An introduction to the RAmazonS3 package

Duncan Temple Lang
Roger Peng

Table of Contents

Getting Started

Note

We note that we talk about S3 in this document. There is some potential for confusion. S3 here refers to the Amazon storage server. In R, S3 typically refers to the "old"-style class mechanism. We do not talk about that in this document; so S3 refers to Amazon.

S3 is an Amazon service for hosting files and allowing them to be accessed from anywhere. This is a globally available file system rather than being tied to a particular machine. It avoids having to run a Web server and also provides a reasonably rich way to provide different levels of access to files.

You can access files created by others on the S3 service and you can create your own files. You can work with these files yourself and also grant access to others to read, write, create, etc. We'll start with the simple case where you can read buckets and files/objects that other people have created and to which you have access. Roger has a bucket named RRupload. We can fetch the list of objects it contains using *listBucket*() , e.g.,

```
R
listBucket("RRupload", auth = NA)
```

We specify the name and explicitly force that no authorization information needs to be sent. [1] The result is a data frame

```
                                                                    R
Key        LastModified                          ETag Size
1 Todo.xml.gz 2009-07-31 18:17:41 55a67aed325ff758a0896473f4c91554 1703 65a011a29c
2         bar 2009-07-31 17:16:01 bb184e3e0ca66a62c07e8f1871dd1039   16 65a011a29c
3    bucket.R 2009-07-30 13:32:35 126b7cdb5ff3c3163735025705115 99d  340 65a011a29c
4  compressed 2009-07-31 17:35:27 f87e83ae612fbf5593ea6a44a4cb08f8   80 65a011a29c
5         foo 2009-07-31 17:14:31 bb184e3e0ca66a62c07e8f1871dd1039   16 65a011a29c
6         tmp 2009-08-02 23:09:11 41fb5b5ae4d57c5ee528adb00e5e8e74   16 65a011a29c
```

[1] We should be able to do this a signature, but this is misbehaving at present. It looks to be something with upper-case bucket names.

```
7         xxx 2009-07-31 18:32:57 b5a7791824a3719256e4884e9c65c7f3   23 65a011a29c
```

This gives us the names of each object and its size and when it was last modified. The ETag and ID fields are used to uniquely identify the object and the developer who created/modified the object.

Getting the Contents of an Object

We can retrieve the contents of objects in a bucket that we have read-access to with *getFile*() . We give this both the name of the object and the bucket. This can be done as two separate arguments (bucket and name) or as a single argument of the form *bucket/object*. So we can get the object bucket.R in RRupload with either

R
```
x = getFile("RRupload", "bucket.R", auth = NA)
y = getFile("RRupload/bucket.R", auth = NA)
```

Depending on how the object was created, there may or may not be information about its content type. If there is, *getFile*() attempts to handle it correctly, e.g. recognizing text content and converting it to a character string. However, if there is not content type information, we return the content as a raw vector. If you know this is text, you can convert it with

R
```
rawToChar(x)
```

This is about all you can do with nothing but read-permissions. So we'll move on to functions that require permissions.

More control

Once you have a login and secret, be they your own or somebody else's, you can do a lot more with Amazon S3. Firstly, you can find all the buckets owned by that login with *listBuckets*() . You specify the login and secret key as a named character vector as the value of the *auth* parameter, e.g.,

R
```
listBuckets(auth = c('login' = 'secret'))
```

Many users will have a single login-secret pair and it is convenient to put these in your R global options. You can set these as

R
```
options(AmazonS3 = c('login' = "secret"))
```

The functions in RAmazonS3 will look for this and use it if *auth* is not specified in a call. So we can set the option and then call *listBuckets*() simply with

R
```
listBuckets()

                bucket        creationDate
1             RRupload 2009-06-01 19:39:36
2                 cpkg 2009-06-01 18:54:56
3       duncantl-test1 2009-08-06 21:11:02
4             rdpshare 2009-06-04 12:19:19
5 reproducibleresearch 2009-06-01 18:40:23
```

```
6        test3duncantl 2009-08-06 15:28:27
7        test4duncantl 2009-08-06 15:28:45
8         testDuncanTL 2009-08-05 23:51:27
9          www.penguin 2009-06-01 21:20:38
```

Now that we know what buckets we have, we can list any one of these with *listBucket*() (and using the implicit specification of the *auth* argument), e.g.,

```
                                                                        R
listBucket("rdpshare")

           Key        LastModified                                ETag
1 greeting.xml 2009-07-30 00:25:10 8822584dd80ffc3c609ed799334d5766
  Size                                                          Owner.ID
1  101 a02e4359c85dad7828cc8a88c8ddd021ee5deb57cb3008ed19444ffa8f9b9a14
  Owner.DisplayName StorageClass
1            rdpeng     STANDARD
```

