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

Ronald (Ryy) Glenn Thomas

Table of contents

Introduction

Photo by Nathan Waters on Unsplash

This is another 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. In this post we propose a simple method to migrate a shiny app from your local workstation to the web. The list of open-source technologies (software stack) we hesuggest for employment here is: linux, R, Shiny, Docker, and Caddy. Here we'll make use of the AWS cloud service. Future posts will describe alternate constructions, e.g. using the low cost cloud service: Hetzner.

This post provides a 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'll 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`. Our working directory is named `power1_app`. 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`) below.

Our shiny app is designed to be 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. The app is intentionally 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.

```
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 by runnning it with the following command issued from the `power1_app` directory.

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

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

Figure 1 shows the Shiny app running locally in a browser on our desktop, 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.

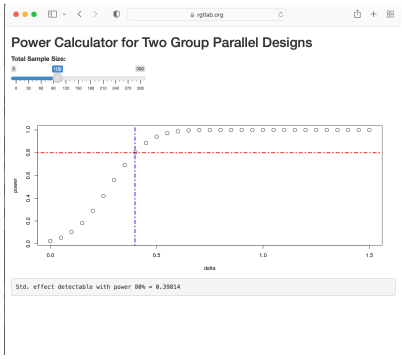

Figure 1: *Shiny app*

Once we determine our app is working as designed, we can move on to the task of hosting the app on a (virtual) server to share with our collaborators. There are many ways to accomplish this. Here we'll demonstrate a straightforward and efficient approach using a cloud service 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 secure web app to share with colleagues.

Hosting

In order to host `power1_shiny` 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
4. install and configure a webserver (tool to interact with https protocol requests)

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.

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 were described in an earlier post: https://focusonr.org/posts/setupaws/

0. Create an AWS account or sign in and navigate to the EC2 dashboard.
1. Set up an working environment with AWS server.
    a. define secure shell (ssh) key-pair (prefix pair with name `power1_app_ssh`.
    b. configure firewall.
    c. obtain static IP.
    d. obtain domain name (rgtlab.org).
    e. select instance operating system (`ubuntu`) and type (`t2-micro`)
    f. launch server

Once the server is available (optionally follow instructions in `Tip`) and connect via ssh.

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

The only necessary software to install is Docker and Caddy. Install them with the following commands:

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

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

Website

To configure the web server and containerize our app we need to add three files to the server, to go along with our Shiny app in the `power1_app` directory and move all files to the server.

The three configuation files are:

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

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

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

This file instructs Docker to build a container based on a Rocker/Shiny image (which is a ubuntu image with

Photo by Ian Taylor on Unsplash

R and Shiny installed) then copy into the container the `power1_shiny` directory containing the shiny code and finally launch Shiny server listening on (default) port 3838. We placed the `power1_shiny/app.R` code in the default location `/srv/shiny-server` so 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 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 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 generates an SSL certificate.

And the third file is the `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 three config files and the the Shiny code directory are in place copy the `power1_app` directory to the server `rgtlab.org` with the command:

```
scp -i "~/.ssh/power1_app_ssh.pem" -r ~/prj/power1_app/  ubuntu@rgtlab.org:~
```

Lastly, ssh to the server and cd to `power1_app` directory

copy `Caddyfile` to location caddy expects in `/etc/caddy` directory

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

copy `index.html` to location caddy expects in `/var/www/html` directory

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

and run the following command to

build the Docker container on `rgtlab.org`

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

create container and run

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

restart Caddy

```
sudo systemctl reload caddy
```

App launch page is now available at `https://rgtlab.org`.

and you're good to go!

Tip construct 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
```