Set up a virtual server on AWS (in anticipation of hosting Shiny apps)

Ronald (Ryy) Glenn Thomas

6/11/23

Table of contents

Photo by Nathan Waters on Unsplash

1 Introduction

> **i** Motivation for this post:
>
> Ok! I've got my Shiny app running just the way I want it. Works great! Now, how do I get it up on the web and shared with my collaborators?

Assuming we have a working shiny app, we often next need to address the task of how to host the app on the web to share with our collaborators. To be sure, there are many ways to accomplish this. Below we describe a recommended process we've found to be straightforward. In this post, we'll describe how to 'spin up' a server on Amazon Web Service EC2. In a future post we'll show how, in just a few steps, through the application of Docker, R, Shiny, and Caddy (webserver) functionality we can have a fully functional and secure app available on the web.

2 Hosting

No matter what procedure we use In order to host a shiny app, (for example one named `power1_shiny`), online we'll need to complete the following set of tasks:

1. obtain a static IP address
2. obtain a domain name (e.g. `rgtlab.org`) and associate it with the IP address
3. configure (select number of CPUs, amount of memory, OS, etc) and launch a virtual server

4. define a security model, aka a firewall for the server
5. associate the IP, domain name and firewater with the server
6. install and configure a webserver
7. obtain and install an SSL certificate (to allow encrypted communication)
8. setup an authentication method (password protection for app access)

9. configure a reverse proxy method i.e. translate https (port 443) requests to Shiny (port 3838). This avoids the need for URLs like `https://rgtlab.org:3838/power1_shiny`

> **Not to worry:**
>
> At first glance these requirements can appear daunting, but on closer inspection all can be met with relative ease and minimal cost (e.g. using a 'free-tier' server on 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, from one of the less popular domains i.e. other than '.com' or '.org') or at no cost if we have your own server with IP address, and domain name.

> **Actually ...:**
>
> Technically, if the goal is simply to get the app up on the web, its not required to make a static IP or a domain name or a firewall or an authentication method or an SSL certificate, or even a reverse proxy. but if these elements of the process are skipped the server will only be able to communicate via the unencrypted HTTP protocol and the site URL will be something like 111.222.333.444:3838/power1_shiny, and anyone with the URL will be able to reach the site.

2.1 Select a hosting service

There are a number of cloud based server options to choose from: Microsoft Azure, Oracle, Google Cloud, Amazon AWS EC2, Digital Ocean or 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.

In this post we'll provide a step-by-step description of a process using Amazon Web Services Elastic Compute Cloud (AWS EC2) infrastructure.

AWS is a reasonable choice for setting up a small custom server. Its not the cheapest option, but the system is well documented and, in our experience, reliable.

To start, open the EC2 console by visiting the URL:

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

(see margin figure)

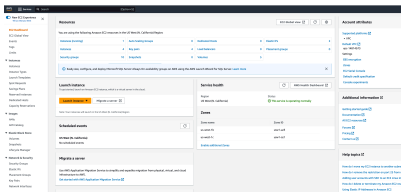


Figure 1: AWS console

In the console window choose regional service. For me its "N. California".

Next create an account, or sign in, and once you're logged in navigate to the EC2 dashboard. Its through this dashboard (or console) that we'll define the parameters for the type of server to launch and the mechanisms for communicating with it.

2.2 AWS Working Environment

Along with selecting a server we need to set up a working environment. We recommend setting up the working environment before launching the server, as it saves some back and forth with the console, but the order is not critical. The working environment consists of four main components:

1. A secure shell (ssh) key-pair to allow remote and secure login to the virtual server once its launched.
2. A firewall or security model which will restrict server access to only secure connections. The firewall, by default, closes off all incoming traffic except through those ports specifically named.

3. A static IP address. This is required for maintaining the link between the domain name and the server when rebooting. (The default is for the instance/server to be assigned a new IP address each time its rebooted). and
4. A domain name, say `rgtlab.org`. A domain name is not required but will facilitate collaborator access by not needing to use the IP address directly.

2.2.1 Ssh key pair

In order to securely communicate with the server we need to exchange an ssh key pair with AWS. The pair consists of a **private** and a **public** key. We can generate an ssh key pair in one of two ways in EC2. Either, locally, on our workstation and upload the public key to EC2, or have EC2 generate the key pair and download the private key.

