Dockerize Shiny Apps

true

2022-08-29

Introduction

This is the first in a series of posts offering suggested strategies for leveraging open source technologies to provide straight-forward 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 group of technologies (or stack) we'll employ is: linux, R, Shiny, Caddy, git, and Docker. We'll make use of two cloud services github and AWS.

This initial post provides a minimal, proof-of-concept example of applying these technologies for the development and hosting of an interactive application for conducting a statistical power analysis.

We start with a simple, but hopefully useful, stand-alone shiny app and end with a secure (encrypted and authenticated) web site with a custom domain name hosting our app. The approach described here is minimalist.

Methods

To begin lets assume we're just finished developing a 'shiny' new shiny app, named `power0` .

The methods described here apply to any shiny app, but to provide a concrete example we've created our own app. See the code for our power0 shiny app here in appendix 1.

A screenshot of the finished product shows a shiny app with a widget to select the sample size and a visualization (2D plot) of the power as a function of the standardized effect size:

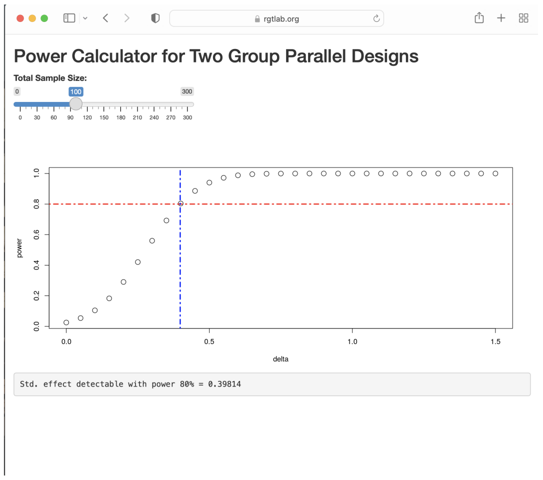
Consider an app that is a balance of simple and functional – one that calculates the power for a 2-sample t-test as a function of the standardized effect size. re is our shiny app `power0.R`:

The app is discussed further is appendix 1

```r
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)
```

Once we determine the app is working as designed we are interested in hosting the app on a (virtual) server to allow others to share in its utility. There are many ways to accomplish this. First off we'll demonstrate one of the most straightforward and efficient: spinning up a server on AWS and using docker containers for both our app and web server.



```
Std. effect detectable with power 80% = 0.39814
```

Hosting

Once we have a production ready shiny app available, we'll want to host it on the internet. To do that we'll need the following :

1. a virtual server (connected via ssh)
2. a static IP address (to identify the server online)
3. a domain name (human readable name for IP address)
4. a web server (software to recognize https protocol requests and respond)
5. an SSL certificate (encrypted communication)
6. an authentication method (password protection)
7. a reverse proxy method (translate port 443 requests to port 3838 for shiny)
8. a containerized version of app. (bundled app and requisite software)

9. a method for power0 container to communicate with containerized web-server.

At first glance these 9 requirements can appear daunting, but on closer inspection all can be met with minimal expertise and minimal or no cost.

This can all be done at no cost if you have your own (self-hosted) server with IP address, and domain name, or at 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.

There are a number of cloud based server options: Microsoft Azure, Google Cloud, Amazon AWS, 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.

Specific instructions for AWS EC2 are here in appendix 2.

Once the server is available connect via ssh, login, and Install docker, docker-compose and git. with the following command:

```
sudo apt install -y git docker docker-compose
```

Once the hosting process is complete items 1, 2, and 3 will be taken care of.

Web-site

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 `power0` directory (in the home directory for default user `ubuntu`).

The easiest way to do this is to add the three files to the power0 directory on our workstation and then "push" a copy to github.

These three configuation files are:

- a Caddyfile
- a Dockerfile
- a docker-compose.yml file

Lets discuss each and initially add them to the power0 directory on our workstation. Once the three files are in place in the directory we can push copies to github and from there we can access them from our server.

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, which is required to use the encrypted communication protocol https.

Caddy is configured with a file named `Caddyfile`. The caddyfile configures three things.

1. the site domain name.
2. the authentication pair login/hash-password, and
3. the 'reverse proxy' map that redirects requests to port 443 (ssl port) to port 3838 (shiny port).

The Caddyfile in our example is:

auth info bob/utter

```
rgtlab.org {
#auth credentials: bob/utter
basicauth * {
        bob JDJhJDE0JElCQmRGaTA0ajY3bkZTLjRiWUZ4enVoZnVSQzVXVGVUMHlVcXJTaTRGYmpRQVFHLnYzN0tx
    }
    handle_path /power0/* {
        reverse_proxy power0:3838
    }
}
```

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

For your project replace:

- rgtlab.org with your domain name
- Bob JDJ… with user names and hash-passwords
- power0 with you app's name

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

The second file is the dockerfile. In its simplest form it instructs Docker to build a container based on a Rocker/Shiny image then copy in the app.R code and launch shiny on (default) port 3838.

