That time I has quite generous credits from AWS, so haven’t worried too much about costs. Unfortunately credits are about to expire, so I had to optimize the tech stack a bit to reduce ongoing costs of running that hobby project.

First and easy part was to downsize EC2 instance, I had rather powerful c type instance, which I replaced by t2.medium

Since the project isn’t extremely high load, it is good enough.

More complex part was to replace DB as a place to collect data.

I have decided to use DB for a reason – it is designed to store that kind of data more suitable for that than files. For example I used DB View to combine couple tables and relied on DB primary key to avoid data duplication (store the same tweet more than once). Unfortunately cost of RDS databases, even medium instances is a bit too high for small hobby project.

So I decided to store data in files and use AWS S3 for that. Potentially I would even use file system of EC2 instance, but I still prefer some separation and maybe later change data storage format and use Athena on top of S3.

So I had to rewrite part of the code that does data collection. The script does the next job, running daily:

* collect last 25 tweets posted by ~90 politicians
* Update the database (now not really database, but data file) with new information

Since I’ve decided to store data files on AWS S3, I found a handy package that helps to work with S3 from R.

To use it, first you need to create a user with programmatic access to S3 bucket in your AWS IAM interface.

Write down somewhere your Access key, Secret Access Key and AWS region, you’ll need that for connecting. The connection details are store in the environment.

library("aws.s3")

# define s3 connection

Sys.setenv("AWS\_ACCESS\_KEY\_ID" = "ACCESS KEY",

"AWS\_SECRET\_ACCESS\_KEY" = "SECRET",

"AWS\_DEFAULT\_REGION" = "ap-southeast-2")

Package documentation also asks for AWS\_SESSION\_TOKEN parameter, but I have found that it isn’t mandatory, my connection worked without them.

You then may check access to what buckets do you have, you with bucketlist command:

bucketlist()

If you see list of the buckets, you set the connection right.

Code that loads a data file from S3 Bucket looks like

s3load("users.RData", bucket = "auspolrappdata")

Code that saves data to S3:

s3save(t, bucket = "auspolrappdata", object = "tweets\_app.RData")

I decided for now for simplicity to save data in RData files.

There were couple things that took a bit of time to solve when I transitioned from DB to files

**Avoid duplication of data**

With DB I had a primary key based on twitter status\_id, so DB took care of duplication. Now I store data in R data frame (and then in file), so that nice feature isn’t available anymore.

There is a bigger data frame with historical data, now it has over 80,000 rows.

Then for each politician every day we collect the last 25 tweets. Some of that tweets (actually most of them) already exist in the database, not so many politicians post extremely actively.

Hence out of 25 tweets, usually 1 to 5 are new and other 20-24 are already in the database. So the challenge is to add just new tweets and ideally update old tweets (for example that old tweets can get more likes or retweets).

I’ve spend fair bit of time trying to find an elegant function in base R or tidyverse that can append new tweets and update old tweets, but couldn’t find anything out of the box. I couldn’t find anything supporting unique primary key. Hence I had to develop an own function doing that job.

t\_join <- function (t\_old, t\_new){

t\_new\_dup <- t\_new %>% filter(t\_new$status\_id %in% t\_old$status\_id)

t\_new\_new <- t\_new %>% filter(!(t\_new$status\_id %in% t\_old$status\_id))

t\_old[match(t\_new\_dup$status\_id, t\_old$status\_id), ] <- t\_new\_dup

t\_old <- rbind(t\_old, t\_new\_new)

return(t\_old)

}

It splits 25 tweets that we found to 2 parts: those that exist in bigger dataframe and another that are completely new and then updates existing rows and adds new.

Another a bit tricky part was that in my original realization the data was automatically transformed in the format that DB supports. Appeared that it was different from coming from Twitter API even after flattering.

So since I have most of the data that way, I had to transform new data that I’ve just pulled from Twitter to the same format with another new function.

tformat <- function (tw\_data){

tw\_fixed <- tw\_data %>% mutate (created\_at = as.character(created\_at),

is\_quote = as.integer(is\_quote),

is\_retweet = as.integer(is\_retweet),

symbols = as.integer(symbols),

urls\_url = as.character(urls\_url),

urls\_expanded\_url = as.character(urls\_expanded\_url),

quoted\_created\_at = as.character(quoted\_created\_at),

quoted\_verified = as.integer(quoted\_verified),

retweet\_created\_at = as.character(retweet\_created\_at),

retweet\_verified = as.integer(retweet\_verified),

protected = as.integer(protected),

account\_created\_at = as.character(account\_created\_at),

verified = as.integer(0),

[urls\_t.co](http://urls_t.co) = as.character([urls\_t.co](http://urls_t.co)),

account\_lang= as.character(account\_lang))

return(tw\_fixed)

}

The rest was rather trivial. You can always use the package Library(“aupolitics”)

I’ve also had to update my Shiny App, to take data from the file rather than DB, but it wasn’t hard.

The changes are done, my Shiny App is up and running and I will save few hundreds on AWS bills.

