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

gitlab, Docker-compose, EC2 version

Ronald (Ryy) Glenn Thomas

11/9/23

Table of contents

Photo by Nathan Waters

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 strategy is based on a set of open source tools. The list of technologies (software stack) we suggest for employment is: linux, R, Shiny, Docker, docker-compose, Git, and Caddy. In this post we'll make use of two cloud services: Gitlab and Amazon Web Service (AWS). Further posts will describe alternate services, e.g. using the low cost cloud service: Hetzner.

Also described in other posts, even more straightforward strategies that avoid Gitlab and docker-compose. These alternative approaches 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 app running securely 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 gitlab and then `clone` it to your local workstation. In other words, log into gitlab

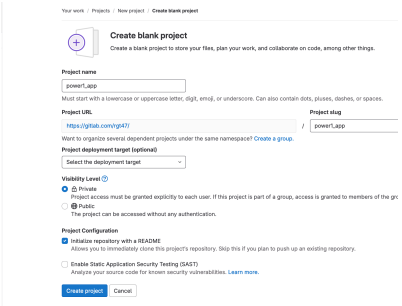
and create a new empty repo, call it, e.g.`power1_app`; While in gitlab you should also create a second, public repo, call it images. We'll use this repo to store screenshots of the app. Then on your local workstation navigate to your Shiny development directory, say `~/prj`, and clone the `power1_app` and `images` repos from gitlab:

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, respectively.

Details for creating a gitlab repo follow:

- login to `gitlab` (screenshot)
- click on `New project`. Then in `repository name` field enter `power1_app`.
- make the repo private, we only want to share with our collaborator at this point).
- leave `deployment target` empty.
- create the repo. Click `Create project` blue button at the bottom of the page.
- on your laptop cd to development repo, say ~/prj and clone the gitlab repo:

```
cd ~/prj
git clone https://gitlab.com/rgt47/power1_app.git
cd power1_app
git clone https://gitlab.com/rgt47/images.git
mkdir power1_shiny
mkdir site
```



Lets jump ahead to the point where you've just finished developing a new Shiny app named `app.R`, in the `power1_shiny` directory. (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`).

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.

The app is 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.

The code is here:

```
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, `power1_app`, by runnning 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 our default browser.

Figure 1 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 statistical power as a function of the standardized effect size:

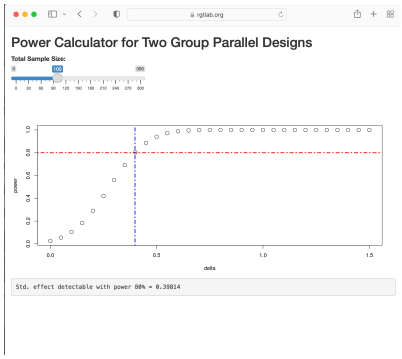


Figure 1: *Shiny app*

After determining app is working as designed, the next step is to set up a secure host on the web. Once the app is hosted we simply need to send a link and security credentials to our collaborators for them to have secure access to the Shiny app.

How to set up the hosting server? 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. In other words, we'll describe how to 'spin' up a virtual 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 our colleagues.

2.1 Hosting

Figure 2 illustrates the tools we'll use and the flow of program and configuration files. In order to host `power1_app` online we'll need to complete the following tasks:

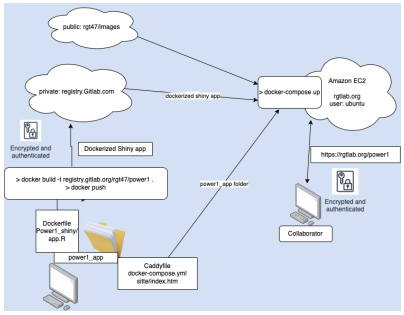


Figure 2: *Data flow*

1. Generate a virtual server (connected via ssh) with a firewall on EC2, either via the interactive `EC2` console of by the `AWS CLI` command line tools.
2. obtain a static IPv4 address (to identify the server online)
3. obtain a custom domain name (name to associate with static IP address) from Route 53 or godaddy.com or another domain registration provider.
4. install and configure a webserver (tool to interact with https protocol requests)
5. configure authentication for web site
6. obtain and install an TLS (transport layer security) security certificate (to allow encrypted communication between the server and other machines on the network)
7. configure a reverse proxy method (translate https, port 443, requests to Shiny, port 3838 requests)

At first glance these 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 using AWS EC2 follows. Detailed instructions for setting up a server on EC2, both via the console and the command line interface are covered in an earlier post here and here.

Step 0. Create an account or sign in. Step 1. Set up an interactive environment with AWS server. This entails: a. define ssh key-pair. b. configure firewall. c. obtain static IP. d. obtain domain name. e. select an instance (AMI, type and disk size), generate and launch server.

On;e the server is available connect via ssh, and login,

The only software necessary 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, with 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.

"What is a reverse proxy? A reverse proxy is a server, app, or cloud service that sits in f"ront of one or more web servers to intercept and inspect incoming client requests before forwarding them to the web server and subsequently returning the server's response to the client.

*An **SSL/TLS certificate** is a digital object that allows systems to verify the identity & subsequently establish an encrypted network connection to another system using the Secure Sockets Layer/Transport Layer Security (SSL/TLS) protocol. Certificates are used to create a public key infrastructure (PKI). PKI provides a way for one party to establish the identity of another party using certificates if they both trust a third-party - known as a certificate authority. SSL/TLS certificates thus act as digital identity cards to secure network communications, establish the identity of websites over the Internet as well as resources on private networks."*

Ref: https://www.ssltrust.com/resources/security-terms-glossary/what-is-reverse-proxy

<a href="https://aws.amazon.com/what-is/ssl-certificate/"

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 `Gitlab` as an intermediate repository between or workstation and the server in a slightly indirect route to create and place the necessary files on the server but this approach will allow us to do all our continuing 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 to `Gitlab` and from there they can be accessed from our server.

These three configuation files are:

3 Docker

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:

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

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 app.R code in the default location `/srv/shiny-server` we only need to start the server and it will find the shiny program.

Then build and push the image to the `gitlab` container registry.

Photo by Ian Taylor on Unsplash
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