We can get the file with

```
                                                                        R
rawToChar(getFile("rdpshare/greeting.xml"))
```

Creating and Removing Buckets

The function *makeBucket*() can be used to create a new bucket. For example, we can create a bucket named "duncantl-test" with the command

```
                                                                        R
makeBucket("duncantl-test")
```
[2]

We can remove a bucket with *removeBucket*() , giving it the name of the bucket, e.g.,

```
                                                                        R
removeBucket("duncantl-test")
```

Creating Objects/Content

It is not very useful to be only able to create buckets. We want to be able to store content. We can do this by uploading files or by taking content directly in R and uploading it from memory. We do this with the function *addFile*() . This expects the contents or file name to upload and then the location on S3 to where it will be uploaded. We can give the bucket-name pair in a single string as before in the form "bucket/name" or as two separate arguments - bucket, name. These two forms are show here

```
                                                                        R
```

[2]At present, this sometimes "hangs" waiting for additional input. Ctrl-D will terminate it and the bucket will be created. This is something to do with the HTTP header, but we have killed off the Expect: 100-continue and sent a Content-length of 0.

```
content = I("This is a string")
addFile(content, "duncantl-test/foo") # note the missing 2nd argument.
addFile(content, "duncantl-test", "bar")
```

The content can be any R object. If it is a string, we assume that this is the name of a file and we read that file and upload it. If we want to specify actual text to be uploaded as-is, we can "escape" it using the AsIs function *I*() as we have shown above. When *addFile*() sees that contents inherits from *AsIs*, it does not consider the string to be a file name. We can also use the *isContents* parameter to specify this explicitly.

Once we have uploaded the content/file, we will see it in the listing:

R
```
listBucket("duncantl-test")

  Key        LastModified                             ETag Size
1 foo 2009-08-06 23:11:27 41fb5b5ae4d57c5ee528adb00e5e8e74   16
                                                          Owner.ID
1 a02e4359c85dad7828cc8a88c8ddd021ee5deb57cb3008ed19444ffa8f9b9a14
  Owner.DisplayName StorageClass
1             rdpeng     STANDARD
```

When we upload content, we should specify its content type. We have seen that if we don't, accessing it requires more intervention by the recipient. We can specify the content type via the *type* parameter. This should be something reasonably standard such as "application/gzip", "application/binary", "text/html" or "text/xml". We may provide functionality that guesses the content-type from the extension of the file or type of the object. For now, if we have a character string, we set the content-type to text. Otherwise, we assume binary content.

We can also specify additional meta-information. These are, in some sense, similar attributes on an R object in that they are name-vale pairs. The values will be converted to strings. You specify these when uploading an object via the *meta* argument. The command

R
```
addFile(I("Some text"), "dtl-ttt", "bob",
         meta = c(foo = 123, author = "Duncan Temple Lang"))
```

provides two meta values named "foo" and "author". We can retrieve this meta-information for any of the S3 objects.

As easy as it is to create content, we can remove an object with *removeFile*() , e.g.

R
```
removeFile("duncantl-test/foo")
removeFile("duncantl-test", "foo")
```

Another somewhat convenient operation is to copy a file/object. We can copy an object in a bucket to an other object in the same bucket or to an entirely separate bucket. We use *copyFile*() for this. We give it the name of the existing source object and the target object as the two arguments. These can (and should be) in the form "bucket/name". The target can be just a name and we assume the target bucket is the same as the source.

R

```
copyFile("duncantl-test/bar", "xxx")
```
Alternatively, we can copy an object from one bucket to another, e.g.

R
```
copyFile("www.penguin/temp1", "dtl-share/temp1")
```
We can also copy an object to another bucket and re-use the object name within the new bucket by adding a "/" to the end of the target bucket. For example,

R
```
copyFile("dtl-share/temp1", "dtl-ttt/")
```
will create a copy of temp1 in dtl-ttt.

Other Facilities

We have described the commonly used facilities above. There are a few others. We can determine if an object exists using *s3Exists*() . For example,

R
```
s3Exists("dtl-ttt/bob")
s3Exists("dtl-ttt/jane")
```
determine if the two objects named bob and jane are present in the bucket "dtl-ttt".

