

# **A simple five-ish step process to get your Shiny app online (securely).**

**Github, Docker-compose, EC2 version**

Ronald (Ryy) Glenn Thomas

3/26/23

## **Table of contents**

<b>Introduction</b>	<b>2</b>
<b>Methods</b>	<b>2</b>
<b>Hosting</b>	<b>4</b>
<b>Select a hosting service</b>	<b>5</b>
<b>Website</b>	<b>6</b>
<b>Github</b>	<b>9</b>
<b>Appendix-1</b>	<b>9</b>
<b>Appendix-4 {#appendix-4} Set up github repo</b>	<b>10</b>
Tip 2: Add ubuntu to the docker group to allow docker to run without sudo. . . . .	12
Tip 3: . . . . .	12
<b>References</b>	<b>13</b>
Appendix Note: 2023-03-22 . . . . .	13

Photo by Nathan Waters on Unsplash



Figure 1: under construction

## Introduction

This is the first in a series of posts offering suggested strategies for leveraging open source technologies to provide straightforward solutions to one of the central challenges in the practice of data science, i.e. how to effectively communicate analysis results to clients and collaborators. The list of open-source technologies (software stack) we suggest for employment is: linux, R, Shiny, Docker, Git, and Caddy. In this post we'll make use of two cloud services Github and AWS. Further posts will describe alternate constructions, e.g. using the low cost cloud service: Hetzner.

Also described in other posts are strategies that avoid Github. This approach provides a simpler initial construction, but a more labor intensive updating process.

This initial post provides a minimal, proof-of-concept example of how to apply these technologies for hosting an interactive Shiny application.

In the following we start with a very simple, but hopefully still useful, stand-alone Shiny app developed on our local workstation. Then after some straightforward interfacing with the Amazon web service environment, We push the Shiny app into the cloud, and end up with a secure (encrypted and authenticated) app running on a website with a custom domain name.

## Methods

To begin, lets assume we're just finished developing a new Shiny app, named `power1_shiny` . (The methods described here apply generically to any Shiny app, but we'll use one of our own for illustration). See the R/Shiny code for our `power1_shiny` app (`power1.R`) [here](#) in appendix 1.

We can test the app locally in your development directory by running it with the following command.

```
R -e "library(shiny); runApp('power1_shiny.R', launch=T)"
```

This command will run the R program, load the Shiny package, and launch the app in your default browser.

Figure 2 below shows the Shiny app running locally in a browser, it consists of a widget to select the sample size and provide a dynamic visualization (2D plot) of the power as a function of the standardized effect size:

