Dockerize Shiny Apps

true

2022-07-09

Introduction

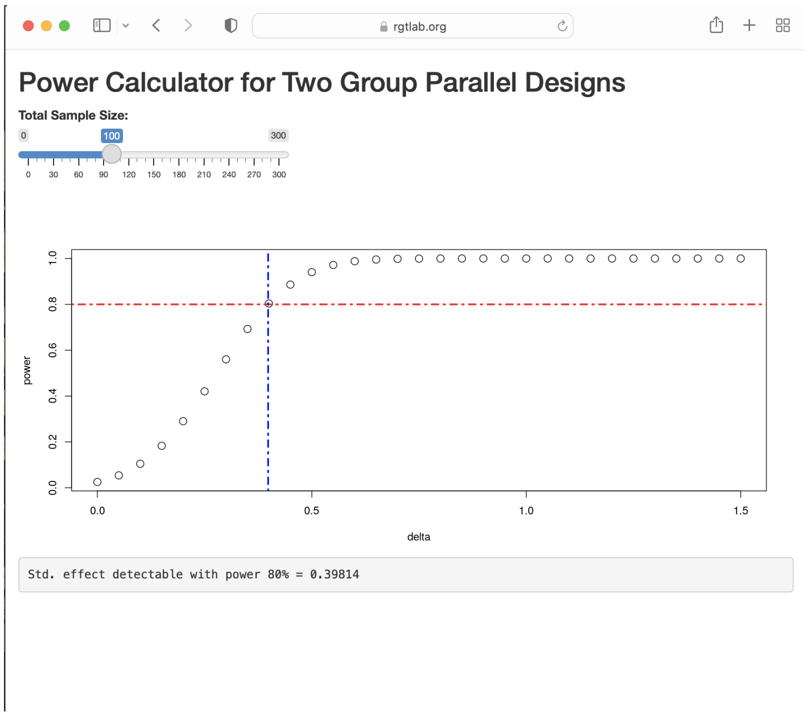
This is the first in a series of posts on how to leverage the technologies of: R (statistics environment), Shiny (dynamic programming), Docker (container packaging), Linux, Caddy (webserver), and AWS (Amazon's cloud hosting service) to provide open-source solutions to one of the central challenges in the practice of data science i.e. how to communicate analysis results to clients and collaborators.

This initial post provides a minimal example of applying these technologies to develop and host an interactive tool for conducting a statistical power analysis.

We start with a simple, but still useful, shiny app and end with a hosted web site with a custom domain name. Lets assume we're just finished developing a 'shiny' new shiny app, named `power0`,

See the details of the power0 shiny app here.

A screenshot of the finished app shows a shiny app with a widget to vary the sample size and a visualization of the power as a function of the standardized effect size:



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

We are interested in hosting the app on a (virtual) server to allow others to share in its wonderfulness. There are many ways to accomplish this. First off we'll demonstrate one of the most straightforward: a docker based approach.

The first step is to dockerize (or more specifically a docker-compose) our app. This provides a whole host of advantages. Lets discuss.

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"]
```

Thats it. With the dockerfile and the app.R fine in the same directory we can first build the docker image, and then push it to Docker hub as a convenient repository. (Alternatively we could copy it directly to the server or to another site, e.g github.com.)

Details are here # Methods

Hosting

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

1. a virtual server with a static IP address,
2. a domain name
3. an SSL certificate (encrypted communication)
4. an authentication method (password protection)
5. a reverse proxy method
6. an index file (landing page)

This can all be done at no cost if you have your own (self-hosted) server 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.

custom EC2 instance

See the details of setting up the server are here

- Open EC2 console.

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

or

```
https://us-west-1.console.aws.amazon.com/ec2/v2/home?region=us-west-1#
```

- log in
- 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. Red Hat, or SUSE.)

e.g. ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20220609

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

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

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

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

click Launch Instance

Use elastic IP (to allow server to be run “on-demand”) * click on elastic IP in left panel * select associate Elastic IP 13.57.139.31 choose an instance (shiny-july22) to associate with.

Log into new instance with ssh from local

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

or

```
ssh ec2
```

One piece of software to install on the server: Docker.

add ubuntu to the docker group.

get domain name from Route 53

3) Install Docker on server. sudo apt update sudo docker run -d -p 80:80 –name test4 -p 443:443 -v ~/mywebsite:/usr/share/caddy/ -v ~/Caddyfile:/etc/caddy/Caddyfile -v caddy_data:/data caddy:2

to debug

sudo docker logs test4

cat Caddyfile (bob/utter credentials)

```
rgtlab.org {
    basicauth * {
        bob JDJhJDE0JEw4Zk5MQi5nYndEaHRjVTFVeGl2a2VBZTVPcFI0THhEZUd0OEJkOTR2d0VjRUxsMDBsQUg2
    }
    respond "It works!"
    root * /var/www/html
}
```

4)
5)

~~~
```sh

view app at
http://ec2-13.57.139.31.us-west-1.compute.amazonaws.com:3838/zenn/hello/
~~~

Security Set-up

1. Get domain name, e.g. from Godaddy.com. Mine is inferencex.com
2. get an elastic IP, reserved for your personal use. Needed for security certificate use, e.g. my elastic IP is 13.57.139.31

Tips

Build and run instance (call it four) open in shell for exploration

```
docker build -t four .
docker run -it --rm four /bin/bash
```

Construct a config file in ~/.ssh as:

```
Host ec2
    HostName 13.57.139.31
    User ubuntu
```

```
    Port 22
    IdentityFile ~/.ssh/aws18.pem
```

then you ssh into new server with

```
sh> ssh ec2
```

To construct the web site we need three files in the home directory for default user ubuntu. The index.html file provides the 'landing page' for the site.

We'll use Caddy as our web server.

The second file is the Caddyfile configuration file. The caddyfile provides 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 and docker-compose.yml file are

Caddyfile

auth info bob/utter

```
rgtlab.org {
    basicauth * {
        bob JDJhJDE0JEw4Zk5MQi5nYndEaHRjVTFVeGl2a2VBZTVPcFI0THhEZUd0OEJkOTR2d0VjRUxsMDBsQUg2
    }
    root * /srv
    handle_path /power0/* {
        reverse_proxy power0:3838
    }
    file_server
}
```

And the third file is the docker compose file that pulls our shiny app, and the caddy server and create a local network for them to communicate in.

docker-compose.yml

```
version: "3.7"

services:
  power0:
    image: rgt47/power0:latest
  caddy:
    image: caddy:2.3.0-alpine
    ports:
      - "443:443"
    volumes:
      - $PWD/Caddyfile:/etc/caddy/Caddyfile
      - $PWD/site:/srv
      - caddy_data:/data
      - caddy_config:/config
volumes:
  caddy_data:
  caddy_config:
```

The index.html file

```html
<!DOCTYPE html>
<html>
  <body>
      <a href="./power0/">Power0</a>
  </body>
</html>
```

to add authentication to the web site use basic auth supplied by caddy first to get access to the command line interface (CLI) for caddy issue the docker command to load a caddy container from Docker Hub

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



docker run -it --rm caddy:2.3.0-alpine /bin/sh
caddy hash-password --plaintext bar
```

the