```
# login to gitlab

cat gitlab_access_token | docker login registry.gitlab.com -u rgt47 --password-stdin

docker build -t registry.gitlab.com/rgt47/power1_app/power1_image:v1.0 \
        --platform linux/x86_64 .
docker push registry.gitlab.com/rgt47/power1_app/power1_image:v1.0
```

4 Caddy

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 acquisition and installing of an SSL certificate. An SSL cert is required by most browsers to use the encrypted communication protocol https.

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 all users (e.g. bob/vanilla47):

Our barebones `Caddyfile` looks like this:

```
# use caddy auth tool to generate a password via the `bcrypt` algorithm.
# > caddy hash-password --plaintext hiccup

rgtlab.org {
basicauth /power1/* {
    Bob $2a$14$Zkx19XLiW6VYouLHR5NmfOFU0z2GTNmpkT/5qqR7hx4IjWJPDhjvG
}
    root * /srv
    handle_path /power1/* {
        reverse_proxy power1:3838
    }
```

```
    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.

5 Docker Compose

And a third file is a config file for Docker Compose. Docker Compose is a Docker module that provides a framework for running multi-container applications. This docker compose YAML file instructs Docker to containerize our Shiny app, pull a caddy webserver image from Docker Hub and create a local network for the two containers to communicate in.

A Docker-compose configuration file (default name `docker-compose.yml`).

The `docker-compose.yml` file:

```
version: "3.7"

services:
  power1:
    image: registry.gitlab.com/rgt47/power1_app/power1_image:v1.0
    restart: unless-stopped
    expose:
      - "3838"
  caddy:
    image: caddy:2.6.4-alpine
    restart: always
    ports:
```

```
      - "443:443"
    volumes:
      - $PWD/Caddyfile:/etc/caddy/Caddyfile
      - $PWD/site:/srv
      - caddy_data:/data
      - caddy_config:/config
    depends_on:
      - power1
    environment:
      - HOST="rgtlab.org"
      - EMAIL="rgthomas@ucsd.edu"
volumes:
  caddy_data:
  caddy_config:
```

6 Landing Page

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

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Power Calculators</title>
    <link rel="stylesheet" href="https://unpkg.com/bulma@0.9.0/css/bulma.min.css" />
  </head>
  <body>
    <div id="app">
      <section class="hero is-small">
        <div class="hero-body">
          <div class="container has-text-centered">
            <h1 class="title">RGT Lab Power Calculators</h1>
          </div>
        </div>
      </section>
```

```
            <hr>

            <div class="columns">
              <div class="column is-4 is-offset-1">
      <img src="https://github.com/rgt47/power0/blob/master/power1.png?raw=true"
        width="200" height="250"  "float: left; padding: 3px 3px 0px 3px;" >
              </div>
              <div class="column is-6">
                <h1 class="title"> Power1 App </h1>
                <p> Power for two-sample t-test </p>
                <br>
                <a href="./power1/" class="button is-info">Go to app</a>
              </div>
            </div>

    </div>
  </body>
</html>
```

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

```
.
  Caddyfile
  Dockerfile
  docker-compose.yml
  site
      index.html
```

7 Gitlab

Push the new content to gitlab.