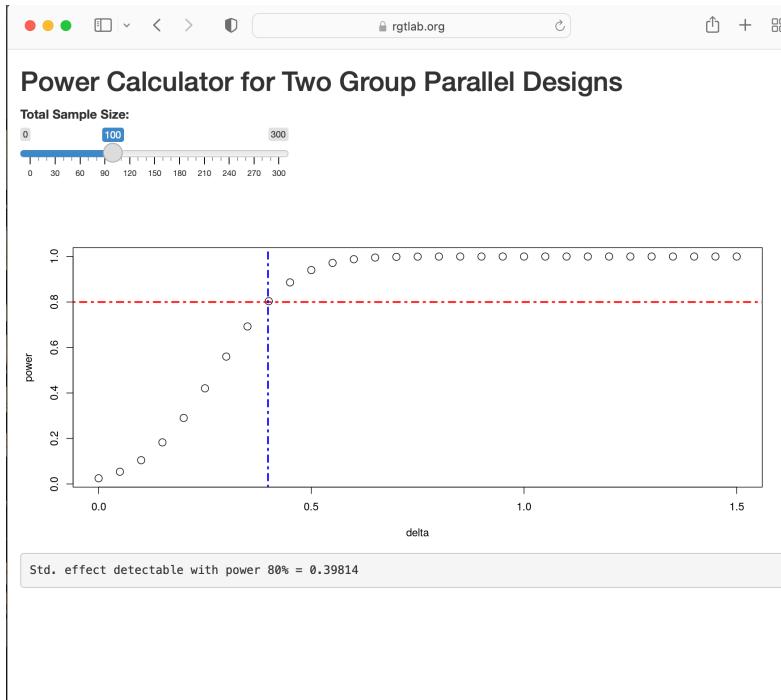


Figure 2: *Shiny app*

Once we determine our app is working as designed, the next step is to set up a secure hosting environment on a virtual server. By hosting the app on a (virtual) server we simply need to send a link and security credentials to our collaborators. There are many ways to accomplish this. Here we'll demonstrate a straightforward and efficient approach using mainstream cloud services and open source tools. That is, we'll describe how to 'spin' up a server on Amazon Web Service EC2 and in just a few steps, through the application of Docker, R, Shiny, and Caddy (webserver) functionality we'll have a fully functional secure web app to share with colleagues.

## Hosting

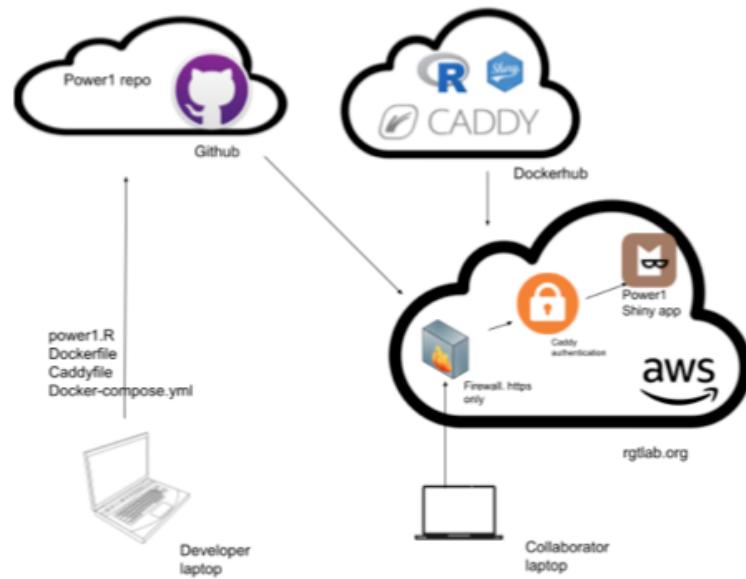


Figure 3: *Data flow*

Figure 3 illustrates the tools we'll use and the flow of program and configuration files. In order to host **power1** online we'll need to complete the following tasks:

1. create a virtual server (connected via ssh) with a firewall
2. obtain a static IP address (to identify the server online)
3. obtain a domain name (name for IP address)
4. install and configure a webserver (tool to interact with https protocol requests and respond)
5. obtain and install an SSL certificate (to allow encrypted communication)
6. setup an authentication method (password protection)
7. configure a reverse proxy method (translate https (port 443) requests to Shiny (port 3838))

At first glance these 7 requirements can appear daunting, but on closer inspection all can be met with relative ease and minimal cost ( using a cloud-hosting service, e.g. Amazon's EC2

or Digital Ocean, and a “leased” domain name from, e.g. Go-Daddy, or Amazon’s Route 53) or no cost( if you have your own server with IP address, and domain name)

## Select a hosting service

There are a number of cloud based server options: Microsoft Azure, Oracle, Google Cloud, Amazon AWS EC2, Digital Ocean to name a few. Each has their own approach to setting up a custom virtual server. Several have free or low-cost service tiers available.

An overview of the process with EC2 follows. (Detailed instructions for AWS EC2 are covered in the post “Setting up AWS server”. )

0. Create an account or sign in.
1. Set up an interactive environment with AWS server.
  - a. define ssh key-pair.
  - b. configure firewall.
  - c. request static IP.
  - d. obtain domain name.

Once the server is available connect via ssh and login, The only necessary software to install is docker, docker-compose and git. Install all 3 with the following command:

```
sudo apt install -y git  
sudo snap install docker.io
```

Once the host is set up and the requisite software installed we’ll have a customized virtual server wtih a static IP address, and unique domain name and firewall in place. In other words, items 1, 2, and 3 from our list will be taken care of.

