Amazon Web Services Simple Storage Service (S3)

Two AWS S3 buckets by default: a private bucket and a public bucket.

General info:

<https://sciwiki.fredhutch.org/scicomputing/store_objectstore/>

AWS Command Line Interface CLI:

Connect to aws?

<https://aws.amazon.com/s3/>

complete set of AWS S3 commands:

<https://docs.aws.amazon.com/cli/latest/reference/s3/index.html>

scratch S3 bucket

Each PI account has an additional scratch S3 bucket, currently called fh-pi-lastname-f-nextflow-scratch. While these have “nextflow” in the name, they are a general-purpose scratch bucket that can be used for temporary data storage while you are running any workflow or for any other temporary purpose. In this bucket, you can create/use the folders “delete10/”, “delete30/”, or “delete60/” and these will automatically delete any data in them 10, 30, or 60 days (respectively) after that data is initially created

Public Access:

Only SciComp is able to make an object publicly available. Objects in the public bucket can be shared several ways:

* public access restricted via IP address range. This is not a guarantee since it is possible (though difficult) for an unauthorized individual to spoof an IP address to gain access to this data. For this reason, this access should still be considered “public” even though it is restricted by IP address.
* Unrestricted public access. This will enable anyone in the world to download a specific object or any object within a given prefix without credentials as long as they know the URL. It is not possible for them to list all the files available publicly, however. If you wish to share a large number of files in this manner, the recommended approach is to create a single text file that contains a list of hyperlinks to the URLs for all of the objects you wish to share and then include the URL of that text file in whatever means you use to advertise your dataset. There are other ways to accomplish this as well; contact SciComp for assistance.

NOTE: Public access is read-only. Write access will *NEVER* be granted to the general public.

copy the file hello.txt from your current directory to the top-level folder of an S3 bucket

aws s3 cp hello.txt s3://fh-pi-doe-j-eco/

Copy the file hello.txt to the folder path a/b/c and folder are not created in advance.

aws s3 cp s3://fh-pi-doe-j-eco/hello.txt s3://fh-pi-doe-j-eco/a/b/c/

Copying files from an S3 bucket to the machine you are logged into, the current directory on the (rhino or gizmo) , represented by dot (.) in here:

aws s3 cp s3://fh-pi-doe-j-eco/hello.txt .

Copy files to prefix:

You can also copy files directly into an S3 prefix (denoted by a “PRE” before the name on S3). The prefix does not have to already exist - this copying step can generate one. To copy a file into a prefix, use the local file path in your cp command as before, but make sure that the destination path for S3 is followed by a / character (the / is essential). Without the trailing /, the file hello.txt will be copied into the S3 bucket under the filename test\_prefix, rather than into the desired prefix itself.

aws s3 cp s3://fh-pi-doe-j-eco/hello.txt s3://fh-pi-doe-j-eco/test\_prefix/

list the contents of your lab’s bucket:

aws s3 ls s3://fh-pi-doe-j-eco/

To list the contents of a specific folder, just add the folder path to the end of the previous example:

aws s3 ls s3://fh-pi-doe-j-eco/a/b/c/

See the entire contents of every folder in your bucket:

aws s3 ls --recursive --summarize s3://fh-pi-doe-j-eco/

You can optionally add the --human-readable argument to that command to get the file sizes output in a more easily recognizable format. This is roughly equivalent to the command ls -alhR /path/on/posix/filesystem when working on a posix file system in a shell such as Bash.

