

[About](#) [Projects](#) [Talks](#) [Blog](#) [Links](#)

## Automatically deploying a Shiny app for browsing #RStats tweets with GitHub Actions

This tutorial blog will show you how to use GitHub actions to automatically update a data source and re-deploy a Shiny app, with an example Shiny application which uses `{rtweet}` to browse R-related tweets.

October 3, 2022

A little while ago I saw this [blog post](#) by Craig Hamilton shared on Twitter. In it, he describes the process of building a Shiny app to search through his favourited tweets, after noting the difficulty of searching through tweets he had liked to find a link he wanted to look at later. Inspired by the blog post, I decided to build a similar Shiny app for my own tweets. This blog post is the story of how it went.

{shinytweet} Favourite Tweets				
Date	User	Tweet	URL	Link
2022-09-24	@WeAreRLadies	1/2 My week as curator is over. I really enjoyed to talk about freelancing and #accessibility in #dataviz. If you enjoyed this week discussions you can follow me @rgiordano79 or if you want discuss, collaborate or say hello check also my website <a href="https://t.co/74HVKEkJeV">https://t.co/74HVKEkJeV</a> #RLadies.	<a href="https://twitter.com/WeAreRLadies/status/1573681397247647745">https://twitter.com/WeAreRLadies/status/1573681397247647745</a>	<a href="https://t.co/74HVKEkJeV">https://t.co/74HVKEkJeV</a>
2022-09-24	@MrPecners	And now, smolder! 🔥 Thanks to the tip from @researchremora, I've upped my samples from 300 to 400. Noticeable increase in quality with acceptable increase in render time 🙌 #rstats + #rayshader code: <a href="https://t.co/jHJg3rP1vz">https://t.co/jHJg3rP1vz</a> <a href="https://t.co/6a8RA1y2C5">https://t.co/6a8RA1y2C5</a> <a href="https://t.co/J7IRBJr2m9">https://t.co/J7IRBJr2m9</a>	<a href="https://twitter.com/MrPecners/status/1573679967988236289">https://twitter.com/MrPecners/status/1573679967988236289</a>	<a href="https://t.co/jHJg3rP1vz">https://t.co/jHJg3rP1vz</a>
2022-09-24	@WeAreRLadies	Regarding colours there are other options to give a try. One suggested by @nrennie35 <a href="https://t.co/mLnWgNNPso">https://t.co/mLnWgNNPso</a> #RStats <a href="https://t.co/0UBZ2nC25g">https://t.co/0UBZ2nC25g</a>	<a href="https://twitter.com/WeAreRLadies/status/1573672178976956418">https://twitter.com/WeAreRLadies/status/1573672178976956418</a>	<a href="https://t.co/mLnWgNNPso">https://t.co/mLnWgNNPso</a>
2022-09-24	@NearandDistant	#TidyTuesday Week 38: WWTIP In	<a href="https://twitter.com/NearandDistant">https://twitter.com/NearandDistant</a>	<a href="https://t.co/TiuvuWlHWX1">https://t.co/TiuvuWlHWX1</a>

## Building the Shiny app

The Shiny app itself is actually relatively simple in comparison to some other Shiny apps I've worked on - it will only contain a table of data. So first of all, we need to get that data...

### 1. Obtain the Twitter data

The [blog post](#) by Craig Hamilton covers how to get the Twitter data using the `{rtweet}` package, so I won't reiterate it all here. Essentially, the steps we follow are:

- use the `get_favorites()` function from `{rtweet}` to get data on the tweets that I have liked. You'll need to have a Twitter developer account in order to be able to authenticate through `{rtweet}`;
- filter the tweets returned to include only those containing phrases such as *rstats*, *tidyverse*, or *rstudio*, **and** a link to an external source e.g. a link to a GitHub repository or a blog post;
- perform a little bit of data wrangling to tidy up the URLs;
- save the data as, e.g. a `.rds` file.

One thing that did trip me up a little: it wasn't obvious how to get the data on the author of the tweets I had liked using the `get_favorites()` function. It turns out if you use the default setting of `parse = TRUE` in `get_favorites()`, the

author information isn't returned in the tidied tibble. Instead, I had to set `parse = FALSE` to get the non-tidy json data from the Twitter API, and extract it manually.

