_usage_application_versions()
find usage for ElasticBeanstalk application verions

_find_usage_applications()
find usage for ElasticBeanstalk applications

_find_usage_environments()
find usage for ElasticBeanstalk environments

api_name = 'elasticbeanstalk'

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 = 'ElasticBeanstalk'

awslimitchecker.services.elb module

class awslimitchecker.services.elb.**_ElbService**(*warning_threshold*, *critical_threshold*, *boto_connection_kwargs={}*)

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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* 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 = 30

_abc_registry = <_weakrefset.WeakSet object>

api_name = 'elb'

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.firehose module

class awslimitchecker.services.firehose.**_FirehoseService**(*warning_threshold*, *critical_threshold*, *boto_connection_kwargs={}*)
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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* string to use when assuming a role via STS.

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.firehose'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 30

_abc_registry = <_weakrefset.WeakSet object>

_find_delivery_streams()

api_name = 'firehose'

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 = 'Firehose'

awslimitchecker.services.iam module

class awslimitchecker.services.iam.**_IamService**(*warning_threshold*, *critical_threshold*, *boto_connection_kwargs={}*)
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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* string to use when assuming a role via STS.

API_TO_LIMIT_NAME = {'Groups': 'Groups', 'Users': 'Users', 'Roles': 'Roles', 'PolicyVersionsInUse': 'Policy Versions

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.iam'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 30

_abc_registry = <_weakrefset.WeakSet object>

_update_limits_from_api()
Call the service's API action to retrieve limit/quota information, and update AwsLimit objects in `self.limits` with this information.

api_name = 'iam'

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 = 'IAM'

awslimitchecker.services.rds module

class awslimitchecker.services.rds.**_RDSService**(*warning_threshold*, *critical_threshold*, *boto_connection_kwargs={}*)
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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.

- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* string to use when assuming a role via STS.

API_NAME_TO_LIMIT = {'DBSubnetGroups': 'Subnet Groups', 'OptionGroups': 'Option Groups', 'ReservedDBInstan

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.rds'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 30

_abc_registry = <_weakrefset.WeakSet object>

_find_usage_instances()
find usage for DB Instances and related limits

_find_usage_security_groups()
find usage for security groups

_find_usage_subnet_groups()
find usage for subnet groups

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

We ignore the usage information from the API,

api_name = 'rds'

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.redshift module

class awslimitchecker.services.redshift.**_RedshiftService**(*warning_threshold*, *critical_threshold*, *boto_connection_kwargs={}*)

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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* string to use when assuming a role via STS.

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.redshift'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 30

_abc_registry = <_weakrefset.WeakSet object>

_find_cluster_manual_snapshots()

_find_cluster_subnet_groups()

api_name = 'redshift'

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 = 'Redshift'

awslimitchecker.services.s3 module

class awslimitchecker.services.s3.**_S3Service**(*warning_threshold*, *critical_threshold*, *boto_connection_kwargs={}*)

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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* string to use when assuming a role via STS.

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.s3'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 30

_abc_registry = <_weakrefset.WeakSet object>

api_name = 's3'

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 = 'S3'

awslimitchecker.services.ses module

class awslimitchecker.services.ses.**_SesService**(*warning_threshold*, *critical_threshold*, *boto_connection_kwargs={}*)

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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* string to use when assuming a role via STS.

__abstractmethods__ = frozenset([])

__module__ = 'awslimitchecker.services.ses'

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 30

_abc_registry = <_weakrefset.WeakSet object>

_update_limits_from_api()

Call the service's API action to retrieve limit/quota information, and update AwsLimit objects in `self.limits` with this information.

api_name = 'ses'

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 = 'SES'

awslimitchecker.services.vpc module

class awslimitchecker.services.vpc.**_VpcService**(*warning_threshold*, *critical_threshold*, *boto_connection_kwargs={}*)
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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* 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 = 30

_abc_registry = <_weakrefset.WeakSet object>

_find_usage_ACLs()
find usage for ACLs

_find_usage_gateways()
find usage for Internet Gateways

_find_usage_nat_gateways(*subnet_to_az*)
find usage for NAT Gateways

Parameters **subnet_to_az** (*dict*) – dict mapping subnet ID to AZ

_find_usage_route_tables()
find usage for route tables

_find_usage_subnets()
find usage for Subnets; return dict of SubnetId to AZ

_find_usage_vpcs()
find usage for VPCs

api_name = 'ec2'