Working with a bucket that belongs to another lab, you will need to ensure that you set the object ACL (access control list) correctly when uploading data into that other lab’s bucket:

To do this, append the argument --acl bucket-owner-full-control to the aws s3 cp or aws s3 sync commands. If you are using Motuz to copy data, Motuz will handle this for you.

aws s3 cp --acl bucket-owner-full-control s3://fh-pi-doe-j-eco/test.txt s3://fh-pi-heisenberg-w-eco/

Find aws commands:

aws s3 help

To see documentation for a specific s3 subcommand, such as cp, do this:

aws s3 help cp

AWS via Python

From any of the rhino systems you can see which Python builds are available by typing ml Python/3. and pressing the TAB key twice. Choose the most recent version (at the time of writing it is Python/3.6.5-foss-2016b-fh3). Once you have loaded a python module with ml, the Python libraries you will need ([boto3](https://boto3.readthedocs.io/en/latest/index.html), [pandas](https://pandas.pydata.org/pandas-docs/stable/), etc.) will be available

<https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>

You can then get to an interactive Python prompt with the python command, but many prefer to use ipython to work with Python interactively.

ml Python/3.6.5-foss-2016b-fh3

python

From within python (or ipython) do the following to get started:

import boto3

import numpy as np

import pandas as pd

import dask.dataframe as dd

from io import StringIO, BytesIO

s3 = boto3.client("s3")

s3\_resource = boto3.resource('s3')

bucket\_name = "fh-pi-doe-j-eco" # substitute your actual bucket name

The following fragments all assume that these lines above have been run.

### List all buckets in our account:

response = s3.list\_buckets()

If you just want to see the bucket names:

for bucket in response['Buckets']:

print(bucket['Name']

### List all objects in a bucket

response = s3.list\_objects\_v2(Bucket=bucket\_name)

To view just the object names (keys):

for item in response['Contents']:

print(item['Key'])

-Note that this method only returns the first 1000 items in the bucket. If there are more items to be shown, response['IsTruncated'] will be True. If this is the case, you can retrieve the full object listing as follows:

paginator = s3.get\_paginator('list\_objects\_v2')

page\_iterator = paginator.paginate(Bucket=bucket\_name)

for page in page\_iterator:

for item in page['Contents']:

print(item['Key'])

### Read object listing into Pandas data frame[Permalink](https://sciwiki.fredhutch.org/compdemos/aws-s3/#read-object-listing-into-pandas-data-frame)

response = s3.list\_objects\_v2(Bucket=bucket\_name)

df = pd.DataFrame.from\_dict(response['Contents'])

<https://docs.dask.org/en/stable/dataframe.html>

There are two implementations of data frames in python: [pandas](https://pandas.pydata.org/pandas-docs/stable/) and dask). Use pandas when the data you are working with is small and will fit in memory. If it’s too big to fit in memory, use dask (it’s easy to convert between the two, and dask uses the pandas API, so it’s easy to work with both kinds of data frame). We’ll show examples of reading and writing both kinds of data frames to and from S3.

**NOTE**: Pandas dataframes are usually written out (and read in) as CSV files. Dask dataframes are written out in parts, and the parts can only be read back in with dask.

# generate a pandas data frame of random numbers:

df = pd.DataFrame(np.random.randint(0,100,size=(100, 4)), columns=list('ABCD'))

# save it in s3:

csv\_buffer = StringIO()

df.to\_csv(csv\_buffer)

s3\_resource.Object(bucket\_name, 'df.csv').put(Body=csv\_buffer.getvalue())

# convert data frame to dask:

dask\_df = dd.from\_pandas(df, 3)

# save dask data frame to s3 in parts:

dask\_df.to\_csv("s3://{}/dask\_data\_parts".format(bucket\_name))

Reading objects from S3

To read the csv file from the previous example into a pandas data frame:

obj = s3.get\_object(Bucket=bucket\_name, Key="df.csv")

df2 = pd.read\_csv(BytesIO(obj['Body'].read()))

To read the parts written out in the previous example back into a dask data frame:

dask\_df2 = dd.read\_csv("s3://{}/dask\_data\_parts/\*".format(bucket\_name))

Upload a file to S3

# write the example data frame to a local file

df.to\_csv("df.csv")

# upload file:

s3.upload\_file("df.csv", Bucket=bucket\_name, "df.csv")

Download a file from S3

# second argument is the remote name/key, third argument is local name

s3.download\_file(bucket\_name, "df.csv", "df.csv")

# AWS via R

You can use [Amazon Web Services’ S3](https://aws.amazon.com/s3/) (Simple Storage Service) directly from R. The R package which facilitates this, aws.s3, is included in recent builds of R available on the rhino systems and the gizmo cluster.

