Webpage From A Docker Container Running On An AWS Virtual Machine

Introduction And Background

Figure 1: Docker Logo

Please make sure you've first read the sibling post to this one - The last lesson covered how to serve a static website from a Docker container. This next step is a very real world operational step. We want to launch that same website container on a production level cloud platform - DigitalOcean.

Digital Ocean is like AWS, Azure, and GCP in principle, but it is more simple than all of those. I like to say that it's more elegant - beautiful and powerful - with just what we need. While Digital Ocean is powerful, it's more of a "bring your own tools and approaches" kind of cloud platform. They aren't trying to invent a bunch of new tools that "code it myself" practitioners don't really want or need. And they are beginner friendly.

I am NOT trying to say that the other cloud platforms aren't good in their own ways. I am only trying to stress how Digital Ocean is different and advantageous. Use what you most like and hopefully you can give all of them a fair try.

Find this and all the other lessons of this Docker Mastery journey in my Docker Mastery repo on DagsHub. DagsHub is the Cloud Git Repo system for data scientists and machine learning engineers. And just so you know, you can get a free DagsHub account.

In the previous lesson, we looked at my repo Docker Served Static Website in my DagsHub repo for Docker Mastery. We went over lesson 2.0 of the Beginner Lab for Docker on the Docker docs website. I hope you will look through the earlier lessons for this **Docker Mastery Journey** if you feel like you are missing something in this lesson.

I've tried to include the same essential content is this lesson as the last one, so that you needn't look back at the previous one to work through this one. In this lesson, we will learn how to launch the same container, that we launched from our localhost on our own machines, and from a DigitalOcean droplet (VM), from an AWS VM. These have been **ONLY** conceptual level lesson to help with this learning step. There's still more ground to cover for:

- establishing a domain name for the page,
- establishing greater security,
- making the page dynamic,
- adding a backend,
- and adding a lot more standard website functionality,
- creating an API,
- building a test platform,
- adding secure logins for users,
- and many more things to make a complete functional website or SaaS.

This lesson is to show how to serve a simple website from a Docker container running in a virtual machine on DigitalOcean.

The High Level Procedure

Some of you, who have never done this, may only need the high level procedure. This would be due to your overall experience with such things. Even if you need every detail, this high level procedure will help you. The high level steps of our procedure will also serve as the approximate or exact titles for each section that covers the procedural steps in detail.

Here are the high level procedural steps that we will follow. 1. Create a free DigitalOcean account. 1. Create a new project within your DigitalOcean account. 1. Create a droplet (virtual machine) within your DigitalOcean account. 1. Establish a secure shell connection between our local machine and our DigitalOcean droplet. 1. Use the command line of our droplet from our local machine. 1. Install the Docker engine and Docker Compose on our droplet. 1. Setup SFTP between our local machine and our droplet to transfer files from our local machine to the droplet file system. 1. SFTP our Dockerfile and our HTML file to our droplet. 1. Build our Docker image to create a Docker container on our Droplet. 1. Run the Docker container in detached mode on the droplet. 1. Check to see that we can access our simple static website that is running in a Docker container on our DigitalOcean droplet.

You may come across some other great tutorials that go about things a little bit differently. Those procedures are OK. All the procedures are roughly accomplishing the same things, but they may be using different tools to accomplish the same steps. However, I must confess that I could not find a procedure like the one that I have presented in the previous paragraph. I was striving to do the above steps, or something close to them, in the simplest possible way.

Why is my procedure the way that it is? Two things. First, I REALLY like to find the simplest way to do something. Then, after I see how to do that, I am eager to add helpful sophistication. Second, I am now beginning to realize that I am a bit old school. I like the simple low level tools that have been around a long time. And trust me, there are steps that I take, that true computational elders would snicker at. The main thing I want to encourage you to do is to see my procedure conceptually. Then, over time, change this procedure to use step descriptions and tools that you prefer.

Finally, I worked through all the above steps over many days. I made many mistakes along the way. Many of them were basic conceptual mistakes that made me laugh at myself and sometimes cringe in embarrassment. I admit to such things so that if you encounter similar learning hurdles, you won't feel alone. It still amazes me how, when I am trying something completely new, I can forget basic principles that I already knew. However, once I get over the pain of lost time and embarrassment with myself, I am grateful to be more skilled for the future.

Creating A Virtual Machine On AWS

After you've created an AWS account, or logged into your existing account, you should find a console close to the one shown below. I believe AWS will be changing some of their look / feel aspects in the AWS Console around September 2022. After a bit of research regarding which **Compute** resource was most economical, I landed on Lightsail. You can use a different one anytime. If you want to follow along with me as closely as possible, it would be best to choose Lightsail this time.

You'll arrive at a Lightsail console page approximately like the one in the next image. Use the **Create instance** button.

Notice that AWS simply uses your detected location and assigns you a location for your resource. Let's choose Linux blueprints. Let's also choose OS only and the latest Ubuntu as indicated by my choices.

Next, we are shown some details about our Ubuntu image.

I elected to use my existing SSH key on my machine, so I clicked on "Change SSH key pair" and directed AWS to "Upload New". You can see that I browsed to my default id_rsa public key, and it was uploaded