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*, *profile_name=None*, *account_id=None*, *account_role=None*, *region=None*, *external_id=None*, *mfa_serial_number=None*, *mfa_token=None*, *ta_refresh_mode=None*, *ta_refresh_timeout=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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* string to use when assuming a role via STS.
- **ta_refresh_mode** (`str` or `int` or `None`) – How to handle refreshing Trusted Advisor checks; this is either None (do not refresh at all), the string "wait" (trigger refresh of all limit-related checks and wait for the refresh to complete), the string "trigger" (trigger refresh of all limit-related checks but do not wait for the refresh to complete), or an integer, which causes any limit-related checks more than this number of seconds old to be refreshed, waiting for the refresh to complete. Note that "trigger" will likely result in the current run getting stale data, but the check being refreshed in time for the next run.
- **ta_refresh_timeout** (`int` or `None`) – If `ta_refresh_mode` is "wait" or an integer (any mode that will wait for the refresh to complete), if this parameter is not None, only wait up to this number of seconds for the refresh to finish before continuing on anyway.

__dict__ = dict_proxy({'set_threshold_overrides': <function set_threshold_overrides>, '__module__': 'awslimitchecke

__init__(*warning_threshold=80*, *critical_threshold=99*, *profile_name=None*, *account_id=None*, *account_role=None*, *region=None*, *external_id=None*, *mfa_serial_number=None*, *mfa_token=None*, *ta_refresh_mode=None*, *ta_refresh_timeout=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.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* string to use when assuming a role via STS.
- **ta_refresh_mode** (str or int or None) – How to handle refreshing Trusted Advisor checks; this is either None (do not refresh at all), the string "wait" (trigger refresh of all limit-related checks and wait for the refresh to complete), the string "trigger" (trigger refresh of all limit-related checks but do not wait for the refresh to complete), or an integer, which causes any limit-related checks more than this number of seconds old to be refreshed, waiting for the refresh to complete. Note that "trigger" will likely result in the current run getting stale data, but the check being refreshed in time for the next run.
- **ta_refresh_timeout** (int or None) – If ta_refresh_mode is "wait" or an integer (any mode that will wait for the refresh to complete), if this parameter is not None, only wait up to this number of seconds for the refresh to finish before continuing on anyway.

__module__ = 'awslimitchecker.checker'

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

_boto_conn_kwargs
Generate keyword arguments for boto3 connection functions.

If self.account_id is defined, this will call *_get_sts_token()* to get STS token credentials using boto3.STS.Client.assume_role and include those credentials in the return value.

If self.profile_name is defined, this will call *boto3.Session() <http://boto3.readthedocs.io/en/latest/reference/core/session.html>* with that profile and include those credentials in the return value.

Returns keyword arguments for boto3 connection functions

Return type dict

_get_sts_token()
Assume a role via STS and return the credentials.

First connect to STS via boto3.client(), then assume a role using boto3.STS.Client.assume_role using self.account_id and self.account_role (and optionally self.external_id, self.mfa_serial_number, self.mfa_token). Return the resulting *ConnectableCredentials* object.

Returns STS assumed role credentials

Return type *ConnectableCredentials*

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()*).

See *AwsLimit.check_thresholds()*.

Parameters

- **service** (*list*) – the name(s) of one or more service(s) 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 `boto3` 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 `list` service names to get) – list of *_AwsService* name(s), 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** (*list*) – the name(s) of one or more services 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

remove_services(*services_to_remove=[]*)

Remove all service names specified in `services_to_remove` from `self.services`. This allows explicitly removing certain services from ever being checked or otherwise handled.

By default, the various methods that work on Services (i.e. *get_limits()*, *find_usage()* and *check_thresholds()*) operate on either all known services, or one specified service name at a time. This method allows you to remove one or more problematic or undesirable services from the dict of all services, and then operate on the remaining ones.

Parameters **services_to_remove** – the name(s) of one or more services to permanently exclude from future calls to this instance

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 *_AwsService.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 *_AwsService.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 `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 *AwsLimit.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 *AwsLimit.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({'__module__': 'awslimitchecker.connectable', '__doc__': '\n Mix-in helper class for connecting

__module__ = 'awslimitchecker.connectable'

__weakref__

list of weak references to the object (if defined)

connect()
Connect to an AWS API via boto3 low-level client and set `self.conn` to the boto3.client object (a `botocore.client.*` instance). If `self.conn` is not None, do nothing. This connects to the API name given by `self.api_name`.

Returns None

connect_resource()
Connect to an AWS API via boto3 high-level resource connection and set `self.resource_conn` to the boto3.resource object (a `boto3.resources.factory.*.ServiceResource` instance). If `self.resource_conn` is not None, do nothing. This connects to the API name given by `self.api_name`.

Returns None

class awslimitchecker.connectable.**ConnectableCredentials**(*creds_dict*)
Bases: object

boto's (2.x) `boto.sts.STSConnection.assume_role()` returns a `boto.sts.credentials.Credentials` object, but boto3's boto3.sts.STSConnection.assume_role just returns a dict. This class provides a compatible interface for boto3.

