

# Set up a virtual server on AWS EC2 Console (in anticipation of hosting Shiny apps)

Ronald (Ryy) Glenn Thomas

10/28/23

## Table of contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Hosting</b>	<b>2</b>
2.1	Select a hosting service . . . . .	3
<b>3</b>	<b>AWS Working Environment</b>	<b>4</b>
<b>4</b>	<b>Overview</b>	<b>4</b>
4.1	Work Environment Details. . . . .	5
4.1.1	Ssh key pair . . . . .	5
4.1.2	Firewall . . . . .	6
4.1.3	Static IP address . . . . .	7
4.1.4	Domain Name . . . . .	7
4.2	Select and launch instance . . . . .	7
<b>5</b>	<b>Access server</b>	<b>8</b>
<b>6</b>	<b>Appendix 1: Tip 1</b>	<b>8</b>
<b>7</b>	<b>Appendix 2: aws_startup_code.sh</b>	<b>9</b>



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, named for example `power1_app`, we next need to address the task of how to host the app on the web. To answer this challenge we'll break the problem into two parts:

- 1) Create and launch a virtual server. and
- 2) Configure the server to provide secure access to our Shiny app.

Below we recommend a solution we've found to be effective and straightforward. We'll describe how to 'spin up' a server on Amazon Web Services using the EC2 (Elastic Compute Cloud) console. In a separate post [here](#) we'll show how, through the application of Docker, R, Shiny, and Caddy (webserver) functionality we can have a fully functional and secure app available on the web with minimal effort.

## 2 Hosting

No matter what method we use to host a shiny app online we'll need to complete the following pre- and post-launch sets of tasks one way or another:

Pre-launch tasks:

1. obtain a static IP address, e.g. `111.222.333.444`
2. obtain a domain name, e.g. `rgtlab.org`.
3. define a security model, aka a firewall, for the server
4. create a public and private RSA keypair.
5. configure a virtual server. Configuration entails selecting the number of CPUs, the amount of memory, and the operating system.

6. generate the virtual server.
7. associate the domain name with the static IP address.
8. associate the IP, keypair and firewall with the server

Post-launch tasks:

1. install and configure a webserver
2. obtain and install an SSL certificate (to allow encrypted Web communication)
3. setup an authentication method (password protection for app access)
4. configure a reverse proxy i.e. a method to translate https (port 443) requests to Shiny (port 3838).

**i** Not to worry:

At first glance these requirements may appear daunting, but on closer inspection each can be met with relative ease through the use of the right tools.

**i** **Actually ...**

Technically, if the goal is simply to get the app up on the Web and security is not an issue, its not required to have a static IP, or a domain name, 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. Also the IP address will change every time the server is rebooted.

## 2.1 Select a hosting service

There are a number of cloud based server hosting options to choose from: for example 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, in our view, 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.

The first step is to get set up with AWS. To start, open the EC2 console by visiting the URL:

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

(see margin figure)

In the console window choose regional service. For us its “N. California”.

Next create an account, or sign in. Once you're logged in navigate to the EC2 dashboard. Its through this dashboard (aka console) that we'll define the parameters for the type of server to launch and the mechanisms for communicating with it. We'll refer to these as “Pre-Launch” tasks.

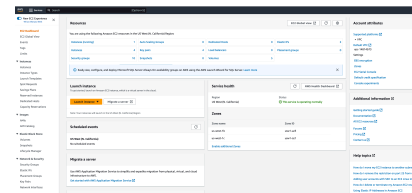


Figure 1: AWS console

### 3 AWS Working Environment

#### 4 Overview

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 security credential (RSA key-pair) to allow remote and secure login to the virtual server once its launched.

2. A firewall or security model which will restrict incoming server access. The firewall closes off all incoming traffic except through those ports specifically named.
3. A static IP address. A static IP 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.

These working environment components are not directly tied to any specific server. In fact, you can define multiple instances of each component. The only requirement is that you pick one of each to associate with each server.

## 4.1 Work Environment Details.

### 4.1.1 Ssh key pair

In order to securely communicate with the server we need to exchange a key pair with AWS. The pair consists of a **private** key and a **public** key. The EC2 based generation process will place the public key in the `~/.ssh/authorized_keys` file. We can define a key pair in one of two ways in EC2. Either, generate the pair 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 (local generation) 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](#). Also detailed instructions are available on AWS [here](#)

In the interactive dialog that follows name the key prefix something like `power1_app.pem`. The dialog will ask for a passphrase. You can enter a phrase for an additional level of security, but its not required. The `ssh-keygen` program will generate two files: `power1_app.pem` and `power1_app.pem.pub`

To complete the process return to the EC2 dashboard and select **Network & Security/Keypair/Actions** and then **Import key pair** in the left panel. Enter the name `power1_app` and select the **Browse** button. Navigate to the file `~/.ssh/power1_app.pem.pub` file 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. The keys will be created and the private key `power1_app.pem` will be downloaded to the local machine and should be placed in the `~/.ssh` directory. Lastly, change the permissions for the private key so only you can access with the following command:

```
> sudo chmod 400 power1_app.pem
```

#### 4.1.2 Firewall

To create a firewall click on **Security groups** under **Network & Security** settings in the left hand panel. Choose **Create security group** and name the security group and fill in the description with something like `power1_app`.