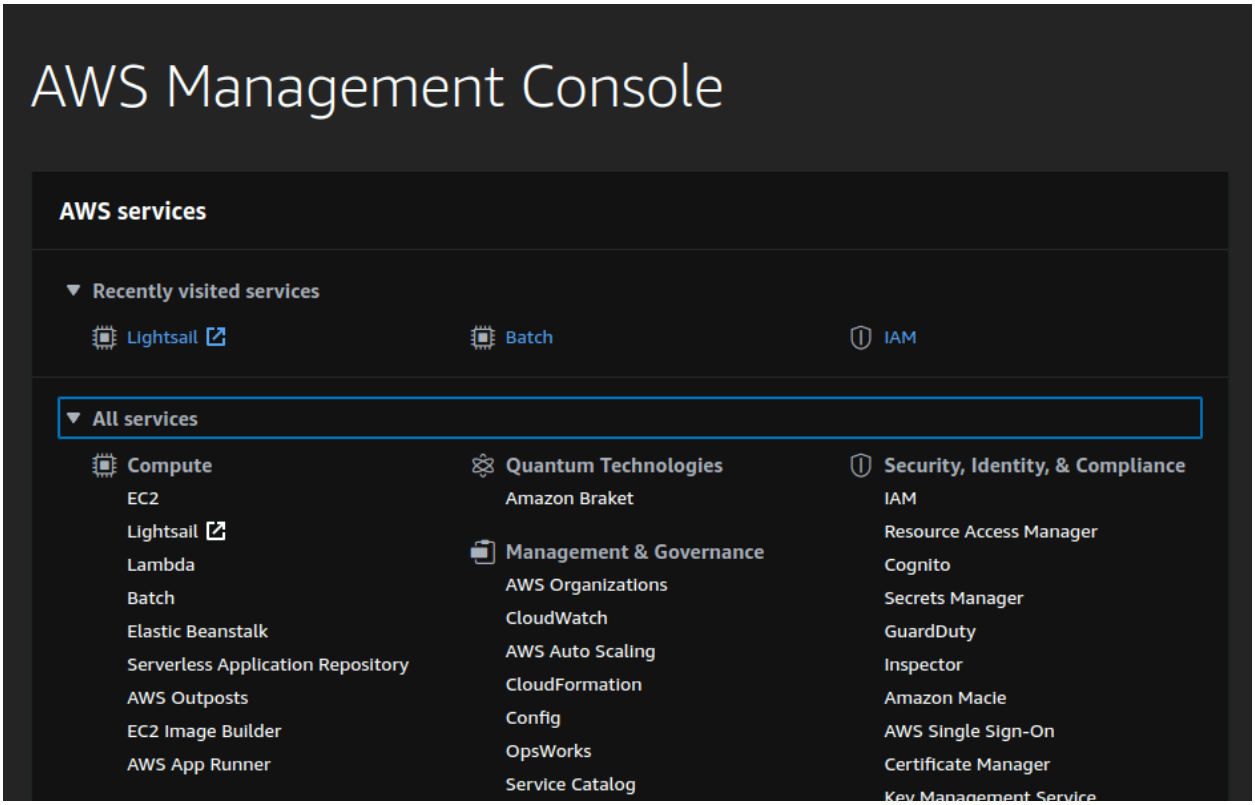

Figure 2: VM Setup Area 0.1

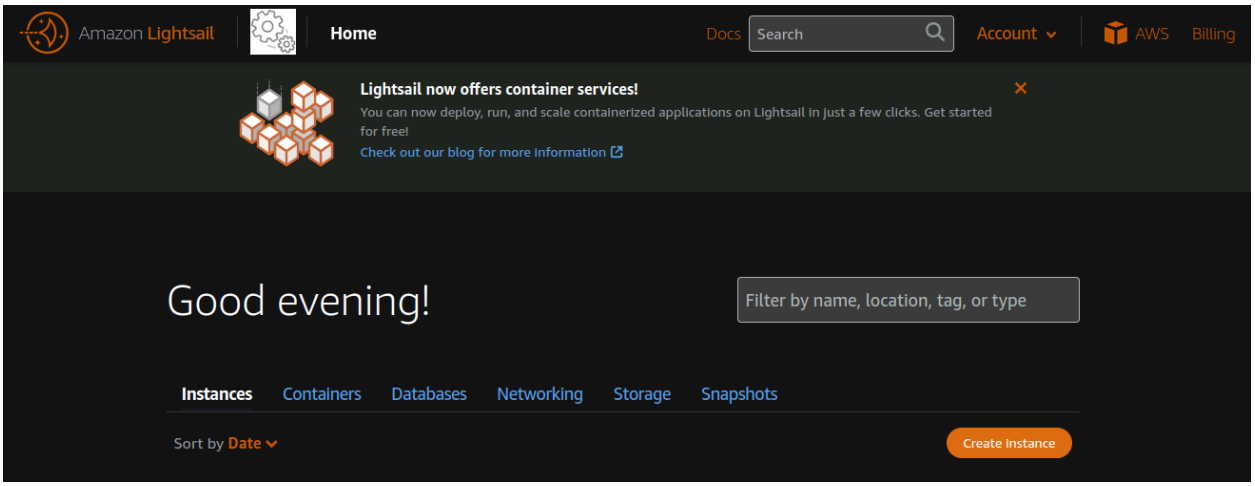

Figure 3: VM Setup Area 0.2

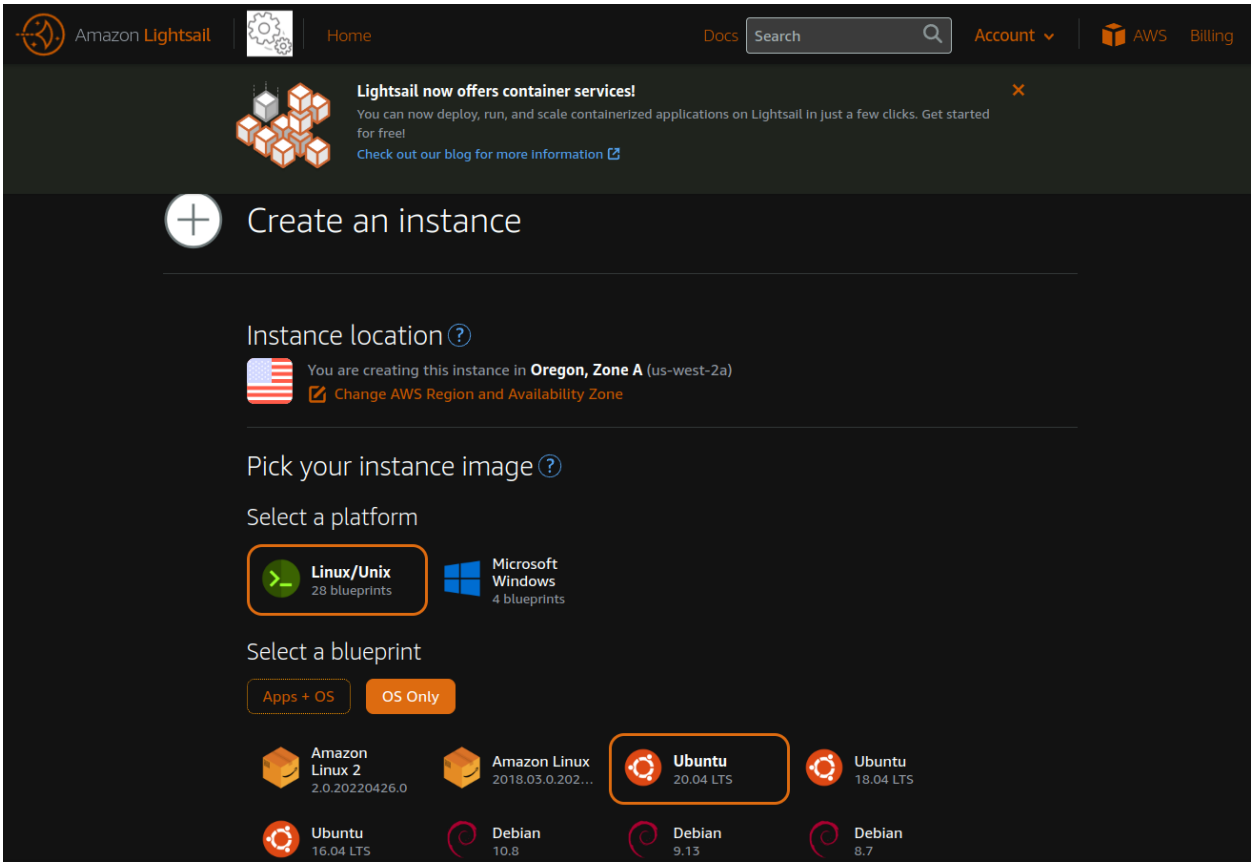

Figure 4: VM Setup Area 1

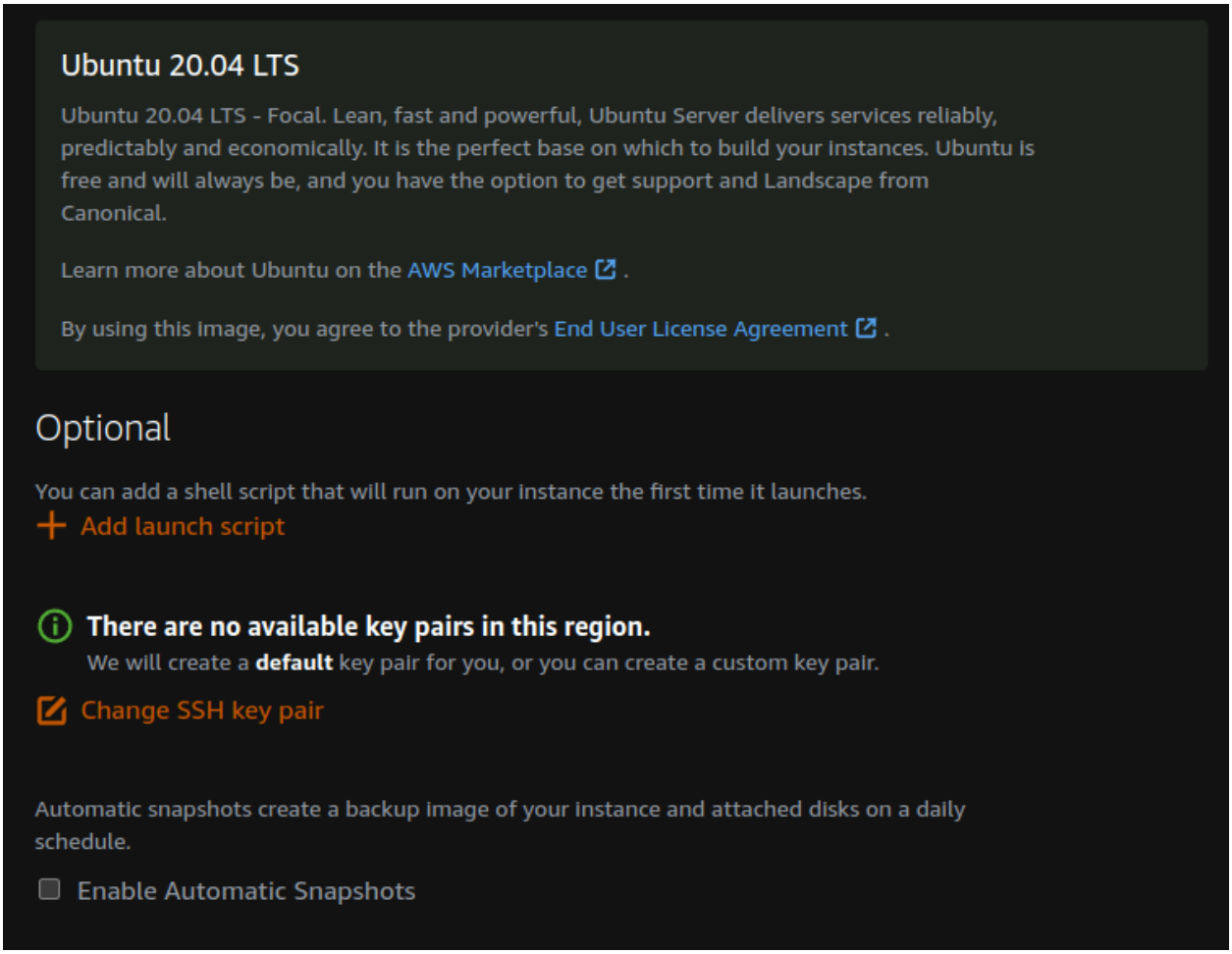

Figure 5: VM Setup Area 2.1

successfully. If you aren't familiar with SSH, I strongly encourage you to take the time to learn to create a key pair, public and private, and only share your public key with applications outside your machine that you wish to connect with such as AWS, GitHub, DigitalOcean, etc. It's much better than using passwords.