We also maintain an `account_id` attribute that can be set to the account ID, to ensure that credentials are updated when switching accounts.

__dict__ = dict_proxy({'__dict__': <attribute '__dict__' of 'ConnectableCredentials' objects>, '__module__': 'awslimi

__init__(*creds_dict*)

__module__ = 'awslimitchecker.connectable'

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

awslimitchecker.limit module

class awslimitchecker.limit.**AwsLimit**(*name*, *service*, *default_limit*, *def_warning_threshold*, *def_critical_threshold*, *limit_type=None*, *limit_subtype=None*, *ta_service_name=None*, *ta_limit_name=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** (*_AwsService*) – 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** (*str*) – resource sub-type for this limit, if applicable, such as "t2.micro" or "SecurityGroup"
- **ta_service_name** (*str*) – The service name returned by Trusted Advisor for this limit, if different from the name of `service`
- **ta_limit_name** (*str*) – The limit name returned by Trusted Advisor for this limit, if different from `name`.

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*, *ta_service_name=None*, *ta_limit_name=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** (*_AwsService*) – 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** (*str*) – resource sub-type for this limit, if applicable, such as "t2.micro" or "SecurityGroup"
- **ta_service_name** (*str*) – The service name returned by Trusted Advisor for this limit, if different from the name of `service`
- **ta_limit_name** (*str*) – The limit name returned by Trusted Advisor for this limit, if different from `name`.

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

_set_ta_unlimited()
Set state to indicate that TrustedAdvisor reports this limit as having no maximum (unlimited).

This method should only be called by *TrustedAdvisor*.

check_thresholds()
Check this limit's current usage against the specified default thresholds, and any custom theresholds that have been set on the class 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.

Note This function returns False if *any* thresholds were crossed. Please be aware of this when setting threshold overrides to suppress alerts. Each threshold (warn_percent, warn_count, crit_percent, crit_count) that has been set is evaluated individually and the result appended to a list of warnings or criticals, respectively. If *any* of these evaluations failed, the method returns False.

Returns False if any thresholds were 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. None is returned for limits that are explicitly unlimited.

Returns effective limit value, `int` or `None`

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.

Note: The percent thresholds (`warn_percent` and `crit_percent`) have default values that are set globally for awslimitchecker, unlike the count thresholds. When setting threshold overrides to quiet or suppress alerts for a limit, you **must** set the percent thresholds. If you only set overrides for the `count`

thresholds, the percent thresholds will continue to be evaluated at their awslimitchecker-wide default, and likely prevent alerts from being suppressed.

see *check_thresholds()* for further information on threshold evaluation.

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

ta_limit_name

Return the effective Trusted Advisor limit name that this limit's data will have. This should be `self._ta_limit_name` if set, otherwise `self.name`.

Returns Trusted Advisor limit data name

Return type str

ta_service_name

Return the effective Trusted Advisor service name that this limit's data will have. This should be `self._ta_service_name` if set, otherwise the name of `self.service`.

Returns Trusted Advisor service data name

Return type str

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__>, '__module__': 'awslimitchecker.limit', '__weakref__': <attribute

__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>, 'parse_args

__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**(*all_services*, *boto_connection_kwargs*, *ta_refresh_mode=None*, *ta_refresh_timeout=None*)
Bases: *awslimitchecker.connectable.Connectable*

Class to contain all TrustedAdvisor-related logic.

Parameters

- **all_services** (*dict*) – *AwsLimitChecker* `services` dictionary.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* string to use when assuming a role via STS.
- **ta_refresh_mode** (`str` or `int` or `None`) – How to handle refreshing Trusted Advisor checks; this is either None (do not refresh at all), the string "wait" (trigger refresh of all limit-related checks and wait for the refresh to complete), the string "trigger" (trigger refresh of all limit-related checks but do not wait for the refresh to complete), or an integer, which causes any limit-related checks more than this number of seconds old to be refreshed, waiting for the refresh to complete. Note that "trigger" will likely result in the current run getting stale data, but the check being refreshed in time for the next run.
- **ta_refresh_timeout** (`int` or `None`) – If `ta_refresh_mode` is "wait" or an integer (any mode that will wait for the refresh to complete), if this parameter is not None, only wait up to this number of seconds for the refresh to finish before continuing on anyway.

__init__(*all_services*, *boto_connection_kwargs*, *ta_refresh_mode=None*, *ta_refresh_timeout=None*)

Class to contain all TrustedAdvisor-related logic.