*Update: the maintainer of {rtweet}, [Lluís Revilla Sancho](#), pointed out that you can use `users_data()` to get the author data, and then `cbind()` to join in to the favourites data, when `parse = TRUE`.*

## 2. Write the UI code

As I said above, the UI for the app is simple - it only contains a table made using {reactable}:

Copy

```
1 ui <- navbarPage(
2
3   title = "{shinytweet}",
4
5   theme = bs_theme(version = 4,
6                     bootswatch = "minty",
7                     primary = "#12a79d"),
8
9   tabPanel("Favourite Tweets",
10    reactable::reactableOutput("table_output")
11  )
12 )
```

Here, we added some minimal styling with `bs_theme()` - entirely optional. At some point in the future, I may add in some options to filter by topic, time range, or users. However, for this initial attempt I decided to keep it incredibly simple.

## 3. Write the server code

The code for the server is a little bit more complicated, though not by much. After reading in the data saved in the first step, we construct the `table_output`. Before creating the reactable table, there's a little bit of data wrangling involved: selecting only the columns of our data we want to display, renaming some of those columns to something that's more human-readable, and converting the date of the tweets to a date object. Due to the (not so friendly) format of the date that comes out of the Twitter API when we set `parse = FALSE` in `get_favourites()`, we use a non-exported function (`rtweet:::format_date(Date)`) from {rtweet} for the re-formatting.

Copy

```
1 # read in data
2 likes <- readRDS('likes.rds')
3
4 # build server
5 server <- function(input, output) {
6
7   output$table_output = reactable::renderReactable({
8     table_df <- likes %>%
9       select(created_at, user, full_text, tweet_link, content_url) %>%
10       rename(Date = created_at,
11             User = user,
12             Tweet = full_text,
13             URL = tweet_link,
14             Link = content_url) %>%
15       mutate(Date = rtweet:::format_date(Date),
16             Date = as.Date(Date))
17     reactable::reactable(table_df,
18                         columns = list(
19                           Date = colDef(align = "center",
20                                         minWidth = 60),
21                           User = colDef(cell = function(User) {
22                             htmltools::tags$a(href = paste0("https://twitter.com/", as.character(User)),
23                                                  target = "_blank", paste0("@", User))
24                           },
25                           minWidth = 60),
26                           Tweet = colDef(align = "left",
27                                         minWidth = 120),
28                           URL = colDef(cell = function(URL) {
29                             htmltools::tags$a(href = as.character(URL),
30                                                  target = "_blank", as.character(URL))
31                           })
32     )
33   })
34 }
```

```

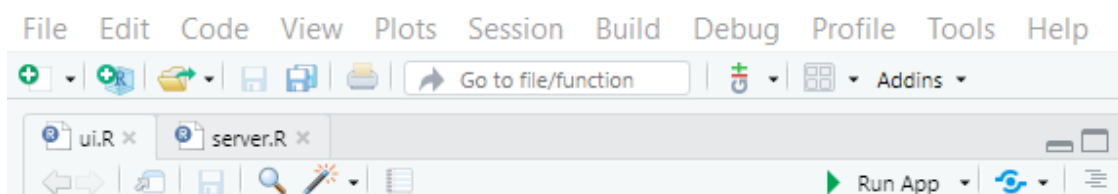
31     }},
32     Link = colDef(cell = function(Link) {
33         htmltools::tags$a(href = as.character(Link),
34                           target = "_blank", as.character(Link))
35     })),
36     striped = TRUE,
37     defaultPageSize = 8)
38 })
39 }

```

To make the links in the table clickable, within the definition of the columns, we use `htmltools::tags$a`. To make the Shiny app a little bit more user-friendly, we also set `striped = TRUE` and `defaultPageSize = 8` to make different tweets more distinguishable, and limit the number of tweets in one page.

## Deploying the Shiny app

Now that we've built the shiny app and got it working locally, we want to deploy it to [shinyapps.io](https://shinyapps.io). If you're working in RStudio IDE, the easiest way to deploy a Shiny app is to click the deploy button (next to Run App). If it's the first time you've deployed to shinyapps.io, you'll be prompted to authenticate. Alternatively, you can call `deployApp()` in the console to do the same thing.



However, this approach means that every time we want to make a change to our app, we manually need to re-deploy it - that's a bit annoying. Instead, since the code is stored on GitHub, we can use [GitHub Actions](#) to automatically deploy our app for us, every time our code is pushed onto the main branch. This [blog post](#) from [Roel Hogervorst](#) was incredibly helpful, and I didn't have to make too many adjustments to get this up and running.