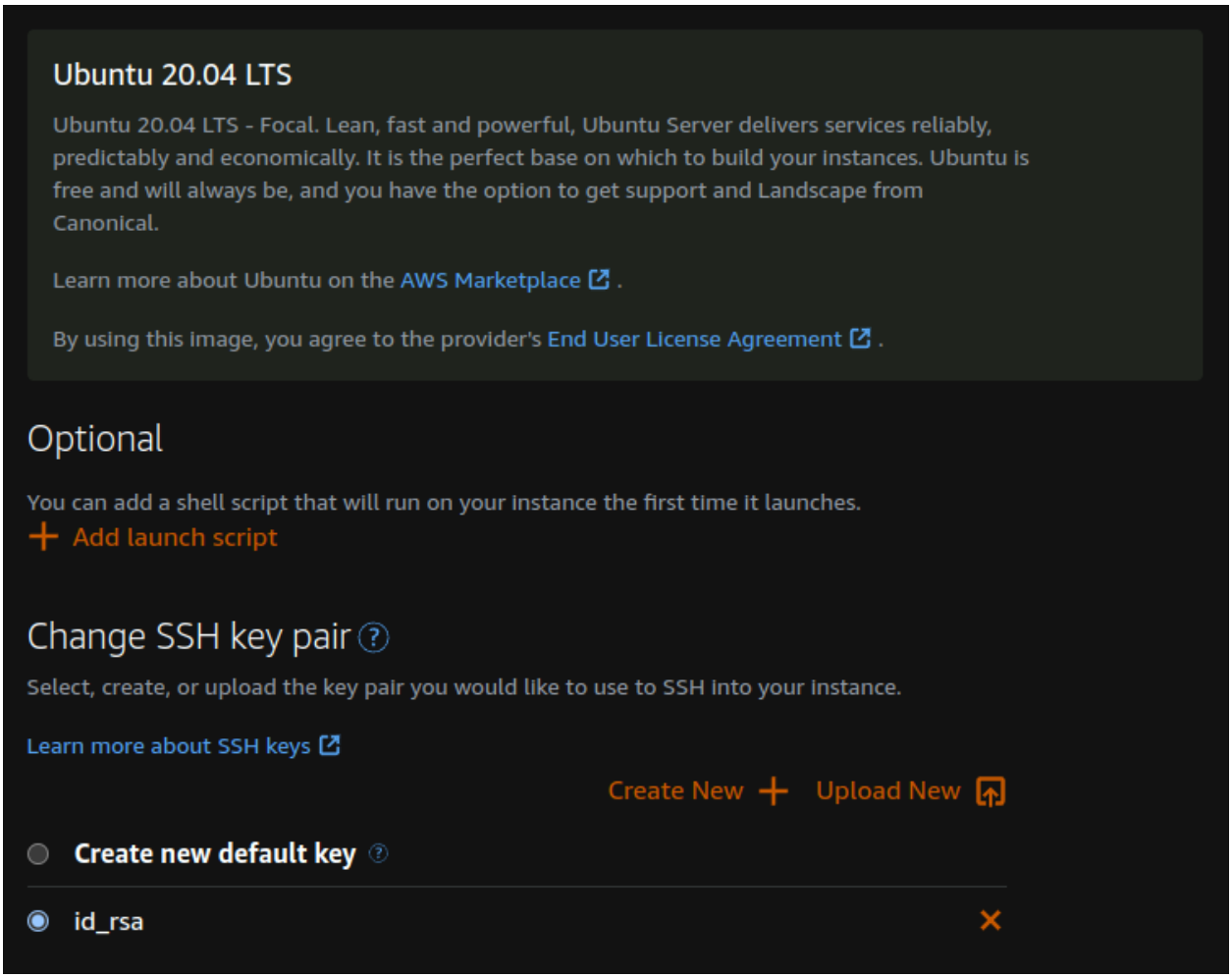

Figure 6: VM Setup Area 2.2

I elected to NOT use "Enable Automatic Snapshots", and, as you can see, I chose the cheapest option for this small test.

I then gave my resourced VM a unique name that I liked, and a tag just in case I needed that for further clarity later. Then I clicked the large "Create instance" button.

It took very little time for my new VM to be built. NOT the IP address. We will need that to connect to our webpage once we run our Docker container.

