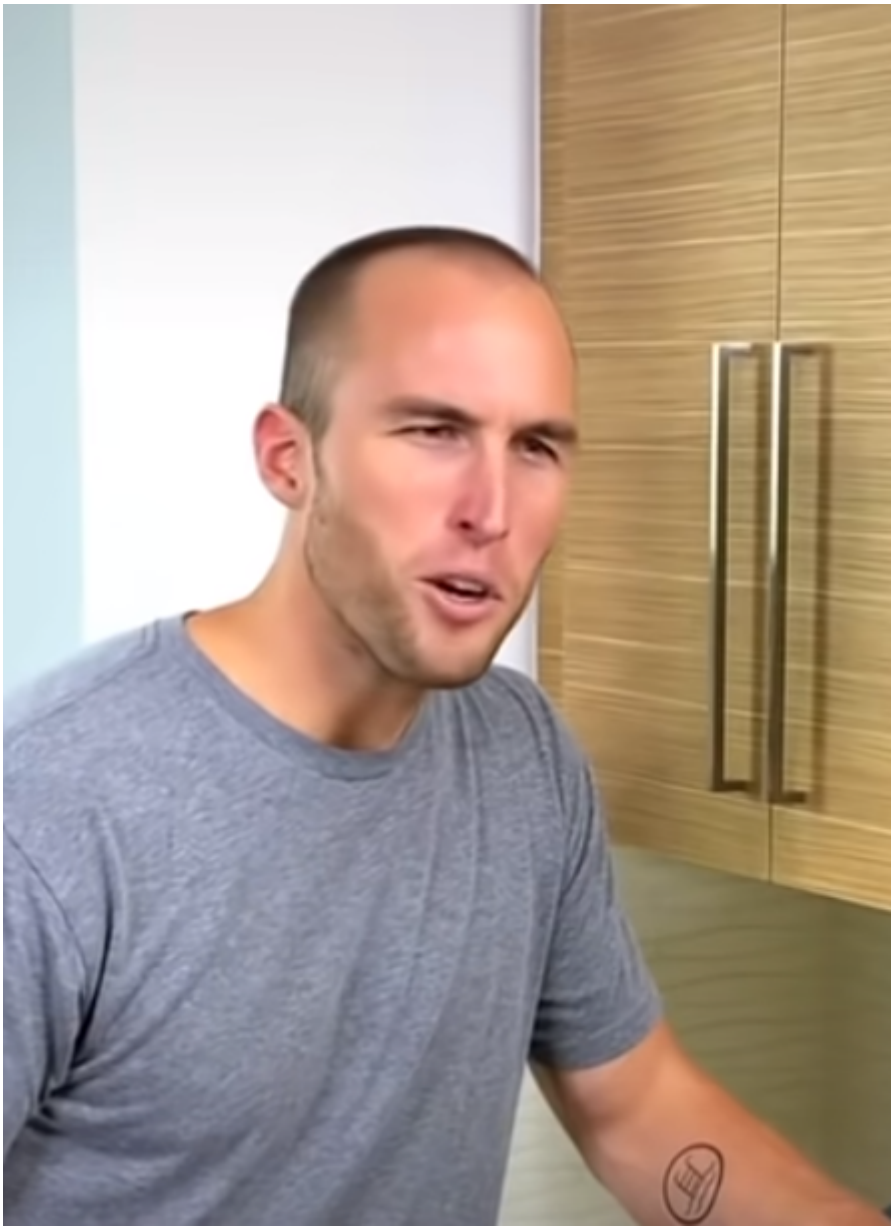
Building your own knitr compile farm on your Raspberry Pi with {plumber}

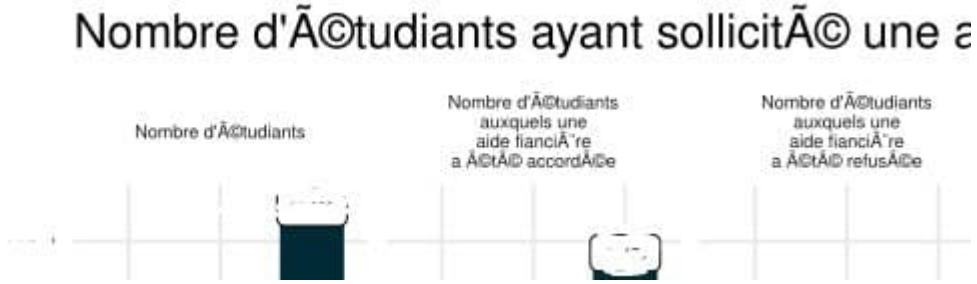
2021/06/04 R



Rage is my fuel

I've had the {plumber} package on my radar for quite some time, but never tried it. However, a couple of weeks ago, I finally had a reason to try it out and see how the package works.