Parameters

- **all_services** (*dict*) – *AwsLimitChecker* `services` dictionary.
- **profile_name** (*str*) – The name of a profile in the cross-SDK shared credentials file for boto3 to retrieve AWS credentials from.
- **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.
- **mfa_serial_number** (*str*) – (optional) the *MFA Serial Number* string to use when assuming a role via STS.
- **mfa_token** (*str*) – (optional) the *MFA Token* string to use when assuming a role via STS.
- **ta_refresh_mode** (`str` or `int` or `None`) – How to handle refreshing Trusted Advisor checks; this is either None (do not refresh at all), the string "wait" (trigger refresh of all limit-related checks and wait for the refresh to complete), the string "trigger" (trigger refresh of all limit-related checks but do not wait for the refresh to complete), or an integer, which causes any limit-related checks more than this number of seconds old to be

refreshed, waiting for the refresh to complete. Note that "trigger" will likely result in the current run getting stale data, but the check being refreshed in time for the next run.

- **ta_refresh_timeout** (int or None) – If ta_refresh_mode is "wait" or an integer (any mode that will wait for the refresh to complete), if this parameter is not None, only wait up to this number of seconds for the refresh to finish before continuing on anyway.

__module__ = 'awslimitchecker.trustedadvisor'

_can_refresh_check(*check_id*)
Determine if the given check_id can be refreshed yet.

Parameters **check_id** (*str*) – the Trusted Advisor check ID

Returns whether or not the check can be refreshed yet

Return type bool

_get_check_result(*check_id*)
Directly wrap Support.Client.describe_trusted_advisor_check_result(); return a 2-tuple of the result dict and the last refresh DateTime.

Parameters **check_id** (*str*) – the Trusted Advisor check ID

Returns 2-tuple of (result dict, last refresh DateTime). If the last refresh time can't be parsed from the response, the second element will be None.

Return type tuple

_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

_get_refreshed_check_result(*check_id*)
Given the check_id, return the dict of Trusted Advisor check results. This handles refreshing the Trusted Advisor check, if desired, according to self.refresh_mode and self.refresh_timeout.

Parameters **check_id** (*str*) – the Trusted Advisor check ID

Returns dict check result. The return value of Support.Client.describe_trusted_advisor_check_result()

Return type dict

_make_ta_service_dict()
Build our service and limits dict. This is laid out identical to self.all_services, but keys limits by their ta_service_name and ta_limit_name properties.

Returns dict of TA service names to TA limit names to AwsLimit objects.