Install The Docker Engine (And Docker Compose) On Our Droplet

The procedures below are identical to the ones in the previous lesson.

Please see the learning step Docker Concepts and Install to understand the details of the install steps.

For the command line commands in this section, we will **not** show the output lines of the terminal except for the last command. That command is a great way to verify our install. I want to provide THAT output

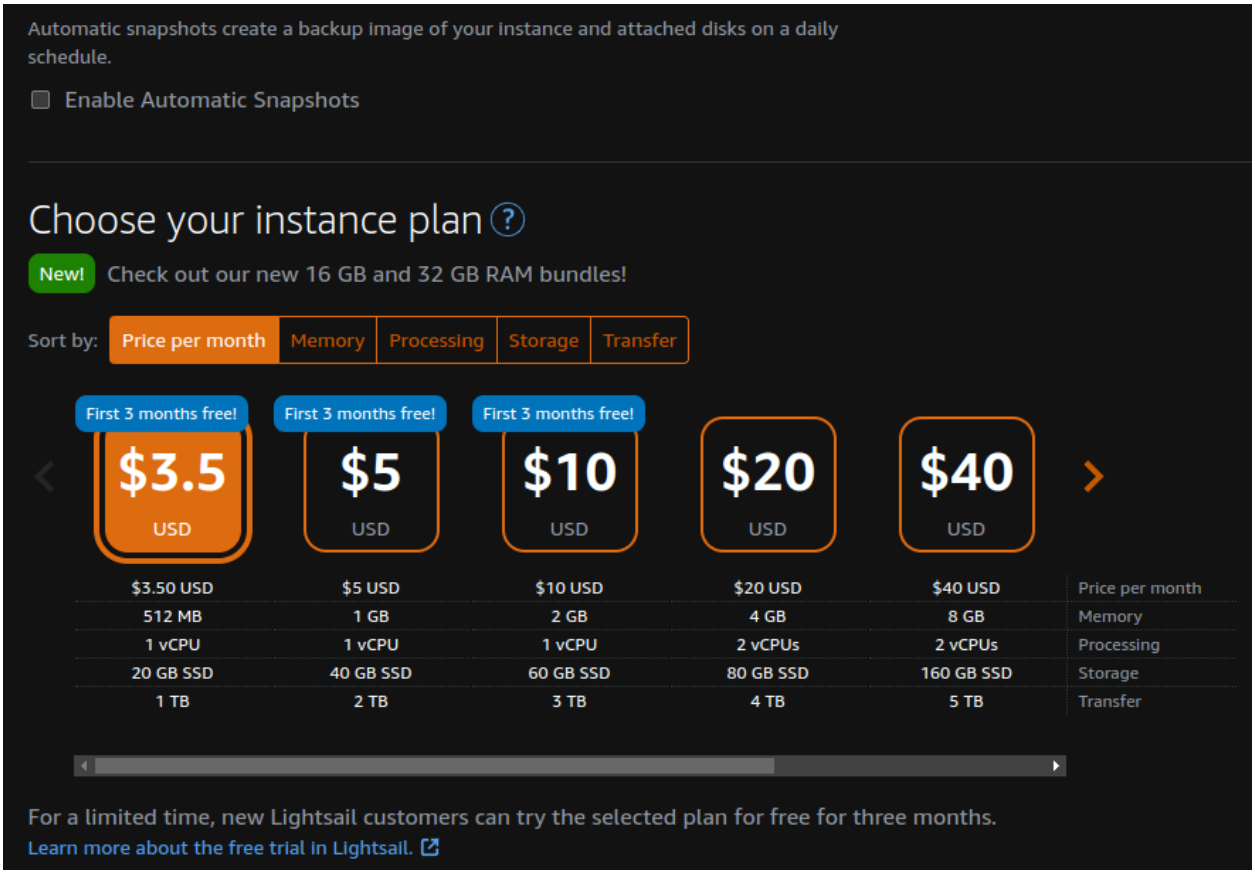

Figure 7: VM Setup Area 3

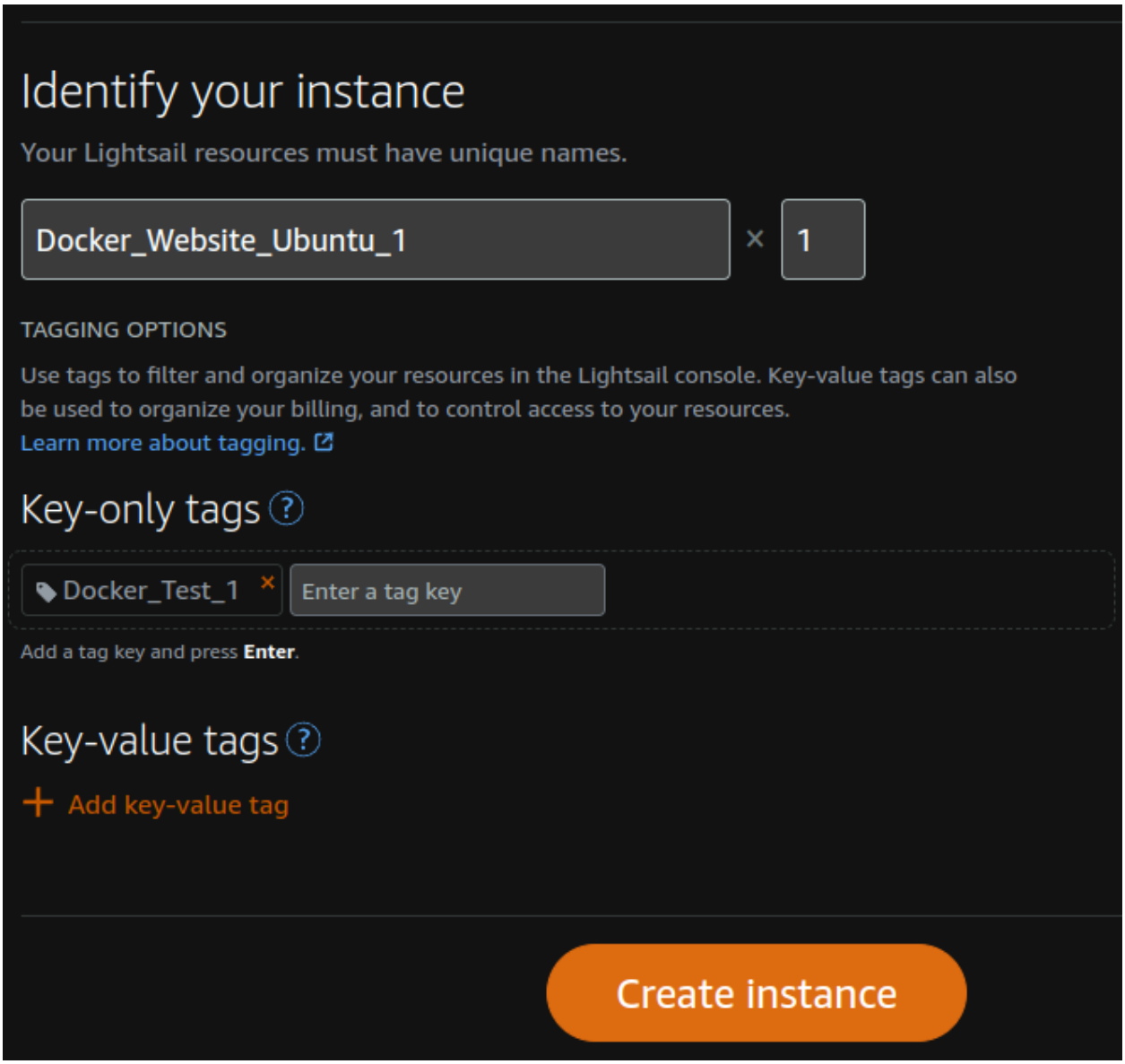

Figure 8: VM Setup Area 4

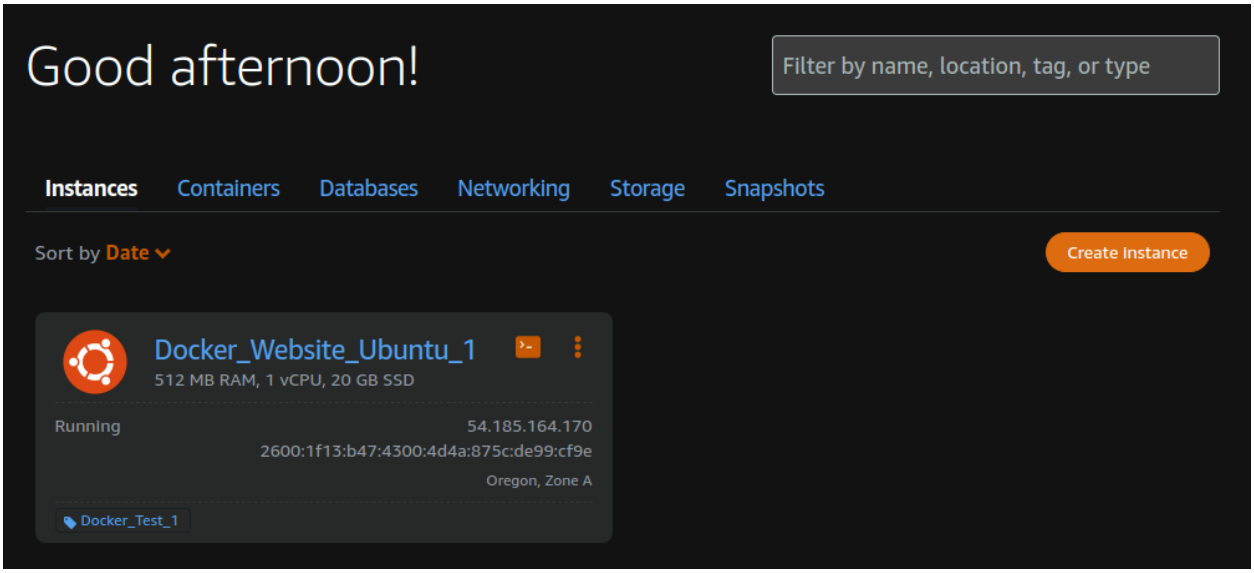

Figure 9: VM Setup Area 5

so that you can detect if anything went wrong with your Docker installation. Let's run these next two commands on our droplet as good stewards of it.

`sudo apt-get update`

`sudo apt-get upgrade` The next group of commands prepare our droplet for our Docker engine installation.

`sudo apt-get install ca-certificates`

`sudo apt-get install curl`

`sudo apt-get install gnupg`

`sudo apt-get install lsb-release`

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/dock
```

```
echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] h
    $(lsb_release -cs) stable nightly test" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Finally, we do an update for the new packages installed.

`sudo apt-get update`

This next line actually installs the Docker programs that we need to run the Docker engine.

`sudo apt-get install docker-ce docker-ce-cli containerd.io` The next command line command tries to run the hello-world image to launch it in a container. It is not yet on our droplet, so Docker will then pull that image / container from Docker Hub, and then run it.

`sudo docker run hello-world`

The output on my Linux box for this command is shown below.

Checking Docker Installation

The infamous Hello Docker image loaded automatically from Docker Hub and launched and ran successfully. **NOTE** that the Hello Docker container runs and then exits, so when you run a `docker ps`, Hello Docker

will not be shown as a running container, but, until you delete it from your Docker images, it will still show up.

```
ubuntu@ip-172-26-0-144:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:80f31da1ac7b312ba29d65080fddf797dd76acfb870e677f390d5acba9741b17
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

ubuntu@ip-172-26-0-144:~$
```

Docker compose is not yet installed. We do not need it for this step, but we will use it in the future. If you want to install it for the experience of it, the commands are below. Again, please see the learning step Docker Concepts and Install to understand the details of the install steps. I also like this tutorial by "did code" for explanations.

Obtain Docker Compose, install it, and set it up.

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Make sure it is executable.

```
sudo chmod +x /usr/local/bin/docker-compose
```

Some people like to create this symbolic link. You can determine if you need or want this.

```
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

A good way to check your Docker Compose install is to check the version from the command line with the following command.

```
docker-compose --version
```

You will see something similar to the following output if all is good.

