A simple process to get your Shiny app online (securely).

Ronald (Ryy) Glenn Thomas

2023-06-20

Table of contents

1 Introduction

Photo by Nathan Waters on Unsplash

This is another in a series of posts suggesting straightforward, open-source strategies for effective communication of data analysis results to clients and collaborators.

In this post we propose a simple method for migrating a shiny app from your local workstation to the web.

The list of open-source technologies (software stack) we propose for use: linux, R, Shiny, Docker, and Caddy. In this post we'll make use of the AWS EC2 cloud service. In future posts we'll

describe alternate cloud constructions, e.g. using the low cost cloud service: Hetzner.

In the following we'll provide a proof-of-concept example of how to apply these technologies for securely hosting an interactive Shiny application on the web.

We start with a very simple, but hopefully still useful, standalone Shiny app developed on our local workstation. After some interfacing with the Amazon web service environment, we'll push the Shiny app into the cloud, configure a web server, and end up with a secure (encrypted and authenticated) app running on a website with a custom domain name.

2 Methods

To begin, lets assume we've just finished developing a new Shiny app, named `power1_shiny`. We've done our development inside a working directory named `power1_app`. `power1_shiny` is a sub-directory inside `power1_app` containing the shiny program file `app.R`.

The methods described here apply generically to any Shiny app. See the `R/Shiny` code for our `power1_shiny` app (`app.R`) below.

Our shiny app is simple yet functional. It calculates the power for a two-sample t-test as a function of the standardized effect size. The app is intentionally minimal, using only base R functions, with a minimum of reactive widgets and layout commands.

```
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)
```

> **Motivation for this post:**
>
> We can test the app locally on our workstation by runnning it with the following command issued from the `power1_app` directory shell prompt
>
> ```
> zsh> R -e "library(shiny); runApp('power1_shiny/app.R', launch=T)"
> ```
>
> This will, in sequence, start the R program, load the Shiny package, run and launch the app in your default browser.

The margin figure shows our Shiny app running locally in a browser on our desktop. It consists of a widget to select the sample size and a plot to provide a dynamic visualization of the power as a function of the standardized effect size.

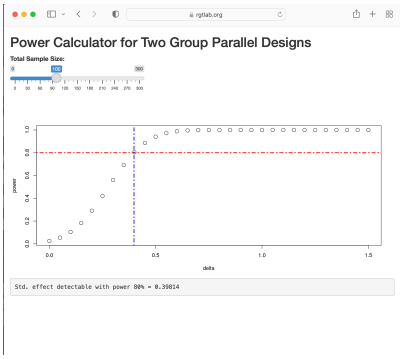

Figure 1: *Shiny app*

Once we determine our app is working as designed, we move on to the task of hosting the app on a (virtual) server with the goal of sharing it with our collaborators. How accomplish this? In the following we'll describe one approach that 'spins up' a server on Amazon Web Service EC2 and in just a few steps, through the application of Docker, R, Shiny, and Caddy yields a secure web app running on the web.

3 Hosting

In order to host `power1_shiny` online we'll need to complete the following tasks:

1. obtain a static IP address
2. obtain a domain name
3. set up a firewall
4. configure and launch a virtual server
5. install and configure a webserver
6. obtain and install an SSL certificate
7. setup an authentication method and
8. configure a reverse proxy method to 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.

3.1 Pre-launch tasks

Select a hosting service

There are a number of cloud based server options: Microsoft Azure, Oracle, Google Cloud, Amazon AWS EC2, Digital Ocean and Hetzner 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 AWS EC2 follows. (Detailed instructions for setting up a virtual server on EC2 both through the EC2 console and the command line interface were described in an earlier post: here and here.

Preface. create an AWS account or sign in and navigate to the EC2 dashboard.

step 1. Set up a working environment within EC2. That is:

a. generate secure shell (ssh) key-pair
b. configure a firewall.
c. obtain a static IP.
d. obtain a domain name.

Once the environment is set up

step 2. Config and launch the server

Configuration has two parts:

a. select an instance operating system (`ubuntu`) and
b. select an instance type (`t2-micro`)

Once the server is available connect via ssh.

```
ssh -i "~/.ssh/power1_app_ssh.pem"  ubuntu@rgtlab.org
```

or using the `config` setup described in Tip 1 at the end of this post.

```
ssh rgtlab.org
```

The only software tools necessary to install are Docker and Caddy. If you followed the CLI or console based instructions to set up a virtual server here or here Docker and Caddy will be pre-installed.