One of my main problems in life is that my work laptop runs Windows, and my second problem is that I need to compile LaTeX documents (via Rmarkdown) on Windows, and it's just a pain. Not because of Rmarkdown, nor LaTeX, but because of Windows. Windows and UTF-8 don't mix well, and I've grown so frustrated that I thought about creating my own Rmarkdown knitr compile farm using my Raspberry Pi 4 to solve this issue. The idea would be to send in the encrypted .Rmd file and get back an encrypted .pdf file. Dear reader, you surely think that this is overkill; let me assure you, it is not. I have wasted so much time on Windows because Windows is a joke that cannot properly handle THE MOST COMMON TEXT ENCODING IN THE UNIVERSE that this the only way out. Even Yihui Xie, the creator of the {knitr} package (among many others), wrote a blog post titled My Biggest Regret in the knitr Package, in which he explains how Windows' crappy handling of UTF-8 made him make a regrettable decision. The issue Yihui Xie discusses is now resolved since {rmarkdown} version 2, as stated in the release notes (ctrl-f "utf-8"), but, for some reason, I still have problems with UTF-8 on Windows. While it is a fact that characters like the french é, è, ô, ç etc are now properly shown in a compiled document, any such character in a plot will not show properly, as you can see in the screenshot below:



I did not really ever notice this issue in the past because I wrote 100% of my documents in English, but now that I'm a public servant in a country where French is the administrative language, man, am I having a bad time.

Now, I make sure my .Rmd files are encoded in UTF-8, but I still get issues with plots. I tried changing the graphics device to Cairo or {ragg}, but I still have these issues.

Who knows, maybe this is also a case of PEBKAC, but in that case it's still Windows' fault for making me feel bad.

Anyway, this was reason enough for me to start developing an API that would allow me to get a nice looking PDF compiled on a serious operating system.

Getting started: Docker

I started by writing a prototype on my local machine that (sort of, but not really) worked, but to put it on my Raspberry Pi I wanted to create a new Docker image to make deployment easier. For this, just like I did for this other blog post, I wrote

a 'Dockerfile' and pushed an image to Docker Hub. The Dockerfile is heavily inspired by hvalev's Dockerfile, and also by the official plumber one you can find here. I then built the image on my Raspberry Pi.

You can use the Dockerfile to build your own image, which you can find here, or you can pull the one I pushed on Docker Hub. Now, something important: this Docker image does not contain my plumber.R file. So the first time you're going to run it, it'll fail. You'll need to make one further adaptation on your server first.

Put your plumber.R where you want, and copy the path to the file. For instance, suppose that you put the file at: /path/to/your/apis/plumber.R. Then, you can finally run the image like so:

```
docker run -d -it -p 8000:8000 -v /path/to/your/apis:/srv/plumber/ --rm --name tex
```

Docker looks for a plumber file inside /srv/plumber/ but that's inside the image; this path gets sort of linked to your /path/to/your/apis/ and thus the plumber.R file you put there will be run. You can also put this there beforehand, adapt the Dockerfile and then build the image. It's not the most elegant way to do it, but hey, I'm a beginner.

These instructions are very general and independent from my API I'm discussing here. What follows will be specific to my API.

An API that ingests an Rmd file and spits out a compiled document

First of all, none of this would have been possible without the following Stackoverflow threads and Github repos:

- https://stackoverflow.com/questions/63808430/r-plumber-getting-as-excel-xlsx/63809737#63809737
- https://github.com/ChrisBeeley/reports_with_plumber/blob/master/plumber.R
- https://stackoverflow.com/questions/64639748/how-to-upload-a-xlsx-file-in-plumber-api-as-a-input

and Bruno Tremblay's help on this thread I made calling for help. You'll probably notice that the answers in the stackoverflow threads all come from Bruno Tremblay, so a big thank you to him!