## Website

To configure the web server and containerize our app we need to add three files to the repo, to go along with our Shiny app.

Its easiest to create files on the workstation and push then push the repo to github and from there we can access them from our server.

Details are in @appendix-1 Appendix 1.

These three configuration files are:

1. a Docker configuration file (default name `Dockerfile`)



Photo by Ian Taylor on Unsplash

We'll use docker to access R/Shiny, and docker-compose to access Caddy, our webserver. The first file is the dockerfile. Here is our minimal dockerfile with comments:

```
FROM rocker/shiny:4.2.0
COPY /power1_shiny/* /srv/shiny-server/
CMD ["/usr/bin/shiny-server"]
```

1. Grab the latest rocker/Shiny image from Docker Hub to use as a base image.

2. Copy the Shiny code to the default location for shiny-server
3. Run the Shiny-server using the default app code

This configuration file instructs Docker to build a container based on a Rocker/Shiny image (which itself is a ubuntu image with R and Shiny installed) then copy into the container the `power1_shiny.R` code and finally launch Shiny on (default) port 3838. We placed the `power1_app.R` code in the default location `/srv/shiny-server` we only need to start the server and it will find the shiny program.

2. a Caddy web server configuration file (default name `Caddyfile`)

We'll use `Caddy` as our web server. Caddy is an open-source tool that has the very useful feature of automating the acquiring and installing of an SSL certificate. An SSL cert is required by most browsers to use the encrypted communication protocol https.

Caddy is configured with a file named `Caddyfile`. We use the caddy configuration file to specify three critical things.

1. the site domain name.
2. the 'reverse proxy' map that redirects requests to port 443 (ssl port) to port 3838 (Shiny port).

Our barebones Caddyfile looks like this:

```
rgtlab.org {
    root * /srv
    handle_path /power1_shiny/* {
        reverse_proxy power1_shiny:3838
    }
    file_server
}
```

We can accomplish what we need for items 4, 5, 6 and 7 through the Caddyfile.

Note:

- `rgtlab.org` is our domain name

- `handle_path` maps all https requests to port 3838 where Shiny is listening.

Providing our servers domain name, `rctlab.org` is sufficient to initiate an exchange with `letsencrypt` to generates an SSL certificate.

And the third file is the docker compose file that containerizes our Shiny app, pulls a caddy webserver image from Docker Hub and creates a local network for the two containers to communicate in.

3. a Docker-compose configuration file (default name `docker-compose.yml`) (These are the default file names. If preferred, you can use custom names and point the program to the config file with command line options).

The `docker-compose.yml` file:

```
version: "3.7"

services:
  power1_shiny:
    build: .
    expose:
    - "3838"
  caddy:
    image: caddy:2.3.0-alpine
    ports:
    - "80:80"
    - "443:443"
    volumes:
    - $PWD/Caddyfile:/etc/caddy/Caddyfile
    - $PWD/site:/srv
    - caddy_data:/data
  volumes:
    caddy_data:
```

## Github

Once in place on your laptop push the four files (power1\_app.R, Dockerfile, Caddyfile, docker-compose.yml) to github

```
git push
```

and then ssh login to server and clone repo.

```
git clone https://github.com/joe47/power1_app.git
```

Lastly, cd to power1 directory and run

```
docker-compose up -d
```

and you're good to go!