Otherwise you can install them with the following commands:

```
sudo apt update
sudo apt install docker.io -y
sudo apt install -y curl debian-keyring debian-archive-keyring apt-transport-https
curl -1sLf 'https://dl.cloudsmith.io/public/caddy/stable/gpg.key' | \
sudo gpg --dearmor -o /usr/share/keyrings/caddy-stable-archive-keyring.gpg
curl -1sLf 'https://dl.cloudsmith.io/public/caddy/stable/debian.deb.txt' | \
sudo tee /etc/apt/sources.list.d/caddy-stable.list
sudo apt update
sudo apt install caddy -y
```

At this point we have a customized virtual server with a static IP address, unique domain name and a firewall in place. In other words, items 1, 2, 3, and 4 from our 'hosting' list above are taken care of.

3.2 Post-Launch steps on local workstation

To run and host our Shiny app online we need to add a few configuration files to our `power1_app` development directory.

The first configuation file is:

3.3 Docker

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

We'll use docker to access R and Shiny. Here is our minimal dockerfile:

```
FROM rocker/shiny:4.2.0
# there are a bunch of files in /srv/shiny-server. dele
RUN rm -rf /srv/shiny-server
COPY /power1_shiny/* /srv/shiny-server/
# rocker/shiny adds a user named shiny
USER shiny
CMD ["/usr/bin/shiny-server"]
```

Photo by Ian Taylor on Unsplash

This file in just a few lines instructs Docker to build a new container based on a Rocker/Shiny image (which is a ubuntu image with R and Shiny installed) and layered with the addition of our Shiny code launch Shiny server listening on (default) port 3838.

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`.)

Note: We placed the `power1_shiny/app.R` code in the default location `/srv/shiny-server` so we only need to start the Shiny server and it will find the shiny program

To configure the web server we need to add a Caddy configuration file (default name `Caddyfile`) to the `power1_app` directory.

The Caddy configuration file specifies three critical things.

1. the site domain name.
2. the authentication pair login/hash-password, for each user and
3. the 'reverse proxy' map that redirects requests to port 443 (ssl port) onto port 3838 (Shiny port) in the docker container.

Our barebones Caddyfile looks like this:

```
rgtlab.org {
    basicauth * /power1_shiny/* {
        bob $2a$14$pYWd5O7JqNeGLS4m4CKkzemM2pq5ezn9bcTDowofZTl5wRVl8NTJm
    }
    root * /var/www/html
    handle_path /power1_shiny/* {
            reverse_proxy 0.0.0.0:3838
    }
    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
- the basicauth directive specifies login credentials for user bob (password: vanilla47)
- `handle_path` maps all https requests to port 3838 where Shiny is listening.
- `root` directive tells Caddy where to look for the `index.html` file.

Providing our servers domain name, `rgtlab.org` is sufficient to initiate an exchange with the `letsencrypt` service to generate an SSL certificate.

Lastly, we need an `index.html` file to provide a launch page for the app.

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

Once the config files, the index.html file and the Shiny code directory are in place copy we the entire `power1_app` directory to the server `rgtlab.org` with the secure copy command:

```
scp -i "~/.ssh/power1_app.pem" -r ~/prj/power1_app/  ubuntu@rgtlab.org:~
```

3.4 Post-Launch steps on remote server

Use ssh to login to the server and cd to `power1_app` directory

Build and run the Docker container (using the docker approach allows us to avoid installing both R and Shiny on the virtual server `rgtlab.org`).

```
docker build -t power1_image .
```

run container

```
docker run -d --name=power1_shiny -p 3838:3838 --restart=always power1_image
```

Next copy the `Caddyfile` to the location caddy expects to find it in the `/etc/caddy` directory

```
sudo cp ./Caddyfile /etc/caddy/
```

copy `index.html` to location Caddy expects to find it in the `/var/www/html` directory

```
cp  ./index.html /var/www/html/
```

Lastly, run the following command to restart Caddy

```
sudo systemctl reload caddy
```

The App launch page will now be available at `https://rgtlab.org`.

and you're good to go!

3.5 Tip construct ssh config file.

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

```
Host rgtlab.org
HostName 13.57.139.31 # static IP
StrictHostKeyChecking no  #avoid known host file error message
User ubuntu # default user on ubuntu server
Port 22  # the default port ssh uses
IdentityFile ~/.ssh/power1_app.pem
```

then you can ssh into the new server with

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