_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,
```

```
    }
}
```

_poll_for_refresh(*check_id*)
Given a Trusted Advisor check_id that has just been refreshed, poll until the refresh is complete. Once complete, return the check result.

Parameters **check_id** (*str*) – the Trusted Advisor check ID

Returns dict check result. The return value of `Support.Client.describe_trusted_advisor_check_result()`

Return type dict

_update_services(*ta_results*)
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

api_name = 'support'

service_name = 'TrustedAdvisor'

update_limits()
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.trustedadvisor.**datetime_now**()
Helper function for testing; return `datetime.datetime.now()`.

Returns `datetime.datetime.now()`

Return type datetime.datetime

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.**_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.**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.**paginate_dict**(*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()*.

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

awslimitchecker.version module

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

__dict__ = dict_proxy({'__module__': 'awslimitchecker.version', '__str__': <function __str__>, '__repr__': <function

__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

Changelog

0.10.0 (2017-06-25)

This release **removes the ElastiCache Clusters limit**, which no longer exists.

- Issue #283 - Add gitter.im chat link to README and docs.
- Issue #282 - versionfinder caused awslimitchecker to die unexpectedly on systems without a `git` binary on the PATH. Bump versionfinder requirement to `>= 0.1.1`.
- Issue #284 - Fix ElastiCache limits to reflect what AWS Support and the current documentation say, instead of a support ticket from July 2015.
  - Remove the "Clusters" limit, which no longer exists.
  - "Nodes per Cluster" limit is Memcached only.
  - Add "Subnets per subnet group" limit.
- Issue #279 - Add Github release to release process.

0.9.0 (2017-06-11)

- Issue #269 - set Trusted Advisor limit name overrides for some RDS limits that were recently added to TA, but with different names than what awslimitchecker uses.
- Fix bug Issue #270 - do not count propagated routes towards the VPC "Entries per route table" limit, per clarification in VPC service limits documentation ("This is the limit for the number of non-propagated entries per route table.")

- PR #276 / Issue #275 - Add new `--skip-service` CLI option and `AwsLimitChecker.remove_services` to allow skipping of one or more specific services during runs. (Thanks to tamsky for this contribution.)
- PR #274 / Issue #273 - Add support for new `i3` EC2 Instance types. (Thanks to tamsky) for this contribution.)
- Fix broken docs build due to changes Intersphinx reference to ValueError in python2 docs
- Add hack to `docs/source/conf.py` as workaround for https://github.com/sphinx-doc/sphinx/issues/3860
- Issue #267 - Firehose is only available in `us-east-1`, `us-west-2` and `eu-west-1`. Omit the traceback from the log message for Firehose `EndpointConnectionError` and log at warning instead of error.

0.8.0 (2017-03-11)

This release includes a **breaking API change**. Please see the first bullet point below. Note that once 1.0.0 is released (which should be relatively soon), such API changes will only come with a major version increment.

This release **requires new IAM permissions**: `redshift:DescribeClusterSnapshots` and `redshift:DescribeClusterSubnetGroups`.

This release **removes Python 3.2 support**. This was deprecated in 0.7.0. As of this release, awslimitchecker may still work on Python 3.2, but it is no longer tested and any support tickets or bug reports specific to 3.2 will be closed.

- PR #250 - Allow the `--service` command line option to accept multiple values. This is a **breaking public API change**; the `awslimitchecker.checker.AwsLimitChecker` *awslimitchecker.checker.AwsLimitChecker.check_thresholds*, *awslimitchecker.checker.AwsLimitChecker.find_usage*, and *awslimitchecker.checker.AwsLimitChecker.get_limits* methods now take an optional `service` *list* keyword argument instead of a *string* for a single service name.
- PR #251 - Handle GovCloud-specific edge cases; specifically, UnsupportedOperation errors for EC2 Spot Instance-related API calls, and limits returned as 0 by the DescribeAccountAttributes EC2 API action.
- PR #249 - Add support for RedShift limits (Redshift subnet groups and Redshift manual snapshots). This requires the `redshift:DescribeClusterSnapshots` and `redshift:DescribeClusterSubnetGroups` IAM permissions.
- Issue #259 - remove duplicates from required IAM policy returned by `awslimitchecker.checker.AwsLimitChecker.get_required_iam_policy` and `awslimitchecker --iam-policy`.
- Various TravisCI/tox build fixes:
  - Fix pip caching; use default pip cache directory
  - Add python 3.6 tox env and Travis env, now that it's released
  - Switch integration3 tox env from py3.4 to py3.6
- PR #256 - Add example of wrapping awslimitchecker in a script to send metrics to Prometheus.
- Issue #236 - Drop support for Python 3.2; stop testing under py32.
- Issue #257 - Handle ElastiCache DescribeCacheCluster responses that are missing `CacheNodes` key in a cluster description.
- Issue #200 - Remove EC2 Spot Instances/Fleets limits from experimental status.
- Issue #123 - Update documentation on using session tokens (Session or Federation temporary creds).

0.7.0 (2017-01-15)

This release deprecates support for Python 3.2. It will be removed in the next release.