```
docker-compose version 1.29.2, build 5becea4c
```

Connecting To Your AWS VM Via FileZilla SFTP

I encourage you to follow the directions in Connecting to your Linux or Unix instance in Amazon Lightsail using SFTP to properly connect FileZilla to your AWS VM. I found this procedure helpful, because finding the information needed for connecting to an AWS VM was a bit different than expected. Your connection information should look relatively similar to mine when you are done.

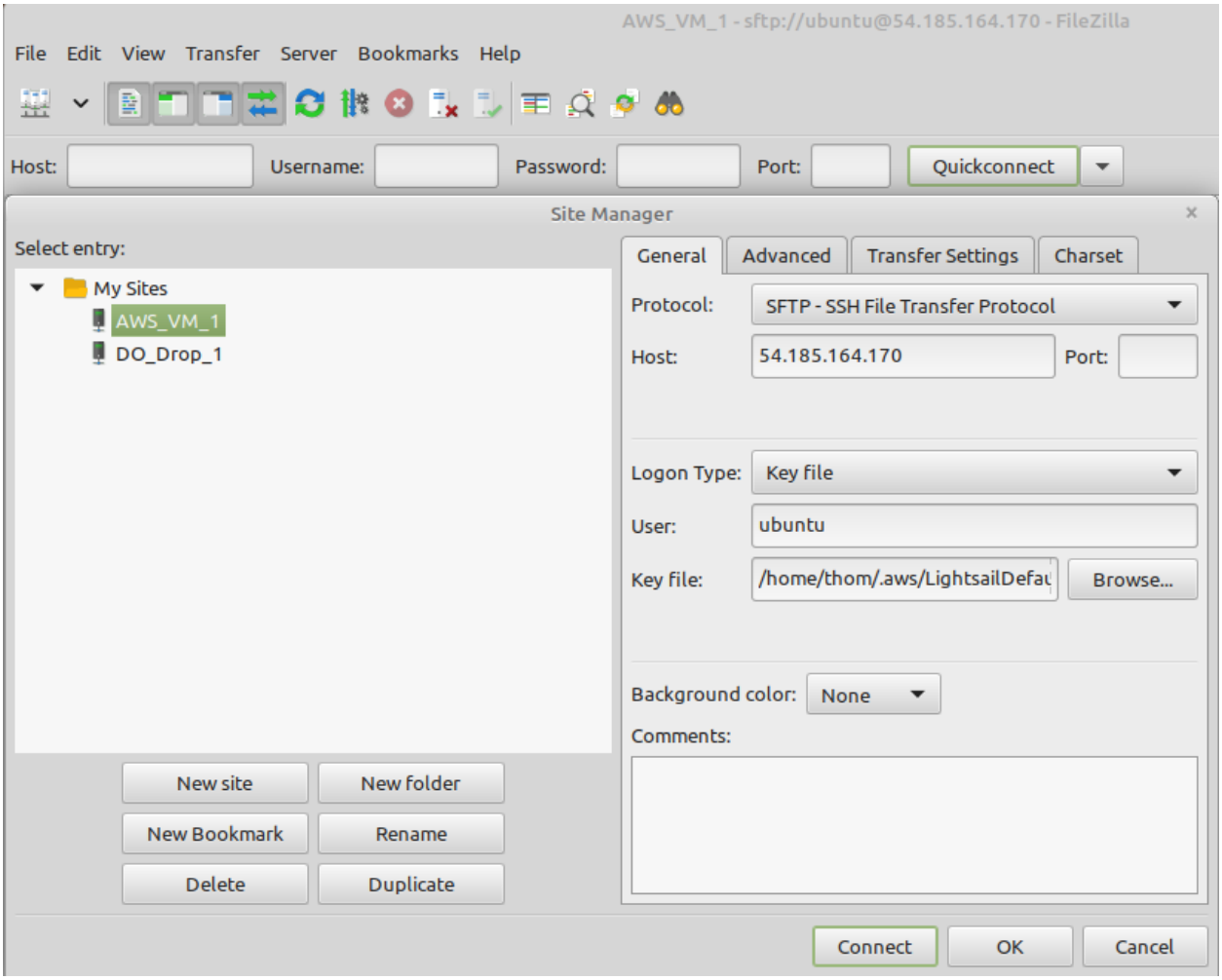

Figure 10: Setting Up SFTP To Connect To The AWS VM With FileZilla

Once I was connected successfully to my VM, I "right clicked" on the `\home\ubuntu` directory to "create directory and enter it" the website directory. I then transferred the "Dockerfile" that we've been using in the previous two lessons, and the HTML file too as shown.

Then I also used the console for the VM to check that the files showed up using it too.

Wait! How did I get that console to launch? If you check the previous lesson, you'll see that we connected to the DigitalOcean VM two ways: 1. Using our Linux Terminal with ssh, and 1. Using the console provided by the DigitalOcean control panel.

This time, I will not show any images of my Linux Terminal using ssh. The way you launch the console from our VM's control panel is by clicking on the small red terminal icon to the right of the blue **Docker_Website_Ubuntu_1** as shown below.