The basic steps to get GitHub Actions up and running:

### 1. Create a new token on [shinyapps.io](https://shinyapps.io)

Even if you've already got a token and authenticated your app deployment to shinyapps.io, I think it's a good idea to create a separate token, just for this project. To create a new token, click on your profile in the top right, then click *Tokens*. Click *Add Token* to create a new token.

### 2. Create a deploy script

Create a deploy script that passes the tokens to `deployApp()`. For me this script was called `deploy.R`, and looked a bit like this:

Copy

```

1 # Authenticate
2 setAccountInfo(name = Sys.getenv("SHINY_ACC_NAME"),
3               token = Sys.getenv("TOKEN"),
4               secret = Sys.getenv("SECRET"))
5 # Deploy
6 deployApp(appFiles = c("ui.R", "server.R", "likes.rds"))

```

In the `deployApp()` call, you can also specify which files make up your app. This may just be a single `app.R` file, or it may be multiple files as in this example.

### 3. Create a dockerfile

We're going to use Docker to deploy our Shiny app. Docker is a platform that allows you to build, test, and deploy applications by packaging everything your software needs to run into a *container*. This includes system dependencies, R packages, and code. If you're new to Docker, the [documentation](#) is good place to start.

We create a Dockerfile that contains the instructions for how to build the container.

Copy

```
1 FROM rocker/shiny:4.2.1
2 RUN install2.r rsconnect tibble dplyr stringr rtweet htmltools lubridate bslib reactable
3 WORKDIR /home/shinytweet
4 COPY ui.R ui.R
5 COPY server.R server.R
6 COPY likes.rds likes.rds
7 COPY deploy.R deploy.R
8 CMD Rscript deploy.R
```

Essentially, the Dockerfile gives the instructions to install all the R packages we need, copy the files relating to our Shiny app, and then run the deploy script. The line `WORKDIR /home/shinytweet` defines the name of our app in the shinyapps.io url: it will be deployed to [nrennie35.shinyapps.io/shinytweet](https://nrennie35.shinyapps.io/shinytweet).

#### 4. Create a workflow for GitHub Actions

Then, we need to set up a GitHub Action that tells GitHub to auto-deploy our app when something changes on main. In a folder called `.github/workflows`, we create a file called `deploy-shinyapps.yml`. The name of the file doesn't really matter, just that the file type is `.yml`, since this is how we write a workflow for GitHub Actions. This workflow is adapted from Roel Hogervorst's [blog post](#) mentioned above.

Copy

```
1 name: Run on push master, main
2
3 # Controls when the action will run.
4 on:
5   # Triggers the workflow on push or pull request events but only for the main branch
6   push:
7     branches: [ main, master ]
8
9   # Allows you to run this workflow manually from the Actions tab
10  workflow_dispatch:
11
12 # A workflow run is made up of one or more jobs that can run sequentially or in parallel
13 jobs:
14   # This workflow contains a single job called "build"
15   build:
16     # The type of runner that the job will run on
17     runs-on: Ubuntu-20.04
18
19     # Steps represent a sequence of tasks that will be executed as part of the job
20     steps:
21       # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
22       - uses: actions/checkout@v2
23
24       # build the docker image and give it the name main
25       - name: Build image
26         run: docker build -t main .
27
28       # run the docker image supply the secrets from the github secrets store.
29       - name: execute
30         run: >
31           docker run
32             -e SHINY_ACC_NAME=${{ secrets.SHINY_ACC_NAME }}
33             -e TOKEN=${{ secrets.TOKEN }}
34             -e SECRET=${{ secrets.SECRET }}
35           main
```

We give the action a name:, and tell it when to run in the `on:` section. In this case, we want it to run when something is pushed to main, or if we manually trigger it on GitHub. In the `jobs:` section, this GitHub action sets up R, adds the R packages we need, then builds the Docker container.

## 5. Add GitHub Secrets

Finally, to make sure that GitHub Actions can authenticate shinyapps.io account, we pass the three variables needed in our `deploy.R` script as GitHub Secrets. GitHub Secrets are found in the settings tab for the repository. Make sure the name of the secret is the same as the variable passed into `deployApp()`. In this case, we add three secrets: `SHINY_ACC_NAME`, `TOKEN`, and `SECRET`.