This release introduces support for automatically refreshing Trusted Advisor checks on accounts that support this. If you use this new feature, awslimitchecker will require a new permission, `trustedadvisor:RefreshCheck`. See *Getting Started - Trusted Advisor* for further information.

- #231 - add support for new f1, r4 and t2.(xlarge|2xlarge) instance types, introduced in November 2016.
- #230 - replace the built-in `versioncheck.py` with versionfinder. Remove all of the many versioncheck tests.
- #233 - refactor tests to replace yield-based tests with parametrize, as yield-based tests are deprecated and will be removed in pytest 4.
- #235 - Deprecate Python 3.2 support. There don't appear to have been any downloads on py32 in the last 6 months, and the effort to support it is too high.
- A bunch of Sphinx work to use README.rst in the generated documentation.
- Changed DEBUG-level logging format to include timestamp.
- #239 - Support refreshing Trusted Advisor check results during the run, and optionally waiting for refresh to finish. See *Getting Started - Trusted Advisor* for further information.
- #241 / PR #242 - Fix default ElastiCache/Nodes limit from 50 to 100, as that's now what the docs say.
- #220 / PR #243 / PR #245 - Fix for ExpiredTokenException Errors. **awslimitchecker.connectable.credentials has been removed.** In previous releases, awslimitchecker had been using a `Connectable.credentials` class attribute to store AWS API credentials and share them between `Connectable` subclass instances. The side-effect of this was that AWS credentials were set at the start of the Python process and never changed. For users taking advantage of the Python API and either using short-lived STS credentials or using long-running or threaded implementations, the same credentials persisted for the life of the process, and would often result in ExpiredTokenExceptions. The fix was to move _boto_conn_kwargs and _get_sts_token from connectable to the top-level AwsLimitChecker class itself, get the value of the `_boto_conn_kwargs` property in the constructor, and pass that value in to all `Connectable` subclasses. This means that each instance of AwsLimitChecker has its own unique connection-related kwargs and credentials, and constructing a new instance will work intuitively - either use the newly-specified credentials, or regenerate STS credentials if configured to use them. I have to extend my deepest gratitude to the folks who identified and fixed this issue, specifically cstewart87 for the initial bug report and description, aebie for the tireless and relentlessly thorough investigation and brainstorming and for coordinating work for a fix, and willusher for the final implementation and dealing (wonderfully) with the dizzying complexity of many of the unit tests (and even matching the existing style).

0.6.0 (2016-11-12)

This release has a breaking change. The `VPC NAT gateways` has been renamed to `NAT Gateways per AZ` and its `get_current_usage()` method will now return a list with multiple items. See the changelog entry for #214 below.

This release requires the following new IAM permissions to function:

- `firehose:ListDeliveryStreams`
- #217 - add support for new/missing EC2 instance types: `m4.16xlarge`, `x1.16xlarge`, `x1.32xlarge`, `p2.xlarge`, `p2.8xlarge`, `p2.16xlarge`.
- #215 - support "Regional Benefit" Reserved Instances that have no specific AZ set on them. Per AWS, these are exempt from On-Demand Running Instances limits like all other RIs.

- #214 - The VPC "NAT gateways" limit incorrectly calculated usage for the entire region, while the limit is actually per-AZ. It also had strange capitalization that confused users. The name has been changed to "NAT Gateways per AZ" and the usage is now correctly calculated per-AZ instead of region-wide.
- #221 / PR #222 - Fix bug in handling of STS Credentials where they are cached permanently in `connectable.Connectable.credentials`, and new AwsLimitChecker instances in the same Python process reuse the first set of STS credentials. This is fixed by storing the Account ID as part of `connectable.ConnectableCredentials` and getting new STS creds if the cached account ID does not match the current `account_id` on the `Connectable` object.
- PR #216 - add new "Firehose" service with support for "Delivery streams per region" limit.
- #213 / PR #188 - support AWS cross-sdk credential file profiles via `-P` / `--profile`, like awscli.

0.5.1 (2016-09-25)

This release requires the following new IAM permissions to function:

- `ec2:DescribeSpot*` or more specifically:
  - `ec2:DescribeSpotDatafeedSubscription`
  - `ec2:DescribeSpotFleetInstances`
  - `ec2:DescribeSpotFleetRequestHistory`
  - `ec2:DescribeSpotFleetRequests`
  - `ec2:DescribeSpotInstanceRequests`
  - `ec2:DescribeSpotPriceHistory`
- `ec2:DescribeNatGateways`
- #51 / PR #201 - Add experimental support for Spot Instance and Spot Fleet limits (only the ones explicitly documented by AWS). This is currently experimental, as the documentation is not terribly clear or detailed, and the author doesn't have access to any accounts that make use of spot instances. This will be kept experimental until multiple users validate it. For more information, see *the EC2 limit documentation*.
- PR #204 contributed by hltbra to add support for VPC NAT Gateways limit.
- Add README and Docs link to waffle.io board.
- Fix bug where `--skip-ta` command line flag was ignored in *`show_usage()`* (when running with `-u` / `--show-usage` action).
- Add link to waffle.io Kanban board
- #202 - Adds management of integration test IAM policy via Terraform.
- #211 - Add working download stats to README and docs
- Fix broken landscape.io badges in README and docs
- #194 - On Limits page of docs, clarify that Running On-Demand Instances does not include Reserved Instances.