We can get (meta-)information about an object with *about*() (also named *getInfo*()). This returns a character vector giving any meta-data associated with the object, e.g. that was specified when the object/file was created. For example,

R
```
about("dtl-ttt/bob")
about("dtl-ttt", "bob")

                          author
                "Duncan Temple Lang"
                               foo
                             "123"
                     Last-Modified
       "Sat, 08 Aug 2009 00:20:11 GMT"
                              ETag
"\"9db5682a4d778ca2cb79580bdb67083f\""
                      Content-Type
                      "text/plain"
                    Content-Length
                               "9"
                            Server
                       "AmazonS3"
```

We can query and set the access controls for a bucket or file/object. The simple way to do this is with *getS3Access*() and *setS3Access*() . These take the name of the object being queried as bucket-name pair. *getS3Access*() tells us who has what permissions. It returns a data frame giving this information for the specified bucket or bucket-object. *setS3Access*() allows us to set a permission in a simple way. We can make a bucket or object private, public for reading, public for reading and writing, or authenticated

read access. These apply to everybody. *setACL*() allows us to specify fine-grained access "limitations" for individuals.

Let's create a new bucket named "dtl-share" and a file to it:

R
```
makeBucket("dtl-share")
addFile(I("Hi everyone"), "dtl-share/hello")
```
Now we look at the access settings:

R
```
getS3Access("dtl-share")

                                                                ID DisplayName   p
1 a02e4359c85dad7828cc8a88c8ddd021ee5deb57cb3008ed19444ffa8f9b9a14      rdpeng FUL
```
(We are using Roger's login and that is why it is owned by him eventhough we are using dtl s a bucket name!) We get the same result for the "hello" file.

So now let's make that file publicly available.

R
```
setS3Access("dtl-share", "hello", "public-read")
```
Now we can examine the access settings

R
```
getS3Access("dtl-share/hello")

                                                                ID DisplayName
1 a02e4359c85dad7828cc8a88c8ddd021ee5deb57cb3008ed19444ffa8f9b9a14      rdpeng
2                                                           <NA>        <NA> htt
```
This shows the original access (for rdpeng) but has a second row and a new column in the data frame - **URI**. This second row indicates that all the users can read this file. You can access it via a Web browser at http://dtl-share.s3.amazonaws.com/hello.

But what if we want to allow DTL to have full control over the file also.

R
```
setACL("dtl-share/hello", "4e10a30dc41e8b3b6c7bcebe32720f27b4a79454e99155590730897
```

Bucket Objects

We have described the low-level functionality in R for directly accessing the S3 Amazon storage server's facilities. We now turn our attention to a different R interface that hides some of these functions. You still list all the buckets available for a particular "login" via *listBuckets*() . However, instead of `listBucket(name)`, we can think of the bucket as being an object in R. We create such an object with

R
```
dtl.share = Bucket("dtl-share")
```
(We could have assigned this to any variable in R.) This is an object of class *Bucket*. It has methods that working with it slightly more convenient, especially for interactive use. The constructor also allows

us to specify the authorization key and secret which is stored in the object. This allows us to avoid having to specify authorization information in subsequent calls. This is convenient if one is working with several different authorization keys, even within the same bucket. One can have a separate `Bucket` object for each authorization. Note that these bucket objects should not be serialized as the secret is private.

The first thing we can do with a `Bucket` object is get a list of the objects it contains using `names`() . This gives the names of the objects as a character vector.

We can fetch the contents of one of the objects in a bucket with the `[[`() or `$`() operator, e.g.

R
```
b$temp1
b[["temp1"]]
```
One of the benefits of the `[[`() syntax is that we can specify additional arguments. For example, we could specify whether the content was binary or not using

R
```
b[["temp1", binary = FALSE]]
```

We can use a `Bucket` object to upload content to an object within the bucket. We use `$<-`() or `[[<-`() , e.g.,

R
```
b$temp3 = I("A string in R")
doc = xmlParse(system.file("doc", "s3amazon.xml"))
b[["temp4", type = "text/html"]] = I(saveXML(doc))
```

Higher-level Functionality

The function `s3Save`() is an almost exact substitute for the `save`() function. It allows us to serialize one or more R objects into a file and to upload that file to S3. The `file` parameter in this function identifies the bucket and object in the S3 server, e.g. "dtl-share/ab.rda"

R
```
a = 1:10
b = letters[1:4]
s3Save(a, b, file = "dtl-share/ab.rda")
```
This saves the objects in a binary format and gives the resulting S3 amazon object a Content-Type of application/x-rda. We will implement a corresponding handler for de-serializing directly from the S3 server as we retrieve it, e.g.

R
```
s3Load("dtl-share/ab.rda")
```
or even

R
```
getFile("dtl-share/ab.rda")
```
and let `getURLContent`() to do the work for us.

Future Directions

Roger has some nice ideas about disseminating objects from statistical analyses using S3 as a repository.