I would also encourage you to study Connecting to your Linux or Unix instance in Amazon Lightsail. I

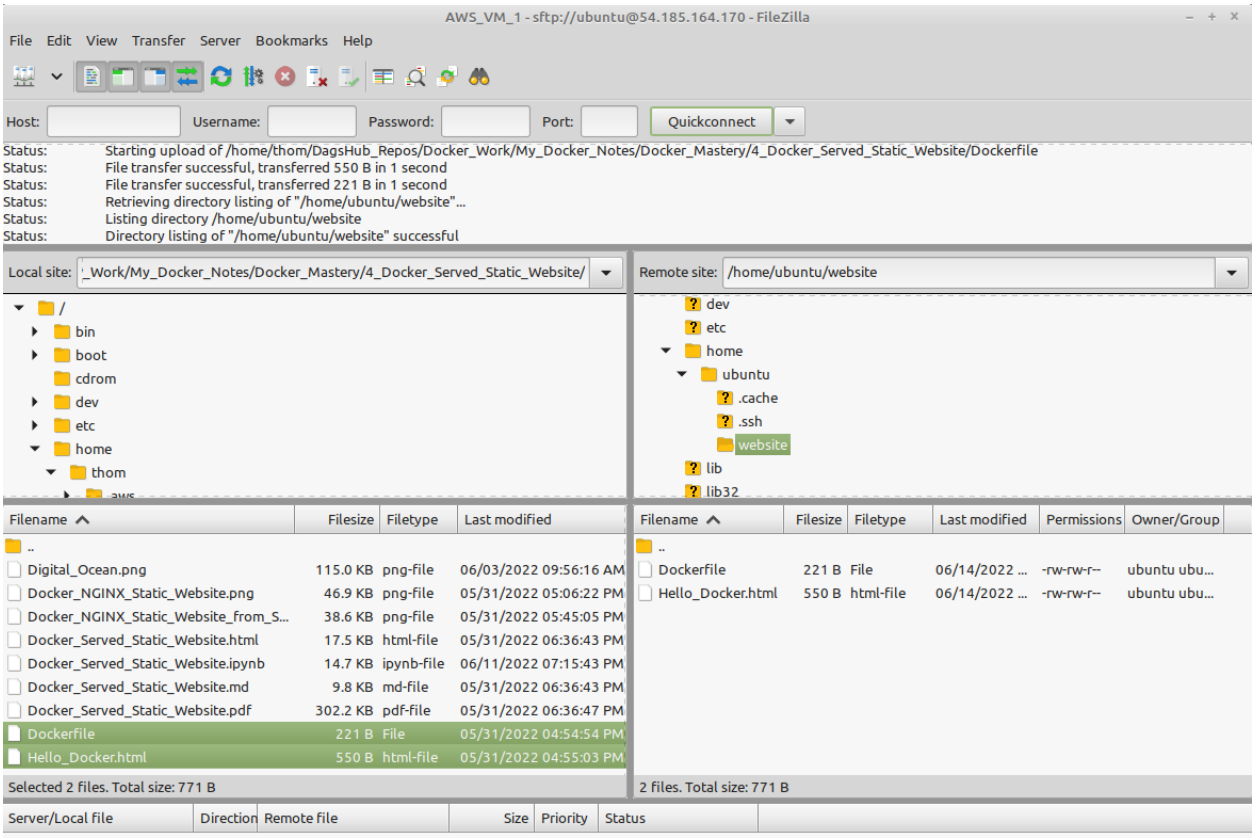

Figure 11: SFTP Transfer Of Dockerfile And HTML File To The New Directory On The AWS VM Complete

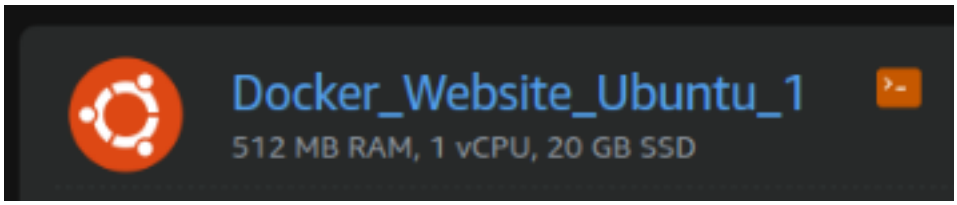

Figure 12: Console Launch Button to Right of Docker_Website_Ubuntu_1

found that page helpful for the first time use of the AWS VM console for many reasons, including how to use the clipboard to copy commands from my machine to the console. It's a bit different than what we were able to do with the DigitalOcean console - not bad, but a bit more bumpy.

When you first launch that terminal, you will see information similar to that shown below.

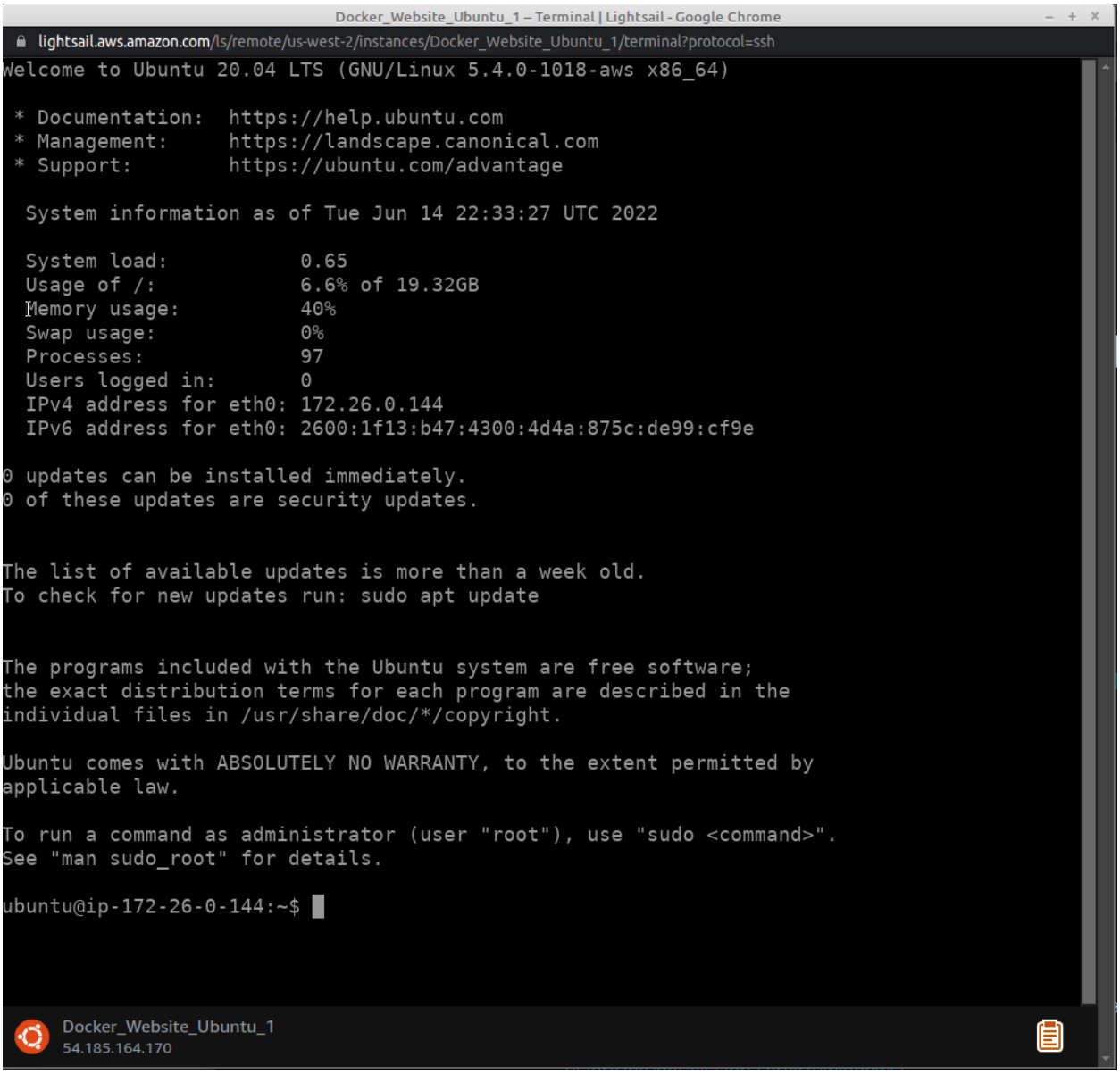

Figure 13: The AWS VM Console When First Opened

Now we can use the terminal to navigate to the location where we SFTP'd our files to, and check that we can see them there.

Build The Docker Image And Run It In A Docker Container

Next, while still in that website directory, we execute `docker build -t website:0.1` to build our image on the AWS VM.

Let's now run `sudo docker images` and `sudo docker ps` to check the state of things.