- Multiple `tox.ini` changes:
  - simplify integration and unit/versioncheck testenv blocks using factors and reuse
  - py26 testenv was completely unused, and py26-unit was running and working with mock==2.0.0
  - use pytest<3.0.0 in py32 envs
- #208 - fix KeyError when `timestamp` key is missing from TrustedAdvisor check result dict

0.5.0 (2016-07-06)

This release includes a change to `awslimitchecker`'s Python API. *`awslimitchecker.limit.AwsLimit.get_limit`* can now return either an `int` or `None`, as TrustedAdvisor now lists some service limits as being explicitly "unlimited".

- #195 - Handle TrustedAdvisor explicitly reporting some limits as "unlimited". This introduces the concept of unlimited limits, where the effective limit is `None`.

0.4.4 (2016-06-27)

- PR #190 / #189 - Add support for EBS st1 and sc1 volume types (adds "EBS/Throughput Optimized (HDD) volume storage (GiB)" and "EBS/Cold (HDD) volume storage (GiB)" limits).

0.4.3 (2016-05-08)

- PR #184 Fix default VPC/Security groups per VPC limit from 100 to 500, per VPC limits documentation (this limit was increased at some point recently). Thanks to Travis Thieman for this contribution.

0.4.2 (2016-04-27)

This release requires the following new IAM permissions to function:

- `elasticbeanstalk:DescribeApplications`
- `elasticbeanstalk:DescribeApplicationVersions`
- `elasticbeanstalk:DescribeEnvironments`
- #70 Add support for ElasicBeanstalk service.
- #177 Integration tests weren't being properly skipped for PRs.
- #175 the simplest and most clear contributor license agreement I could come up with.
- #172 add an integration test running against sa-east-1, which has fewer services than the popular US regions.

0.4.1 (2016-03-15)

- #170 Critical bug fix in implementation of #71 - SES only supports three regions (us-east-1, us-west-2, eu-west-1) and causes an unhandled connection error if used in another region.

0.4.0 (2016-03-14)

This release requires the following new IAM permissions to function:

- `rds:DescribeAccountAttributes`
- `iam:GetAccountSummary`
- `s3:ListAllMyBuckets`
- `ses:GetSendQuota`
- `cloudformation:DescribeAccountLimits`
- `cloudformation:DescribeStacks`

Issues addressed:

- #150 add CHANGES.rst to Sphinx docs
- #85 and #154
  - add support for RDS 'DB Clusters' and 'DB Cluster Parameter Groups' limits
  - use API to retrieve RDS limits
  - switch RDS from calculating usage to using the DescribeAccountAttributes usage information, for all limits other than those which are per-resource and need resource IDs (Max auths per security group, Read replicas per master, Subnets per Subnet Group)
  - awslimitchecker now **requires an additional IAM permission**, `rds:DescribeAccountAttributes`
- #157 fix for TrustedAdvisor polling multiple times - have TA set an instance variable flag when it updates services after a poll, and skip further polls and updates if the flag is set. Also add an integration test to confirm this.
- #50 Add support for IAM service with a subset of its limits (Groups, Instance Profiles, Policies, Policy Versions In Use, Roles, Server Certificates, Users), using both limits and usage information from the GetAccountSummary API action. This **requires an additional IAM permission**, `iam:GetAccountSummary`.
- #48 Add support for S3 Buckets limit. This **requires an additional IAM permission**, `s3:ListAllMyBuckets`.
- #71 Add support for SES service (daily sending limit). This **requires an additional IAM permission**, `ses:GetSendQuota`.
- #69 Add support for CloudFormation service Stacks limit. This **requires additional IAM permissions**, `cloudformation:DescribeAccountLimits` and `cloudformation:DescribeStacks`.
- #166 Speed up TravisCI tests by dropping testing for PyPy and PyPy3, and only running the -versioncheck tests for two python interpreters instead of 8.

0.3.2 (2016-03-11)

- #155 Bug fix for uncaught KeyError on accounts with Trusted Advisor (business-level support and above). This was caused by an undocumented change released by AWS between Thu, 10 Mar 2016 07:00:00 GMT and Fri, 11 Mar 2016 07:00:00 GMT, where five new IAM-related checks were introduced that lack the `region` data field (which the TrustedAdvisorResourceDetail API docs still list as a required field).

0.3.1 (2016-03-04)

- #117 fix Python 3.5 TravisCI tests and re-enable automatic testing for 3.5.
- #116 add t2.nano EC2 instance type; fix typo - "m4.8xlarge" should have been "m4.10xlarge"; update default limits for m4.(4|10)xlarge
- #134 Minor update to project description in docs and setup.py; use only _VERSION (not git) when building in RTD; include short description in docs HTML title; set meta description on docs index.rst.
- #128 Update Development and Getting Help documentation; add GitHub CONTRIBUTING.md file with link back to docs, as well as Issue and PR templates.
- #131 Refactor TrustedAdvisor interaction with limits for special naming cases (limits where the TrustedAdvisor service or limit name doesn't match that of the awslimitchecker limit); enable newly-available TrustedAdvisor data for some EC2 on-demand instance usage.

0.3.0 (2016-02-18)

- Add coverage for one code branch introduced in PR #100 that wasn't covered by tests.
- #112 fix a bug in the versioncheck integration tests, and a bug uncovered in versioncheck itself, both dealing with checkouts that are on a un-cloned branch.
- #105 build and upload wheels in addition to sdist
- #95 **major** refactor to convert AWS client library from boto to boto3. This also includes significant changes to the internal connection logic and some of the internal (private) API. Pagination has been moved to boto3 wherever possible, and handling of API request throttling has been removed from awslimitchecker, as boto3 handles this itself. This also introduces full, official support for python3.