The app **power1** can be accessed by ‘bob’ at the url

```
https://power1.net/power1
```

with password ‘thebunny’

## Appendix-1

Consider an app that is a balance of simple and functional – one that calculates the power for a 2-sample t-test as a function of the standardized effect size. re is our shiny app **power1.R**:

Consider the power1.R file:

```
ui <- fluidPage(  
  titlePanel("Power Calculator for Two Group Parallel Designs"),  
  sliderInput("N", "Total Sample Size:", min = 0, max = 300, value = 100),  
  plotOutput("plot"),  
  verbatimTextOutput("eff"))  
  
server <- function(input, output, session) {
```

```

delta = seq(0, 1.5,.05)
pow = reactive(sapply(delta, function(x) power.t.test(input$N, d=x)$power ))
eff = renderText(power.t.test(input$N, power=.8)$d)
output$plot <- renderPlot({
  plot(delta, pow(), cex=1.5, ylab="power")
  abline(h = .8, col = "red", lwd =2.5, lty = 4)
  abline(v = eff(), col = "blue",lwd =2.5, lty = 4)})
output$eff <- renderText(
  paste0("Std. effect detectable with power 80% = ", eff()))
})
shinyApp(ui, server)

```

The app is designed to be maximally minimal. Using only base R functions, with a minimum of reactive widgets and layout commands to keep it simple while still performing a useful function.

## **Appendix-4 {#appendix-4} Set up github repo**

Start by creating a repo for the app on github.

- login to github (screenshot)
- click on `new`. Then in `repository name` field enter `power1`. (Make the
- repo private, we only want to share with Bob at this point).
- create repo. Click `Create repository` green button at the bottom of the page.
- back on your laptop: clone the repo:

```
git clone https://github.com/joe47/power1.git
```

then you can ssh into the new server with

```
sh> ssh ec2
```

The screenshot shows the GitHub interface for creating a new repository. At the top, there's a navigation bar with links for 'Pulls', 'Issues', 'Marketplace', and 'Explore'. Below the navigation is a search bar and a user profile icon. The main content area is titled 'Create a new repository'. It includes fields for 'Owner' (set to 'rgt47') and 'Repository name' (set to 'power'). A note suggests using a short and memorable name like 'curly-computing-machine'. There's an optional 'Description' field containing 'repo to house power1 app files'. The repository type is set to 'Private'. Under 'Initialize this repository with:', the 'Add a README file' option is selected. Other options include 'Add .gitignore' (with a dropdown menu showing 'None') and 'Choose a license' (set to 'GNU General Public...'). A note at the bottom states that this will set 'main' as the default branch. A warning message indicates that the user is creating a private repository in their personal account. At the bottom right is a green 'Create repository' button.

## **Tip 2: Add ubuntu to the docker group to allow docker to run without sudo.**

### **Tip 3:**

Copy file contents to workstation without using editor with shell commands.  
\* Copy file from blog  
\* Issue the following shell command  
\* paste the file contents  
\* type EOF on a new line.

```
cat << EOF > Dockerfile
```

- install docker `sudo snap install docker`
- clone github repo `joe47/power1`. Note: You'll need to authenticate to github.

```
git clone https://github.com/joe47/power1.git
```

- back on laptop...
- add `Caddyfile` and `docker-compose.yml` files to `power1` repo. `Caddyfile`

```
power1app.org {
  basicauth * {
    bob JDJhJDE0JE1CQmRGaTA0ajY3bkZTLjRiWUZ4enVoZnVSQzVXVGVUMH1VcXJTaTRGYmpRQVFHLnYzN0tx
  }
  handle_path /power1/* {
    reverse_proxy power1:3838
  }
}
```

```
docker-compose.yml
```

```
version: "3.7"

services:
  power1:
    build: .
```

```
caddy:  
  image: caddy:2.3.0-alpine  
  ports:  
    - "443:443"  
  volumes:  
    - $PWD/Caddyfile:/etc/caddy/Caddyfile  
    - caddy_data:/data  
volumes:  
  caddy_data:
```

- update (push to) github repo.

```
git add *  
git commit -am "adding Caddyfile and docker-compose.yml"  
git push
```

- back on server

Lastly, cd to power1 directory and run

```
git pull  
docker-compose up -d
```

## References

- Setup Jupyter Notebook for R article.

## Appendix Note: 2023-03-22

To get docker functioning on Mac desktop

```
docker build -t test --platform linux/x86_64 .  
  
docker run -d -p 80:3838 --platform linux/x86_64 test
```