## Getting Started[Permalink](https://sciwiki.fredhutch.org/compdemos/aws-s3/#getting-started-1)

The first step is to load a recent R module:

ml R/3.5.0-foss-2016b-fh1

Then start R:

R

Load the aws.s3 R package:

library(aws.s3)

**NOTE:** The example fragments from this point on assume you are in an Rsession with aws.s3 loaded.

### List all buckets[Permalink](https://sciwiki.fredhutch.org/compdemos/aws-s3/#list-all-buckets)

blist <- bucketlist()

### List all objects in a bucket[Permalink](https://sciwiki.fredhutch.org/compdemos/aws-s3/#list-all-objects-in-a-bucket-1)

The bucket name you supply must be one you have access to.

b <- 'fh-pi-doe-j-eco'

objects <- get\_bucket(b)

### Get bucket contents as a data frame[Permalink](https://sciwiki.fredhutch.org/compdemos/aws-s3/#get-bucket-contents-as-a-data-frame)

df <- get\_bucket\_df(b)

### Saving objects to S3[Permalink](https://sciwiki.fredhutch.org/compdemos/aws-s3/#saving-objects-to-s3-1)

Create a data frame of random numbers and save it to S3:

df <- data.frame(replicate(10,sample(0:1,1000,rep=TRUE)))

s3save(df, bucket=b, object="foo/bar/baz/df")

### Loading objects from S3[Permalink](https://sciwiki.fredhutch.org/compdemos/aws-s3/#loading-objects-from-s3)

# first remove the object from memory if it's there:

if ("df" %in% ls()) rm("df")

# now load it:

s3load(object="foo/bar/baz/df", bucket=b)

# demonstrate that it exists again:

head(df)

### Upload a file to S3[Permalink](https://sciwiki.fredhutch.org/compdemos/aws-s3/#upload-a-file-to-s3-1)

First, write the existing df data frame to a csv file on your local disk:

write.csv(df, file="df.csv")

# copy the csv to s3:

put\_object("df.csv", "foo/bar/baz/df.csv", b)

### Read a CSV in S3 into a data frame[Permalink](https://sciwiki.fredhutch.org/compdemos/aws-s3/#read-a-csv-in-s3-into-a-data-frame)

# first remove the object from memory if it's there:

if ("df" %in% ls()) rm("df")

df <- s3read\_using(read.csv, object="foo/bar/baz/df.csv", bucket=b)

# demonstrate that it exists again:

head(df)

### Download a file from S3[Permalink](https://sciwiki.fredhutch.org/compdemos/aws-s3/#download-a-file-from-s3-1)

This will create the file df.csv in the current directory:

save\_object("foo/bar/baz/df.csv", b)

### Work with object names matching a pattern[Permalink](https://sciwiki.fredhutch.org/compdemos/aws-s3/#work-with-object-names-matching-a-pattern)

Assume your S3 bucket has three objects whose keys start with foo/bar/baz/and end with one of d, e, or f. You want to read each object into memory and end up with d, e, and f objects in your R session.

bdf <- get\_bucket\_df(b)

matches <- bdf$Key[grep("^foo/bar/baz/", bdf$Key)]

for (match in matches) {

s3load(object=match, bucket=b)

}

### Write data frame to S3 as a file[Permalink](https://sciwiki.fredhutch.org/compdemos/aws-s3/#write-data-frame-to-s3-as-a-file)

When you have a data frame in R that you’d like to save as an object in S3, you’ll do the following:

# using write.table

s3write\_using(df,

FUN = write.table, quote = F, row.names = F, sep = "\t",

object = "foo/bar/baz/df.txt",

bucket = b)

# write write.csv

s3write\_using(df,

FUN = write.csv, quote = F, row.names = F,

object = "foo/bar/baz/df.csv",

bucket = b)