- Add separate `localdocs` tox env for generating documentation and updating output examples.
- #113 update, expand and clarify documentation around threshold overrides; ignore some sites from docs linkcheck.
- #114 expanded automatic integration tests
- **Please note** that version 0.3.0 of awslimitchecker moved from using `boto` as its AWS API client to using `boto3`. This change is mostly transparent, but there is a minor change in how AWS credentials are handled. In `boto`, if the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables were set, and the region was not set explicitly via awslimitchecker, the AWS region would either be taken from the `AWS_DEFAULT_REGION` environment variable or would default to us-east-1, regardless of whether a configuration file (`~/.aws/credentials` or `~/.aws/config`) was present. With boto3, it appears that the default region from the configuration file will be used if present, regardless of whether the credentials come from that file or from environment variables.

0.2.3 (2015-12-16)

- PR #100 support MFA tokens when using STS assume role
- #107 add support to explicitly disable pagination, and use for TrustedAdvisor to prevent pagination warnings

0.2.2 (2015-12-02)

- #83 remove the "v" prefix from version tags so ReadTheDocs will build them automatically.
- #21 run simple integration tests of `-l` and `-u` for commits to main repo branches.

0.2.1 (2015-12-01)

- #101 Ignore stopped and terminated instances from EC2 Running On-Demand Instances usage count.
- #47 In VersionCheck git -e tests, explicitly fetch git tags at beginning of test.

0.2.0 (2015-11-29)

- #86 wrap all AWS API queries in `awslimitchecker.utils.boto_query_wrapper` to retry queries with an exponential backoff when API request throttling/rate limiting is encountered
- Attempt at fixing #47 where versioncheck acceptance tests fail under TravisCI, when testing master after a tagged release (when there's a tag for the current commit)

- Fix #73 versioncheck.py reports incorrect information when package is installed in a virtualenv inside a git repository
- Fix #87 run coverage in all unit test Tox environments, not a dedicated env
- Fix #75 re-enable py26 Travis builds now that pytest-dev/pytest#1035 is fixed (pytest >= 2.8.3)
- Fix #13 re-enable Sphinx documentation linkcheck
- Fix #40 add support for pagination of API responses (to get all results) and handle pagination for all current services
- Fix #88 add support for API-derived limits. This is a change to the public API for `awslimitchecker.limit.AwsLimit` and the CLI output.
- Fix #72 add support for some new limits returned by Trusted Advisor. This renames the following limits: * `EC2/EC2-VPC Elastic IPs` to `EC2/VPC Elastic IP addresses (EIPs)` * `RDS/Read Replicas per Master` to `RDS/Read replicas per master` * `RDS/Parameter Groups` to `RDS/DB parameter groups`
- Fix #84 pull some EC2 limits from the API's DescribeAccountAttributes action
- Fix #94 pull AutoScaling limits from the API's DescribeAccountLimits action
- Add `autoscaling:DescribeAccountLimits` and `ec2:DescribeAccountAttributes` to required IAM permissions.
- Ignore `AccountLimits` objects from result pagination

0.1.3 (2015-10-04)

- Update trove classifier Development Status in setup.py to Beta
- Fix markup formatting issue in `docs/source/getting_started.rst`
- temporarily disable py26 testenv in Travis; failing due to upstream bug https://github.com/pytest-dev/pytest/issues/1035
- PR #64 and #68 - support [STS](http://docs.aws.amazon.com/STS/latest/APIReference/Welcome.html) and regions * Add support for passing in a region to connect to via `-r` / `--region` * Add support for using STS to check resources in another account, including support for `external_id` * Major refactor of how service classes connect to AWS API
- #74 add support for EC2 t2.large instance type
- #65 handle case where ElastiCache API returns CacheCluster response with CacheNodes None
- #63 update Python usage documentation
- #49 clean up badges in README.rst and sphinx index.rst; PyPi downloads and version badges broken (switch to shields.io)
- #67 fix typo in required IAM policy; comma missing in list returned from *_Ec2Service.required_iam_permissions()*
- #76 default limits for EBS volume usage were in TiB not GiB, causing invalid default limits on accounts without Trusted Advisor
- Changes to some tests in `test_versioncheck.py` to aid in debugging #47 where Travis tests fail on master because of git tag from release (if re-run after release)

0.1.2 (2015-08-13)

- #62 - For 'RDS/DB snapshots per user' limit, only count manual snapshots. (fix bug in fix for #54)

0.1.1 (2015-08-13)

- #54 - For 'RDS/DB snapshots per user' limit, only count manual snapshots.
- PR #58 - Fix issue where BotoServerError exception is unhandled when checking ElastiCache limits on new accounts without EC2-Classic.
- #55 - use .version instead of .parsed_version to fix version information when using pip<6
- #46 - versioncheck integration test fixes * Rename `-integration` tox environments to `-versioncheck` * Skip versioncheck git install integration tests on PRs, since they'll fail
- #56 - logging fixes * change the AGPL warning message to write directly to STDERR instead of logging * document logging configuration for library use * move boto log suppression from checker to runner
- Add contributing docs

0.1.0 (2015-07-25)

- initial released version

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

Symbols

A

C

D

F

G

I

L

P

R

S

T

U

V