Chapter 34: Deploying with AWS

By Jonas Schwartz

Amazon Web Services (AWS) is by far the largest Cloud provider today. It provides a number of service offerings which simplify the deployment and maintenance of applications. In this chapter, you'll learn how to use a few of these to deploy a Vapor app.

Before starting

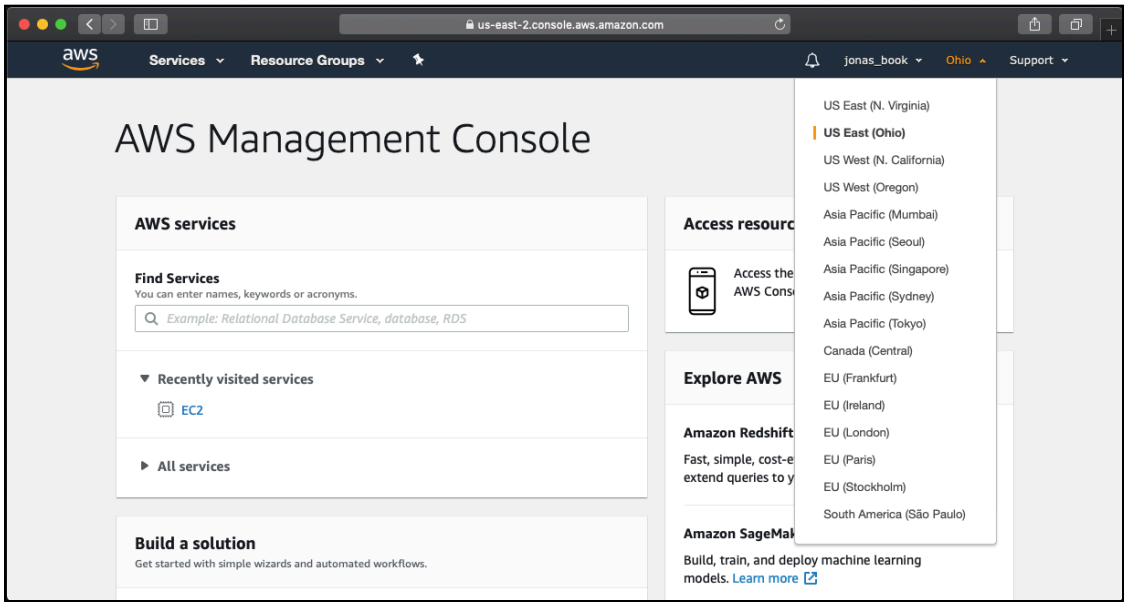
To perform the steps in this chapter, you must have an AWS account. If you don't already have one, follow the instructions at https://aws.amazon.com/premiumsupport/knowledge-center/create-and-activate-aws-account/ to create one.

Setup your AWS instance

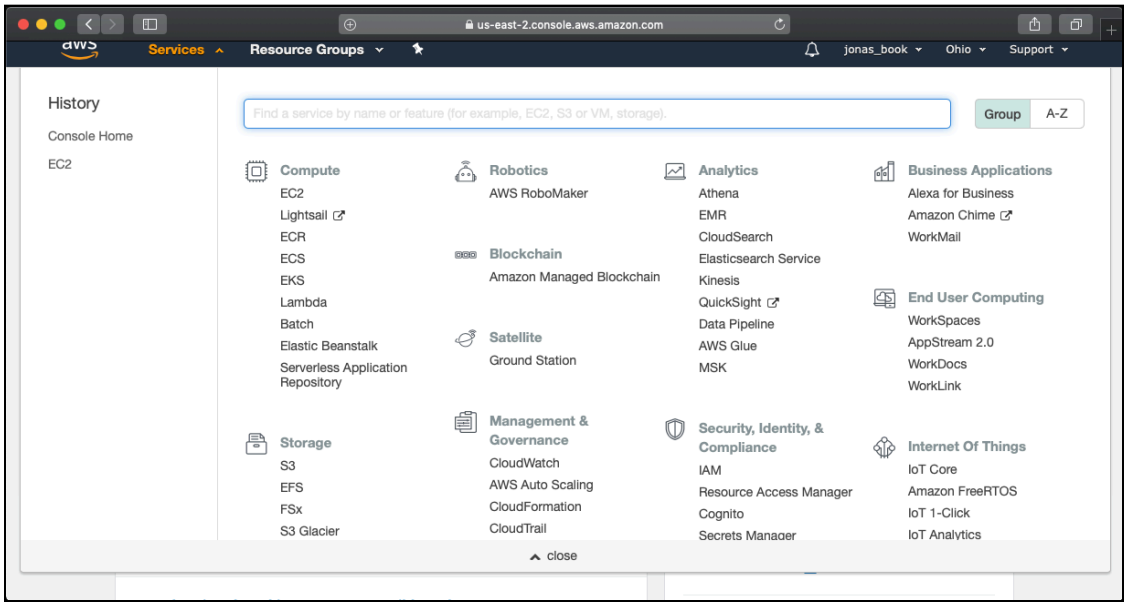
Your first step is to start an EC2 instance. EC2 is AWS Virtual Machine product. This gives you a plain Linux machine you can use to run your Vapor application.

For this example, you create an Ubuntu 18.04 instance. 18.04 is the latest LTS (Long Term Service) version from Ubuntu.