Under **Inbound Rules** select **SSH** and **HTTPS** from the **Type** dropdown menu. Select **Anywhere IPv4 0.0.0.0/0** for both.

This will create a firewall that leaves open only ports 22 and 443, for `ssh` and `https` incoming traffic respectively.

### 4.1.3 Static IP address

You can use the `elastic` IP service to get a static IP. 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.

### 4.1.4 Domain Name

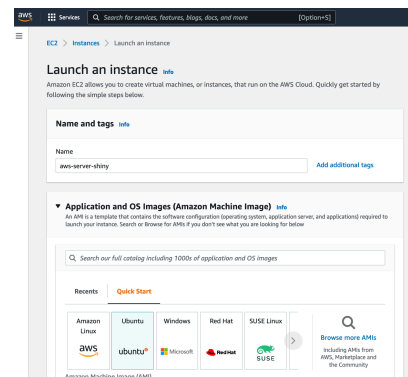
To obtain a dedicated domain name leave the **EC2 dashboard** and go to Amazon route 53 dashboard to select a domain name and associate it with our static IP (select **Service** and search for **Route 53**).

Once a domain name is acquired, e.g. `rgtlab.org`, you can associate it with any IP address, static or dynamic. This can be done via the **Route 53** service. For example, to associate domain name `rgtlab.org` with the elastic IP 13.57.139.31 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 13.57.139.31 in “value”.

## 4.2 Select and launch instance

2. From **Instances** in the EC2 dashboard, click **Launch Instances** Name the server, say `power1_app` and select an operating system for the server. We recommend the **Ubuntu** OS. Ubuntu is a mature Linux distribution based on **Debian** Linux. Click the **Ubuntu** button. (see margin figure)
3. Next choose an instance **type**, e.g. `t2-micro`. Different instance types are combinations of number and architecture of processors, memory, storage capacity, and network performance. The `t2-micro` type has 1 cpu and 1 GiB of memory, for example.



4. choose a Key pair e.g. select `power1_app` the drop down list.
5. Select existing security group, e.g. use `power1_app` from your environment.
6. choose a storage amount. e.g. enter 30 GB of EBS General Purpose SSD GP2 . Thirty GBs is the maximum allowed in the ‘Free tier’ of servers on AWS. In our experience smaller disk sizes can lead to problems.
7. Under **advanced options** scroll to the bottom and browse to ??? if you have code you want to run at startup. see an example here `aws_startup_code.sh`.
8. click **Launch Instance** to launch the server.

After the instance launches open the Elastic IP dialog under **Network Security** and the **Actions** button and associate the IP address with the new instance.

## 5 Access server

On your laptop log into server with

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

## 6 Appendix 1: Tip 1

### Tip 1.

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

```
Host rgtlab.org #domain name
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 # private key
```



then we can ssh into the new server with the abbreviated command

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

## 7 Appendix 2: aws\_startup\_code.sh

```
#!/bin/bash
apt update
# apt upgrade
apt-get install curl -y
apt-get install gnupg -y
apt-get install ca-certificates -y
apt-get install lsb-release -y
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

### Add Docker and docker compose support to the Ubuntu's packages list

echo \
  "deb [arch="$(dpkg --print-architecture)" \
  signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

apt-get update
### Install docker and docker compose on Ubuntu
apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin -y
apt install neovim -y
apt install exa trash-cli ripgrep -y
apt install zsh -y
curl -fLo ~/.vim/autoload/plug.vim --create-dirs \
  https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim
```

```
su ubuntu -
usermod -aG docker ubuntu
# set up zsh after first login
# install oh my zsh, with zsh-z and zsh-autosuggestions plugins
# sh -c "$(curl -fsSL \
# https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
# git clone https://github.com/zsh-users/zsh-autosuggestions \
# ${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-autosuggestions
# git clone https://github.com/MichaelAquilina/zsh-you-should-use.git \
# $ZSH_CUSTOM/plugins/you-should-use
# scp (upload) .vimrc and .zshrc:
# scp .zshrc .vimrc rgtlab.org
```