Photo by Ian Taylor on Unsplash

Here is our dockerfile:

```
FROM rocker/shiny:4.2.0
COPY app.R /srv/shiny-server/
USER shiny
CMD ["/usr/bin/shiny-server"]
```

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

The docker-compose.yml file:

```
version: "3.7"

services:
  power0:
    build: .
  caddy:
    image: caddy:2.3.0-alpine
    ports:
      - "443:443"
    volumes:
      - $PWD/Caddyfile:/etc/caddy/Caddyfile
      - caddy_data:/data
volumes:
    caddy_data:
```

To generate a password for say foobar.

```
docker-compose run caddy caddy hash-password foobar
```

Once in place push power0 to github and then clone repo to server.

Lets clone them into the server from github.

```
git clone https:// ???
```

cd to power0 directory and run

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

and you're good to go! The app `power0` can be accessed by 'bob' at the url

```
https://rgtlab.org/power0
```

with password 'utter'

Appendix1

Consider the power0.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)
```

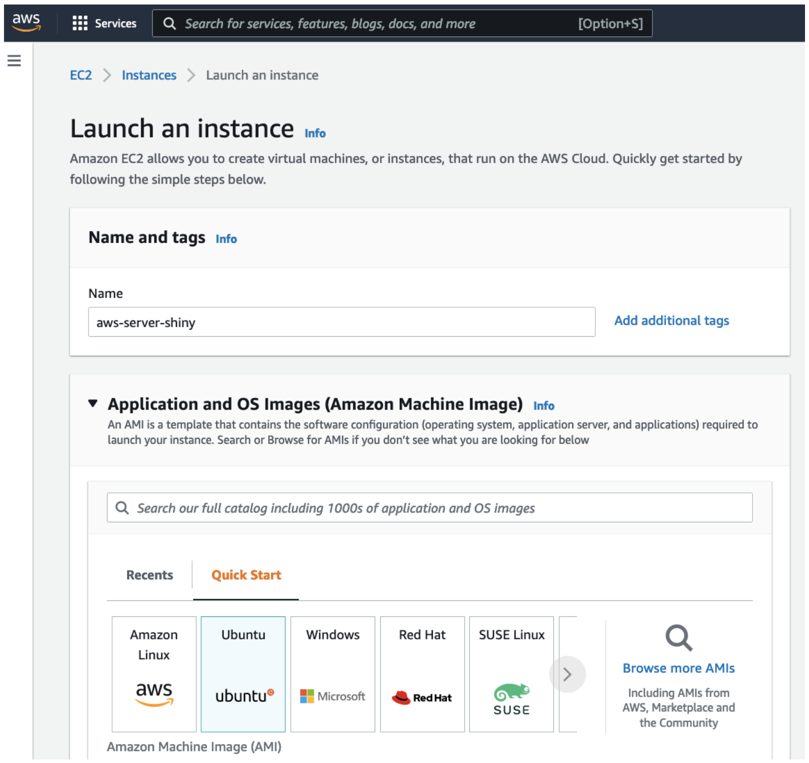
The app is designed to be maximally minimal. Using only base R functions, with a minimum of reactive widgets and layout commands to keep it simple while still performing a useful function.

Appendix2

AWS is a reasonable choice for setting up a small custom server. AWS offers a free set of servers for the first 12 months.

To start open the EC2 console.

```
   https://aws.amazon.com/console
```


EC2 > Instances > Launch an instance

Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags Info

Name: aws-server-shiny

Add additional tags

Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search for services, features, blogs, docs, and more
Search our full catalog including 1000s of application and OS images
Recents
Quick Start
Amazon Linux
Ubuntu
Windows
Red Hat
SUSE Linux
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Amazon Machine Image (AMI)

Create an account or sign in. Next set up a working environment. Specifically you'll want to set up four components of the environment:

1. Ssh key pair
2. Firewall
3. Static IP
4. Domain Name

The first time you create an AWS account you need to exchange an SSH key pair with AWS. You can generate the ssh key pair locally on you workstation and upload the public key to EC2.

Create a directory to hold the keys. e.g. `~/.ssh`. Change directory into `.ssh`. Generate the keys with the command

```
ssh-keygen -m PEM
```

name the key prefix something like `ssh-rsa`.

On EC2 select `security/keys`, browse to the `.ssh` directory and import the public key `ssh-rsa.pub`.

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

```
Host ec2
HostName 13.57.139.31 # static IP
User ubuntu # default user on ubuntu server
Port 22  # the default port ssh uses
IdentityFile ~/.ssh/ssh-rsa
```

then you can ssh into the new server with

