A straightforward strategy to get your Shiny app online, securely and continuously updated.

Github, Docker-compose, EC2 version

Ronald (Ryy) Glenn Thomas

5/24/23

Table of contents

Photo by Nathan Waters on Unsplash

1 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 Amazon Web Service (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 and docker-compose. 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 AWS 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.

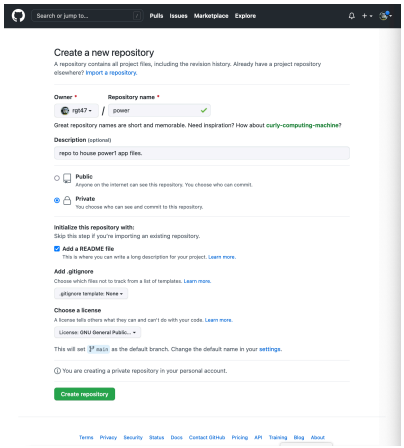
2 Methods

Start by creating a repository (repo) for the project. The best way to do this is to initiate the repo on Github and then `clone` it to your local workstation. Start by logging in to Github and creating a new empty repo, call it power1_app. On your local workstation navigate to your Shiny development directory, say `~/prj` and clone the power1_app repo from Github:

Details for creating a Github repo follow:

- login to github (screenshot)
- click on **new**. Then in **repository name** field enter **power1**. (Make the
- make the repo private, we only want to share with our client at this point).
- create the repo. Click **Create repository** green button at the bottom of the page.
- on your laptop cd to development directory, say ~/prj and clone the github repo:

```
git clone https://github.com/rgt47/power1_app.git
```



After cloning the repo to ~/prj/power1_app cd into the directory and create two new sub-directories, **power1_shiny** and **site**. These directories will house our shiny app and our web site landing page file, respectively.

Lets jump ahead to the point where you've 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 (**app.R**) here in appendix 1.

```
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)
```

We can test the app locally in our development directory, say `power1_app`, by runnning it with the following command.

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

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

Figure 1 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:

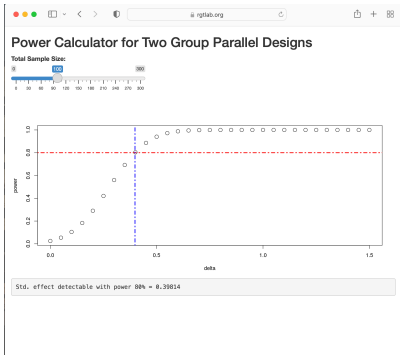


Figure 1: *Shiny app*

Once we determine our app is working as designed, the next step is to set up a secure hosting environment on a virtual server. Once the app is hosted the we simply need to send a link and security credentials to our collaborators for them to have secure access to the Shiny app. There are many ways to accomplish the hosting. Here we'll describe 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 we'll have a fully functioning secure web app to share with colleagues.

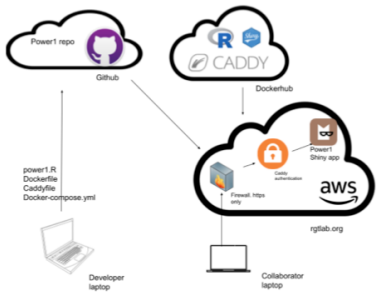


Figure 2: *Data flow*

2.1 Hosting

Figure 2 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. GoDaddy, or Amazon's Route 53) or no cost(if you have your own server with IP address, and domain name)

2.2 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 an earlier post here.

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.
    e. select an instance and launch server.

Once the server is available connect via ssh, and login,

The only necessary software to install is docker and git. Install both with the following commands:

```
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 a unique domain name and firewall in place. In other words, items 1, 2, and 3 from our `hosting` list above will be taken care of.

2.3 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.

We'll use a slightly indirect route to create and place the necessary files on the server but this approach will allow to do all our countinuing development on our local workstation and have the web app be automatically continually undated. We'll create the configuration files we need on our workstation and push them github and from there they can be accessed from our server.

These three configuation files are:

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

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:

Photo by Ian Taylor on Unsplash

```
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).
3. add login credentials for bob/vanilla47:

Our barebones `Caddyfile` looks like this:

```
# use caddy auth tool to generate a password via the `bcrypt` algorithm.
# > caddy hash-password --plaintext vanilla47
rgtlab.org {
    basicauth * /power1_shiny/* {
        bob $2a$14$pYWd5O7JqNeGLS4m4CKkzemM2pq5ezn9bcTDowofZTl5wRVl8NTJm
    }
    root * /srv
    handle_path /power1_shiny/* {
            reverse_proxy power1_shiny:3838
    }
    file_server
}
```

We can accomplish what we need for items 4, 5, 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, `rgtlab.org` is sufficient to initiate an exchange with the `letsencrypt` service to generates an SSL certificate.

And a 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`).

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:
```

Lastly, we need an html file, `index.html` in a subdirectory named `site` that provides the landing page for our server.

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Power1 app</h1>
    <ul>
      <li><a href="./power1_shiny/">Power1 app</a></li>
    </ul>
  </body>
</html>
```

At this point our `power1_app` repo looks like this:

```
.
   Caddyfile
   Dockerfile
   README.md
   docker-compose.yml
   power1_shiny
       app.R
   site
       index.html
```

3 Github

Push the new content to Github.

```
git push
```

Next login to the virtual server and clone the repo from Github.

```
ssh rgtlab.org
git clone https://github.com/rgt47/power1_app.git
```

Lastly, cd into `power1_app` directory and run

```
docker compose up -d
```

and you're good to go! The power1_shiny app is available at

```
https://rgtlab.org/
```

4 Appendices

4.1 Appendix-1

4.2 App.R

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_shiny.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.

4.3 Tip 1. Docker on M1 macbook.

To get docker functioning properly with `rocker` images on M1 Mac desktop use `--platform` option.

```
docker build -t power1_shiny --platform linux/x86_64 .
docker run -d -p 80:3838 --platform linux/x86_64 power1_shiny
```

4.4 Tip 2 add user to docker group on server.

Add ubuntu to the docker group to allow docker to run without sudo.

```
sudo usermod -aG docker ${USER}
```

4.5 Tip 3 ssh config file.

For convenience, construct a `config` file in `~/.ssh` as:

```
Host rgtlab.org
HostName 13.57.139.31 # static IP
User ubuntu # default user on ubuntu server
Port 22  # the default port ssh uses
IdentityFile ~/Downloads/power1.rsa
```

then you can ssh into the new server with

```
sh> ssh rgtlab.org
```

5 References

- Focus on R: a new qblog - Set up a virtual server on AWS (in anticipation of hosting Shiny apps)
- Shiny Apps with Docker Compose, Part 1: Development
- Shiny Apps with Docker Compose, Part 2: Production