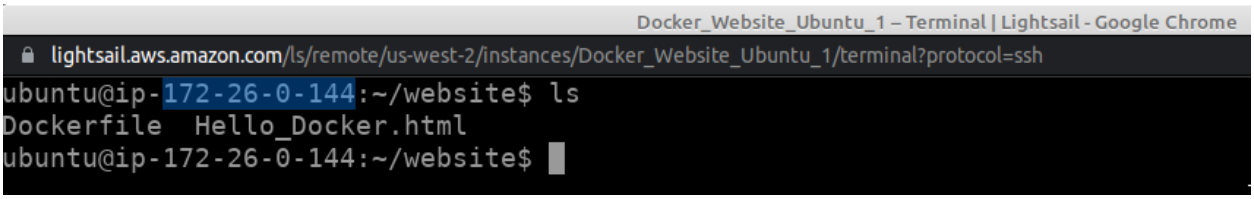

Figure 14: Seeing The Files Moved Via SFTP In The Console

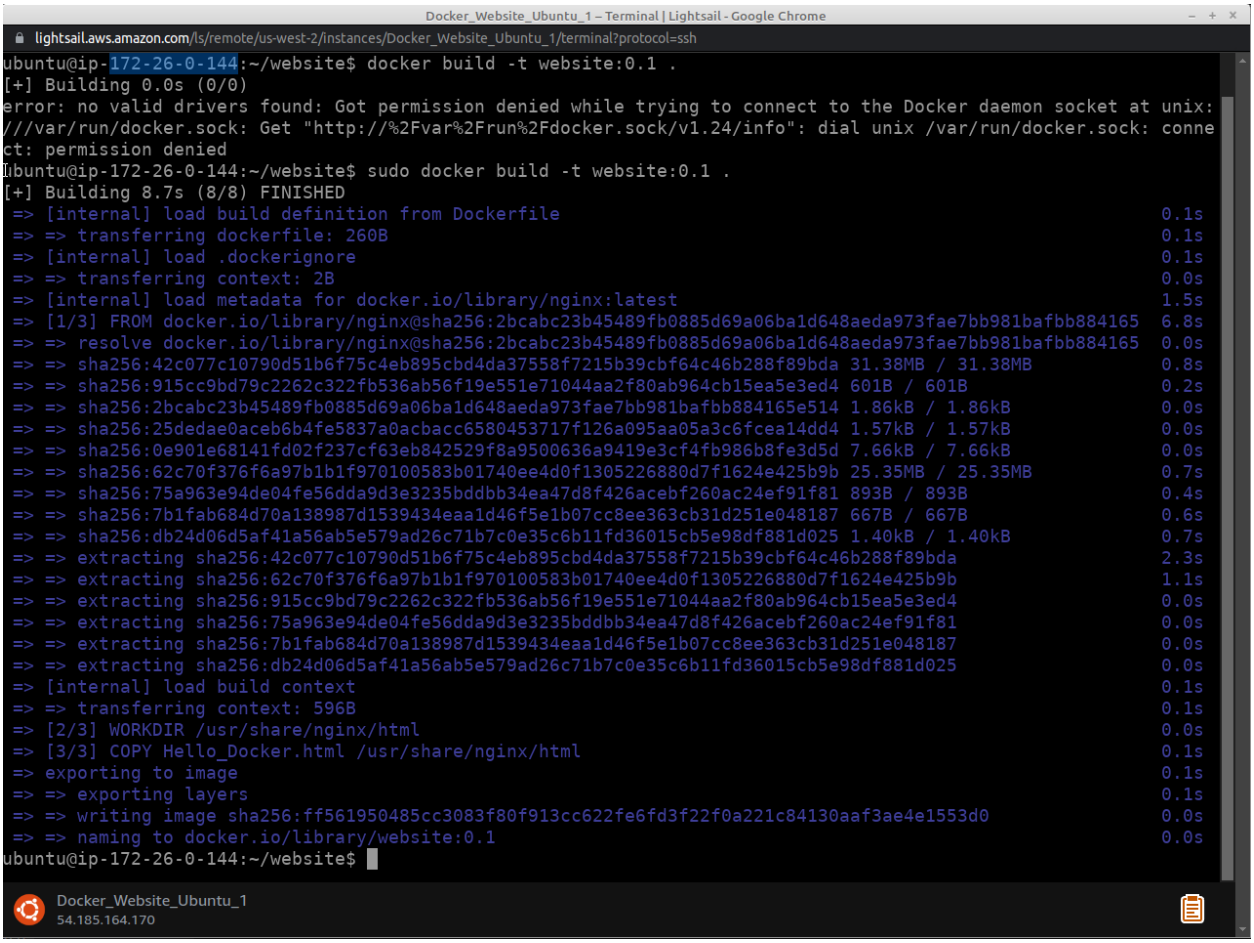

Figure 15: Building The Docker Image From The VM Console

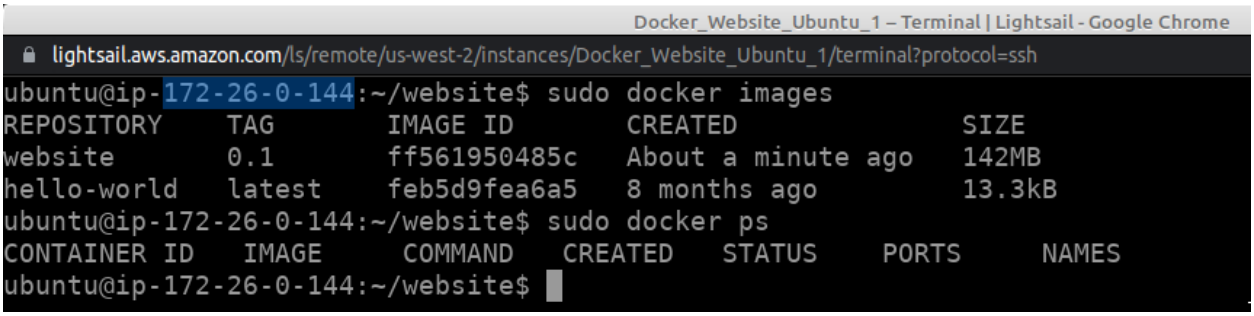

Figure 16: Checking Docker Images and Running Processes From The VM Console

The image is built, but no containers are running yet, which is as expected. Now let's run the website image. We will expose port 80 of the container to port 80 of the AWS VM.

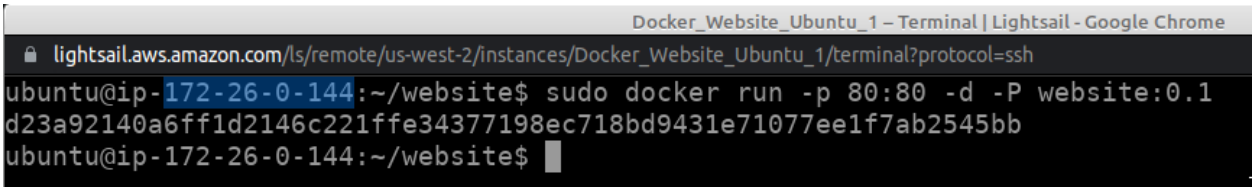

Figure 17: Running The Docker Container From The VM Console

The container launched successfully and returned its ID. Nice! Now we use `http://54.185.164.170/` in our browser of choice to see if our page will serve. Holy Containerized Webpage Service Batman! It worked on Chrome!

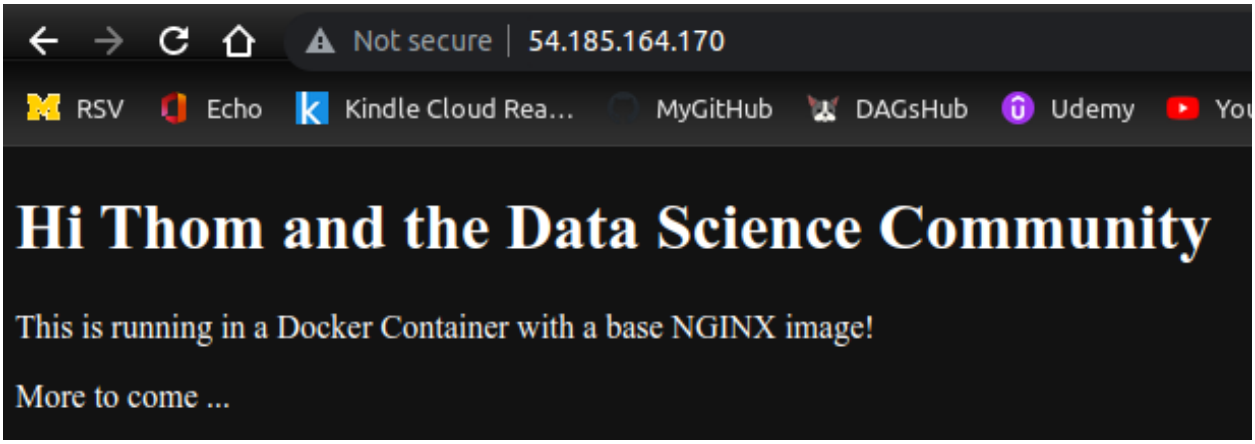

Figure 18: The Simple Website Working In Chrome

Holy Firefox! It worked in Mozilla Firefox too! We are dark web meisters now.

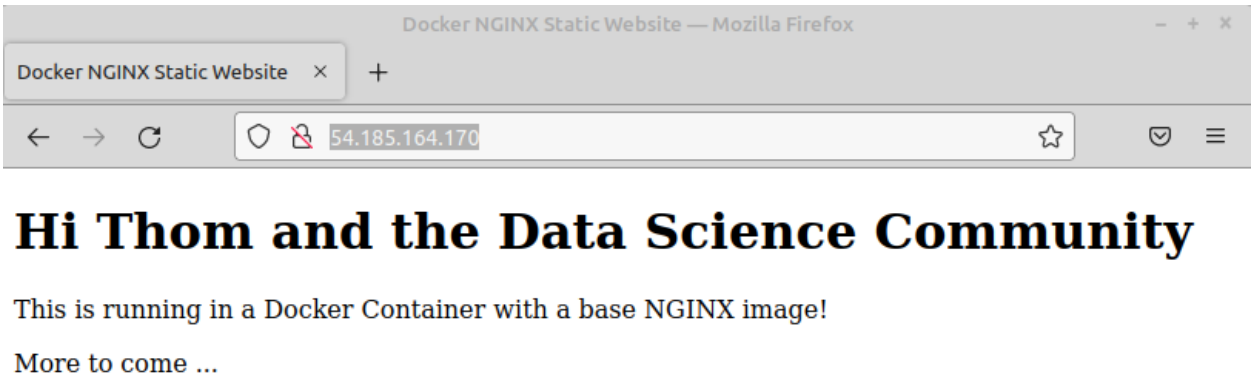

Figure 19: The Simple Website Working In Firefox

Summary

Well, that's cool. Now we just need to grow our container with more functionality and we can serve up any application that we like. We have a lot more to do, but at least we have the end-to-end working on two

major cloud services now. YeeeeHaaaaah!