```
sh> ssh ec2
```

Use "elastic IP" to get a static IP that can be assigned to the server

- click on elastic IP in left panel

- select associate Elastic IP 13.57.139.31 choose an instance (shiny-july22) to associate with.
- side panel, click "ec2"
- side panel, click "Instances"
- from top bar, click "Launch Instances"

2. From "Quick Start" click Ubuntu button.

- Name the server, say shiny-july22
- Choose an AMI (instance template, operating system):

Suggest choose "Ubuntu Server 22.04 LTS", but other linux distributions can be utilized, e.g.u Red Hat, or SUSE.)

3. Next choose an instance **type**, e.g. "t2-micro". (different instance types are mixtures of size, processors, memory, instance storage, network performance) click "Next: Configure Instance Details"

4. choose Key pair (use in place aws18.pem) or set up new pair

5. Add security group, e.g. 'shiny' (sg-0f37c94ac1e1b6250) allowing ports 80 (http), 22 (ssh), 443 (https), and 3838 (shiny). and 8787

6. choose 30 GB of EBS General Purpose (SSD) or Magnetic storage

7. click Launch Instance

Log into new instance with ssh from local

```
ssh -i ~/.ssh/aws18.pem ubuntu@13.57.139.31
```

or

```
ssh ec2
```

if you've set up a `config` file in `~/.ssh`

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

Go to godaddy or Amazon route 53 to associate a domain name with the Elastic IP in EC2.

(screenshot)

To associate domain name rgtlab.org with elastic IP.

in Route 53:

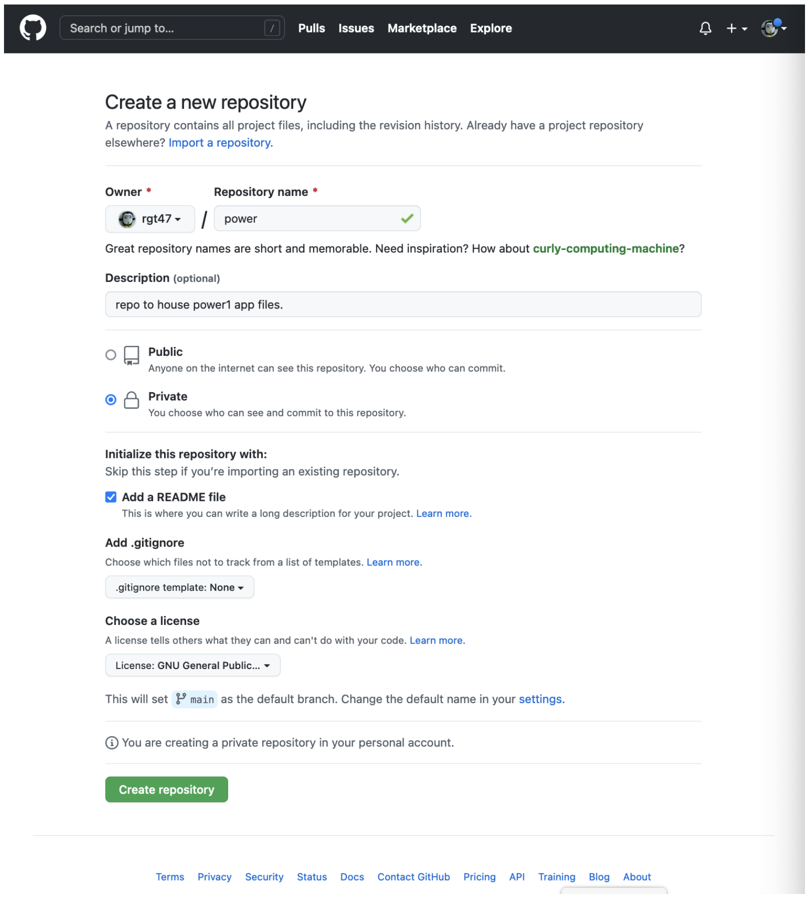
- click on 'hosted zones' in side panel
- click on rgtlab.org in center panel
- click on checkbox for rgtlab.org type=A line
- then click on edit record in right panel
- change ip address to 13.57.139.31

appendix x (joe data version)

ok! got my shiny app running. Works great! Now how do I get it up on the web and shared with my client Bob?

Start by creating a repo for the app on github.

- login to github (screenshot)



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * rgt47 / Repository name * power

Great repository names are short and memorable. Need inspiration? How about curly-computing-machine?

Description (optional): repo to house power1 app files.

Public — Anyone on the internet can see this repository. You choose who can commit.

[✓ Private] — You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

[✓ Add a README file] — This is where you can write a long description for your project. Learn more.

Add .gitignore — Choose which files not to track from a list of templates. Learn more.
.gitignore template: None

Choose a license — A license tells others what they can and can't do with your code. Learn more.
License: GNU General Public...

This will set `main` as the default branch. Change the default name in your settings.

You are creating a private repository in your personal account.

- click on `new` in `repo name` and enter `power1`. (Make the repo private we only want to share with Bob at this point).
- create repo
- on workstation: clone the repo

```
git clone https://github.com/rgt47/power1.git
```

- copy `power1.R` in `power1` repo directory.
- update remote repo
  - git add .
  - git commit -m 'add shiny code'
  - git push

Now we need a (virtual) server to host the app. Let use Amazon AWS.