For the first option we create a directory on our workstation to hold the keys and navigate to it, e.g. `~/.ssh`. In the `~/.ssh` directory generate the keys with the command

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

"PEM" defines the key format. More information on public key authentication can be found here. In the interactive dialog that follows name the key prefix something like `power1_app`. The dialog will ask for a passphrase. Enter one for an additional level of security, but its not required. The `ssh-keygen` program will generate two files: `power1_app.pem` and `power1_app.pub`

To complete the process return to the EC2 dashboard and select `Actions` and then `Import key pair` in the left panel. Enter the name `power1_app` and select the `Browse` button. Navigate to the file `power1_app.pub` in the directory `~/.ssh` and and select the `Import key pair` button at the bottom of the page.

For the second approach select `Create key pair` button in the upper right of the console page.

A form will appears and ask for a name. Enter something like `power1_app`. Select `RSA` for key pair type and `.pem` for key file format.

Give the pair a name, say `power1_app`, and the keys will be created and the private key `power1_app.pem` will be downloaded to our local machine to the `~/.ssh` directory. Lastly, change the access permissions for the private key with the following command:

```
sudo chmod 600 power1_app.pem
```

2.2.2 Firewall

To create a firewall click on **Network settings** in the left hand panel. Choose **Create security group** and select **Allow SSH traffic** and **Allow HTTPS traffic**. This will create a firewall that leaves open only ports 22 and 443, for `ssh` and `https` incoming traffic respectively. Name the security group something like `power1_firewall`.

2.2.3 Static IP address

The next step is to use the `elastic IP` service to get a static IP that can be assigned to the server. Navigate to **Network and Security** again and select **Allocate Elastic IP**. An IP will be assigned from the EC2 pool of available IPv4 IP addresses e.g. 13.57.139.31.

2.2.4 Domain Name

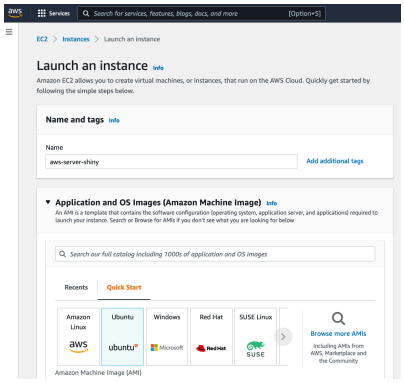
To obtain a dedicated domain name go to Amazon route 53 to select a domain name and associate it with our static IP.

Once a domain name is acquired, e.g. `rgtlab.org`, associate it with our static IP address. This can be done via the `Route 53` service. To associate domain name `rgtlab.org` with the elastic IP do the following in Route 53:

- click on `hosted zones` in the 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 the assigned static IP (e.g. 13.57.139.31).

2.3 Select and launch instance

2. From `Quick Start` in the EC2 dashboard select an operating system for the server. We recommend the `Ubuntu` OS. Ubuntu is a mature Linux distribution based on Debain Linux. Click the `Ubuntu` button. (see margin figure)



- Name the server, say `power1`

3. Next choose an instance **type**, e.g. `t2-micro`. Different instance types are combinations of, number of processors, memory, storage capacity, and network performance.
4. click **Configure Instance Details**
5. choose a Key pair (select `power1_app` from your environment) .
6. Add security group, use `power1_firewall` from your environment.
7. choose 30 GB of `EBS General Purpose (SSD) or Magnetic storage`. Thirty GBs is the maximum allowed in the 'Free tier' of servers on AWS. In our experience smaller disk sizes can lead to problems.
8. click **Launch Instance**

to launch the server.

3 Access server

On your laptop log into server with

```
ssh -i "~/.ssh/power1_app.pem" ubuntu@rgtlab.org
```

4 Appendix: Tip 1

> Tip 1.
>
> 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 we can ssh into the new server with

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

5 Appendix: succinct instructions

Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/

From the top navigation bar, select a Region to create an instance in. For me its "N. California".

Create an account or sign in and navigate to the EC2 dashboard.

In the left side panel select **Key Pairs** (under Network and Security).

At the top right select the **Create key pair** button. A **Key Pair** form will open.

Give the key pair a name. Something like `power1_app`. Select a key pair type, suggest `RSA`. Select a **Private key file format**, suggest `.pem`

Below the form select the **Create key pair** button. A pair of keys will be created and the private key `power1_app.pem` will be downloaded to we local machine. In my case to the default `~/Downloads` directory.

Move the file to the `~/.ssh` directory: `mv ~/Downloads/power1_app.pem ~/.ssh`

Change the access permissions: `sudo chmod 600 power1ssh.pem` to be more restrictive.