```
git push
```

Login to `gitlab` and issue a personal access token. Save it as `gitlab_access_token` in the `power1_app` directory.

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

```
ssh rgtlab.org
cat gitlab_access_token | \
  docker login registry.gitlab.com -u rgt47 --password-stdin
git clone https://gitlab.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/
```

8 Appendices

8.1 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
```

8.2 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}
```

8.3 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 ~/.ssh/power1_app.pem
```

then you can ssh into the new server with

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

9 References

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

10 APPENDIX

```
> cd /Users/zenn/Dropbox/prj/c060
> aws_create_keypair.sh power1_app
> aws_create_security_group.sh -s power1_app -h on -k on -i on -j on
> security_grp="sg-0679282d70f727f1e"
> echo $security_grp
If we need a new IP address:
> static_ip=`aws ec2 allocate-address | jq -r '.PublicIp'`
> echo $static_ip
> aws_create_instance.sh -p power1_app
instance_id="i-0ecb046262ab6efca"
 aws ec2 associate-address --instance-id $instance_id --public-ip $static_ip
```

```
associate IP with domain name in Route 53
change the IP address in ~/.ssh/config

scp -i "~/.ssh/power1_app.pem" .zshrc ubuntu@rgtlab.org:~
scp -i "~/.ssh/power1_app.pem" .vimrc ubuntu@rgtlab.org:~

> ssh rgtlab.org

Check the log of your user data script in:
    /var/log/cloud-init-output.log

git clone https://gitlab.com/rgt47/power1_app.git
git clone https://gitlab.com/rgt47/images.git

 docker run -d \
    --name=hello \
    --restart=always \
    -p 9000:3838 \
    registry.gitlab.com/analythium/shinyproxy-hello/hello:latest


 docker run -d \
    --name=power1 \
    --restart=always \
    -p 9001:3838 \
    registry.gitlab.com/rgt47/power1_app/power1_image:v1.0

http://rgtlab.org:9001/
displays power1_shiny app

git clone https://github.com/analythium/docker-compose-shiny-example.git
cd docker-compose-shiny-example.git

edit the docker-compose-prod.yml file to include domain name and email

docker compose \
  -f docker-compose.yml \
  -f docker-compose-prod.yml \
  up -d
```

```
change docker-compose.yml to point at gitlab repo rgt47/power1_app for 'hello' app

works.

now cp  docker-compose-shiny-example.git
to 'master' and begin to transform 'master' into 'power1_app'
keep logging info

to rebuild all the containers in the docker-compose.yml file:

docker compose up --build -d

remove all containers and images

docker system prune -a

if changes are made to shiny app on m1 mac. stop the compose network, pull updated container

cd power1_shiny
vim app.R
cd ~/Dropbox/prj/c060/docker_simple_power1_app
docker build -t registry.gitlab.com/rgt47/power1_app/power1_image:v1.0 --platform linux/x86_
docker push registry.gitlab.com/rgt47/power1_app/power1_image:v1.0

docker compose stop
docker compose rm -f
docker compose pull
docker compose up -d


~/master-repo (main )    cat Caddyfile
rgtlab.org {
basicauth /power1/* {
    Bob $2a$14$Zkx19XLiW6VYouLHR5NmfOFU0z2GTNmpkT/5qqR7hx4IjWJPDhjvG
}

    root * /srv
    handle_path /power1/* {
        reverse_proxy power1:3838
    }
```

```
    file_server
}


version: "3.7"

services:
  power1:
    image: registry.gitlab.com/rgt47/power1_app/power1_image:v1.0
    restart: unless-stopped
    expose:
      - "3838"
  caddy:
    image: caddy:2.6.4-alpine
    restart: always
    ports:
      - "443:443"
    volumes:
      - $PWD/Caddyfile:/etc/caddy/Caddyfile
      - $PWD/site:/srv
      - caddy_data:/data
      - caddy_config:/config
    depends_on:
      - power1
    environment:
      - HOST="rgtlab.org"
      - EMAIL="rgthomas@ucsd.edu"
volumes:
  caddy_data:
  caddy_config:

~/master-repo (main  )    cat site/index.html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Power Calculators</title>
```

```
    <link rel="stylesheet" href="https://unpkg.com/bulma@0.9.0/css/bulma.min.css" />
  </head>
  <body>
    <div id="app">
      <section class="hero is-small">
        <div class="hero-body">
          <div class="container has-text-centered">
            <h1 class="title">RGT Lab Power Calculators</h1>
          </div>
        </div>
      </section>
            <hr>

            <div class="columns">
              <div class="column is-4 is-offset-1">
      <img src="https://github.com/rgt47/power0/blob/master/power1.png?raw=true"
        width="200" height="250"  "float: left; padding: 3px 3px 0px 3px;" >
              </div>
              <div class="column is-6">
                <h1 class="title"> Power1 App </h1>
                <p> Power for two-sample t-test </p>
                <br>
                <a href="./power1/" class="button is-info">Go to app</a>
              </div>
            </div>


    </div>
  </body>
</html>
```