cloudyr / aws.s3

Amazon Simple Storage Service (S3) API Client https://cloud.r-project.org/package=a...

#cloudyr #aws #r #aws-s3 #s3 #s3-storage #r-package #amazon

298 commits | 3 branches | 3 releases | 22 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

jon-mago and leeper Update NEWS.md — Latest commit ff29667 13 days ago

.github	add .github files	6 months ago
R	Make sure content-length is an integer	13 days ago
inst	failed example of post method	3 years ago
man-roxygen	expose ACL functionality (closes #137)	a year ago
man	add 'acl' and 'headers' args to put_acl() (#137)	a month ago
tests	Fix simple typo in tests (closes #232)	4 months ago
.Rbuildignore	tweak .travis.yml	5 months ago
.gitignore	ignore revdep/ directory	4 months ago
.travis.yml	tweak .travis.yml	5 months ago
DESCRIPTION	add 'acl' and 'headers' args to put_acl() (#137)	a month ago
Makefile	add Makefile to simplify build	a year ago
NAMESPACE	add s3connection() function (closes #217)	2 months ago
NEWS.md	Update NEWS.md	13 days ago
README.Rmd	add s3connection() function (closes #217)	2 months ago
README.md	add s3connection() function (closes #217)	2 months ago
drat.sh	modify drat deploy message	3 years ago

README.md

AWS S3 Client Package

aws.s3 is a simple client package for the Amazon Web Services (AWS) Simple Storage Service (S3) REST API. While other packages currently connect R to S3, they do so incompletely (mapping only some of the API endpoints to R) and most implementations rely on the AWS command-line tools, which users may not have installed on their system.

To use the package, you will need an AWS account and to enter your credentials into R. Your keypair can be generated on the IAM Management Console under the heading *Access Keys*. Note that you only have access to your secret key once. After it is generated, you need to save it in a secure location. New keypairs can be generated at any time if yours has been lost, stolen, or forgotten. The **aws.iam** package profiles tools for working with IAM, including creating roles, users, groups, and credentials programmatically; it is not needed to *use* IAM credentials.

A detailed description of how credentials can be specified is provided at: https://github.com/cloudyr/aws.signature/. The easiest way is to simply set environment variables on the command line prior to starting R or via an `Renviron.site` or `.Renviron` file, which are used to set environment variables in R during startup (see `? Startup`). They can be also set within R:

```
Sys.setenv("AWS_ACCESS_KEY_ID" = "mykey",
           "AWS_SECRET_ACCESS_KEY" = "mysecretkey",
           "AWS_DEFAULT_REGION" = "us-east-1",
           "AWS_SESSION_TOKEN" = "mytoken")
```

To use the package with S3-compatible storage provided by other cloud platforms, set the `AWS_S3_ENDPOINT` environment variable to the appropriate host name. By default, the package uses the AWS endpoint: `s3.amazonaws.com`

Code Examples

The package can be used to examine publicly accessible S3 buckets and publicly accessible S3 objects without registering an AWS account. If credentials have been generated in the AWS console and made available in R, you can find your available buckets using:

```
library("aws.s3")
bucketlist()
```

If your credentials are incorrect, this function will return an error. Otherwise, it will return a list of information about the buckets you have access to.

Buckets

To get a listing of all objects in a public bucket, simply call

```
get_bucket(bucket = '1000genomes')
```

Amazon maintains a listing of Public Data Sets on S3.

To get a listing for all objects in a private bucket, pass your AWS key and secret in as parameters. (As described above, all functions in **aws.s3** will look for your keys as environment variables by default, greatly simplifying the process of making an s3 request.)

```
# specify keys in-line
get_bucket(
  bucket = 'my_bucket',
  key = YOUR_AWS_ACCESS_KEY,
  secret = YOUR_AWS_SECRET_ACCESS_KEY
)

# specify keys as environment variables
Sys.setenv("AWS_ACCESS_KEY_ID" = "mykey",
           "AWS_SECRET_ACCESS_KEY" = "mysecretkey")
get_bucket("my_bucket")
```

S3 can be a bit picky about region specifications. `bucketlist()` will return buckets from all regions, but all other functions require specifying a region. A default of `"us-east-1"` is relied upon if none is specified explicitly and the correct region can't be detected automatically. (Note: using an incorrect region is one of the most common - and hardest to figure out - errors when working with S3.)

Objects

This package contains many functions. The following are those that will be useful for working with objects in S3:

1. `bucketlist()` provides the data frames of buckets to which the user has access.
2. `get_bucket()` and `get_bucket_df()` provide a list and data frame, respectively, of objects in a given bucket.
3. `object_exists()` provides a logical for whether an object exists. `bucket_exists()` provides the same for buckets.
4. `s3read_using()` provides a generic interface for reading from S3 objects using a user-defined function. `s3write_using()` provides a generic interface for writing to S3 objects using a user-defined function
5. `get_object()` returns a raw vector representation of an S3 object. This might then be parsed in a number of ways, such as `rawToChar()`, `xml2::read_xml()`, `jsonlite::fromJSON()`, and so forth depending on the file format of the object. `save_object()` saves an S3 object to a specified local file without reading it into memory.
6. `s3connection()` provides a binary readable connection to stream an S3 object into R. This can be useful for reading for very large files. `get_object()` also allows reading of byte ranges of functions (see the documentation for examples).
7. `put_object()` stores a local file into an S3 bucket. The `multipart = TRUE` argument can be used to upload large files in pieces.
8. `s3save()` saves one or more in-memory R objects to an .Rdata file in S3 (analogously to `save()`). `s3saveRDS()` is an analogue for `saveRDS()`. `s3load()` loads one or more objects into memory from an .Rdata file stored in S3 (analogously to `load()`). `s3readRDS()` is an analogue for `readRDS()`
9. `s3source()` sources an R script directly from S3

They behave as you would probably expect:

```
# save an in-memory R object into S3
s3save(mtcars, bucket = "my_bucket", object = "mtcars.Rdata")

# `load()` R objects from the file
s3load("mtcars.Rdata", bucket = "my_bucket")

# get file as raw vector
get_object("mtcars.Rdata", bucket = "my_bucket")
# alternative 'S3 URI' syntax:
get_object("s3://my_bucket/mtcars.Rdata")
```

```
# save file locally
save_object("mtcars.Rdata", file = "mtcars.Rdata", bucket = "my_bucket")

# put local file into S3
put_object(file = "mtcars.Rdata", object = "mtcars2.Rdata", bucket = "my_bucket")
```

Installation

This package is not yet on CRAN. To install the latest development version you can install from the cloudyr drat repository:

```
# latest stable version
install.packages("aws.s3", repos = c("cloudyr" = "http://cloudyr.github.io/drat"))

# on windows you may need:
install.packages("aws.s3", repos = c("cloudyr" = "http://cloudyr.github.io/drat"), INSTALL_opts = "--no-multiarch")
```

Or, to pull a potentially unstable version directly from GitHub:

```
if (!require("remotes")) {
    install.packages("remotes")
}
remotes::install_github("cloudyr/aws.s3")
```