With his help, I was able to clob together this API:

```
#* Knit Rmarkdown document
#* @param data:file The Rmd file
#* @param string The output format
#* @post /knit
# We use serializer contentType, the pdf serializer is the plot output from grDevi
# Since the content is already in the right format from render, we just need to se
```

```
  # the content-type
  #* @serializer contentType list(type = "application/gzip")
  function(data, output_format) {
    # Save the RMD file to a temporary location
    rmd_doc <- file.path(tempdir(), names(data))
    writeBin(data[[1]], rmd_doc)
    # render document to the selected output format
    # (file will be saved side by side with source and with the right extension)
    output <- rmarkdown::render(rmd_doc, output_format)
    tar("output.tar.gz", normalizePath(output), compression = "gzip", tar = "tar")
    # remove files on exit
    on.exit({file.remove(rmd_doc, output, "output.tar.gz")}, add = TRUE)
    # Include file in response as attachment
    value <- readBin("output.tar.gz", "raw", file.info("output.tar.gz")$size)
    plumber::as_attachment(value, basename("output.tar.gz"))
  }
```

This will go inside the `plumber.R` script. When the Docker image is running, you can hit the endpoint `/knit` to knit a document. But before discussing how to hit the API, let's go through the above code.

```
  function(data, output_format) {
    # Save the RMD file to a temporary location
    rmd_doc <- file.path(tempdir(), names(data))
    writeBin(data[[1]], rmd_doc)
```

This function takes two arguments: `data` and `output_format`. `data` is your Rmd file (I should have named this better… oh well) that you will send via a POST. The Rmd will get written to a temporary location. In a previous version of the function I've used `writeLines` instead of `writeBin` which works as well.

The next lines render the output as the provided output format (through the second argument, `output_format`) and the output file gets compressed to a `tar.gz` archive. Why? The first reason is, obviously, to save precious bandwidth. The second, most important reason, is for the API to be able to download it.

```
    output <- rmarkdown::render(rmd_doc, output_format)
    tar("output.tar.gz", normalizePath(output), compression = "gzip", tar = "tar")
```

The way I understand how this works, is that if you want your API to return an attachment, you need to set the right content type. This is done by decorating the function with the right serializer:

```
#* @serializer contentType list(type = "application/gzip")
```

At first I only wanted PDF files, and thus set the `pdf` serializer. This was a mistake, as the `pdf` serializer is only used if the API is supposed to return a plot (in the pdf format). When this was pointed out to me (in the Rstudio forums), Bruno Tremblay showed me the right solution:

```
#* @serializer contentType list(type = "application/pdf")
```

which worked! However, I then thought about how I would make the API more flexible by allowing the user to compile any format, and this is when I thought about compressing the file and returning a `tar.gz` file instead.

The first line of the final lines:

```
  on.exit({file.remove(rmd_doc, output, "output.tar.gz")}, add = TRUE)
  # Include file in response as attachment
  value <- readBin("output.tar.gz", "raw", file.info("output.tar.gz")$size)
  plumber::as_attachment(value, basename("output.tar.gz"))
```

simply clean up after exiting. The final lines read in the compressed file in a variable called `variable` which then gets downloaded automatically as an attachment.

Ok, so now, how do I get a document compiled? With the following script:

```
library(httr)
library(magrittr)

my_file <- "testmark"

res <-
  POST(
    "http://url_to_compile_farm:8000/knit?output_format=html_document",
    body = list(
      data = upload_file(paste0(my_file, ".Rmd"), "text/plain")
    )
  ) %>%
  content()

names(res)

output_filename <- file(paste0(my_file, ".tar.gz"), "wb")
writeBin(object = res, con = output_filename)
```

```
close(output_filename)
```

This script is saved in a folder which also contains testmark.Rmd, which is the Rmarkdown file I want to compile (and which gets sent to the server as the data argument). You'll notice in the url that the second argument from my API is defined there:

```
"http://url_to_compile_farm:8000/knit?output_format=html_document"
```

you can change html_document to pdf_document or word_document to get a PDF or Word document respectively.

I'm pretty happy with this solution, even though it's quite rough, and still needs some adjustments. For instance, I want to make sure that I can leave this API running without worry; so I need to build in some authentication mechanism, which will probably be quite primitive, but perhaps good enough. I also need to send and receive encrypted documents, and not plain text.

Further reading

If you're into tinkering with Raspberry Pi's, Rstudio Server an {plumber}, Tyler Littlefield has a pretty cool github repo with lots of interesting stuff. Definitely give it a look!

Hope you enjoyed! If you found this blog post useful, you might want to follow me on twitter for blog post updates and buy me an espresso or paypal.me, or buy my ebook on Leanpub. You can also watch my videos on youtube. So much content for you to consoom!

LATEST POSTS

The theme this blog uses is a slight variation of the Smol theme.

Back to main page.