## Updating the Shiny app

At the moment, our deployed Shiny app is essentially a static screenshot of tweets I had liked up to the point we deployed the app. Of course, I could manually pull the most recent data, and upload it to GitHub where it would be automatically re-deployed. But we can use GitHub actions to automate that as well. There are two steps to running an update:

### 1. Create an update script

In the same way that we created a `deploy.R` script to deploy our Shiny app, we can create an `update-data.R` script that, when run, updates the data. The code in this script is very similar to the process of initially grabbing the data through `{rtweet}` so I won't go into a lot of detail here. However, the code is available on [GitHub](#) if you're especially interested.

Basically, we want to grab tweets I liked since the last one recorded in the current data set, append that to the old data, and overwrite the data file.

### 2. Set up GitHub Actions

Now, we set up another GitHub Action to run the `update-data.R` script every day and push the data to main, which triggers a re-deploy of the app. This GitHub Action is a little more custom, and took a bit longer to figure out than the deployment actions, so I'll go into a bit more detail here. First, define a new `.yaml` file in the `.github/workflows` folder.

Define the name of the action and when it will run. `cron: "0 8 * * *"` tells GitHub to run it everyday at 8am. Adding `workflow_dispatch:` allows me to also trigger the workflow from the actions tab in GitHub. This is useful when debugging the initial setup, and also if I want to trigger a data refresh at a different time of day for some reason.

Copy

```
1 name: Update data
2
3 on:
4   schedule:
5     - cron: "0 8 * * *"
6   workflow_dispatch:
```

Next, we define the jobs and set up R. This part is exactly the same as the deployment action.

Copy

```
1 jobs:
2   update:
3     runs-on: ubuntu-latest
4     env:
5       GITHUB_PAT: ${ secrets.GITHUB_TOKEN }
6     steps:
7       - uses: actions/checkout@v2
8
9       - uses: r-lib/actions/setup-r@v2
10        with:
11          use-public-rspm: true
```

Then, we want to install any R packages our `update-data.R` script requires to run. In the deployment action, we did this through the `Dockerfile`, but here we write it directly into the GitHub actions file.

Copy

```
1 - name: Install dependencies
2   run: |
3     install.packages(c("rtweet", "dplyr", "tibble",
4       "stringr", "lubridate", "readr"))
5   shell: Rscript {0}
```

Then we run our update script. Since `{rtweet}` requires authentication, we need to add another GitHub secret (`TWITTER_BEARER`) containing the token that gets passed to `{rtweet}`. The process is very similar to adding a token for authenticating shinyapps.io.

Copy

```
1 - name: Update data
2   run: |
3     source("update_data.R")
4   shell: Rscript {0}
5   env:
6     BEARER: ${ secrets.TWITTER_BEARER }
```

Finally, we push the changes to main. Once the new `.rds` file has been created in the `update_data.R` script, we need to push the changes to GitHub (in exactly the same we would if we ran it locally). The following step adds, commits, and pushes the changes to main, which triggers the running of the deploy workflow that's already defined. The Shiny app is then updated and re-deployed with the new data.

Copy

```
1 - name: Commit files
2   run: |
3     git config --local user.email "actions@github.com"
4     git config --local user.name "GitHub Actions"
5     git add --all
6     git commit -m "add data"
7     git push
```

And that's it! The deployed app is available at [nrennie35.shinyapps.io/shinytweet](https://nrennie35.shinyapps.io/shinytweet), and the source code is available on [GitHub](#). I hope you've found this blog post a useful case study in automating some of your Shiny app workflows - now you can relax while GitHub Actions does your work for you!



---

For attribution, please cite this work as:

**Automatically deploying a Shiny app for browsing #RStats tweets with GitHub Actions.**  
Nicola Rennie. October 3, 2022.

Licence: [creativecommons.org/licenses/by/4.0](https://creativecommons.org/licenses/by/4.0)

1 Comment - powered by [utteranc.es](#)

shaenzi commented on 16. März 2023

Thanks so much for your super-helpful blogposts in general, but this one in particular! I've been using github actions both for personal apps and at work every since!


One thing that cost me a couple of hours: the push to the repo did not work until I changed the settings in the github repo (which I did not know about...) that allows github actions to modify the repo. Maybe this saves other people some time! Plus I think the general `GITHUB_PAT` does not trigger github actions that are supposed to start on push, which is why I ended up also scheduling my deployment. I guess the alternative would be to add a personal access token as a secret and use that.

anyway, thanks again!

Write

Preview

Sign in to comment

 Styling with Markdown is supported

Sign in with GitHub

[← Introducing {PrettyCols}. Using functional analysis to model air pollution data in R →](#)