First, you must decide which region you want to use. Click the dropdown next to your name and select the region closest to you.



After selecting the region, click **Services** and **EC2**.

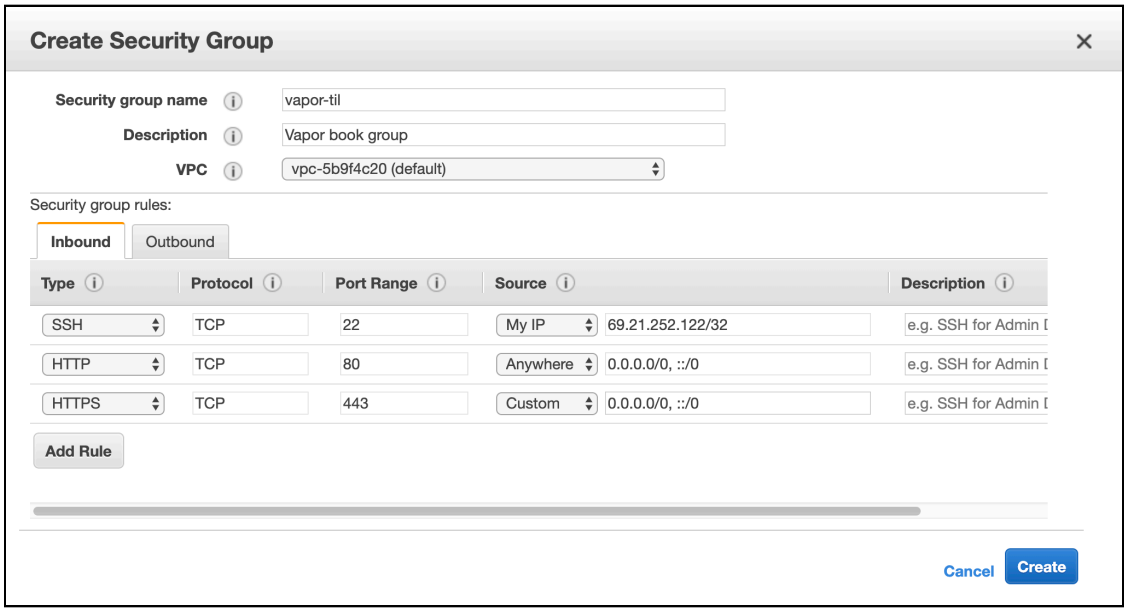


Before you start your instance, you must create a **Security Group**. This is essentially the firewall for your instance, allowing you to specify which ports are open on the server.

Click **Security Groups**, then click **Create Security Group**.

In the resulting dialog, enter a **Security group name** and **Description** that will make it easy for you to associate it with your app. For this example, name your group **vapor-til**.a

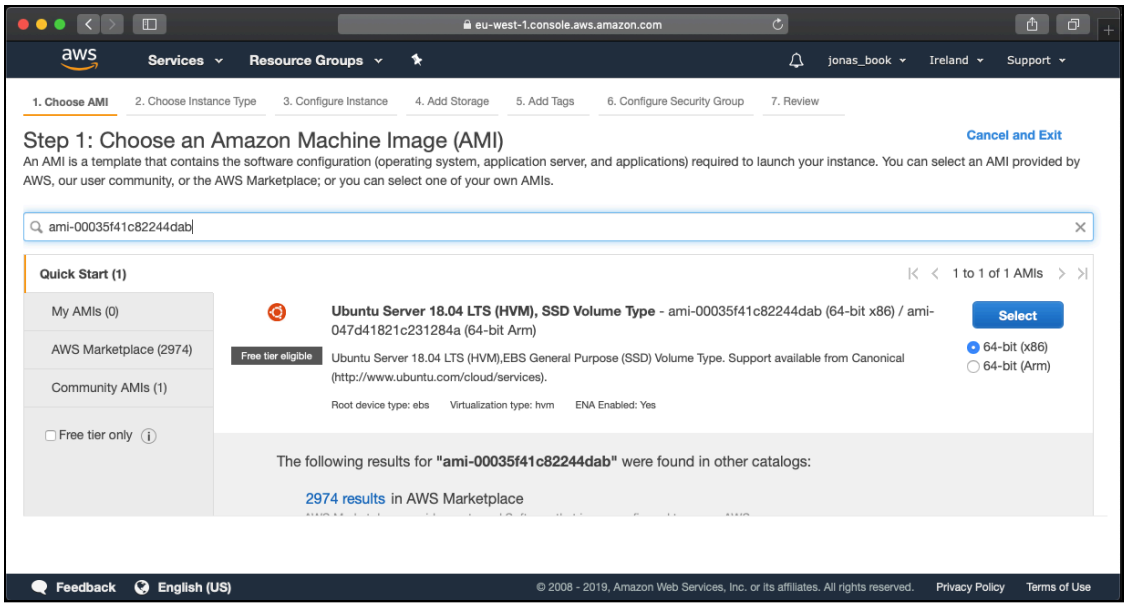
With the **Inbound** tab selected, click **Add Rule** to add a new rule. Use the dropdown under **Type** to select **SSH**. Repeat the process for **HTTP** and **HTTPS**. Your screen should look similar to the following:



Click **Create** to create your security group.

You're now ready to create your instance. Click **Instances** and **Launch Instance**. This begins a seven step process to configure and launch an EC2 instance.

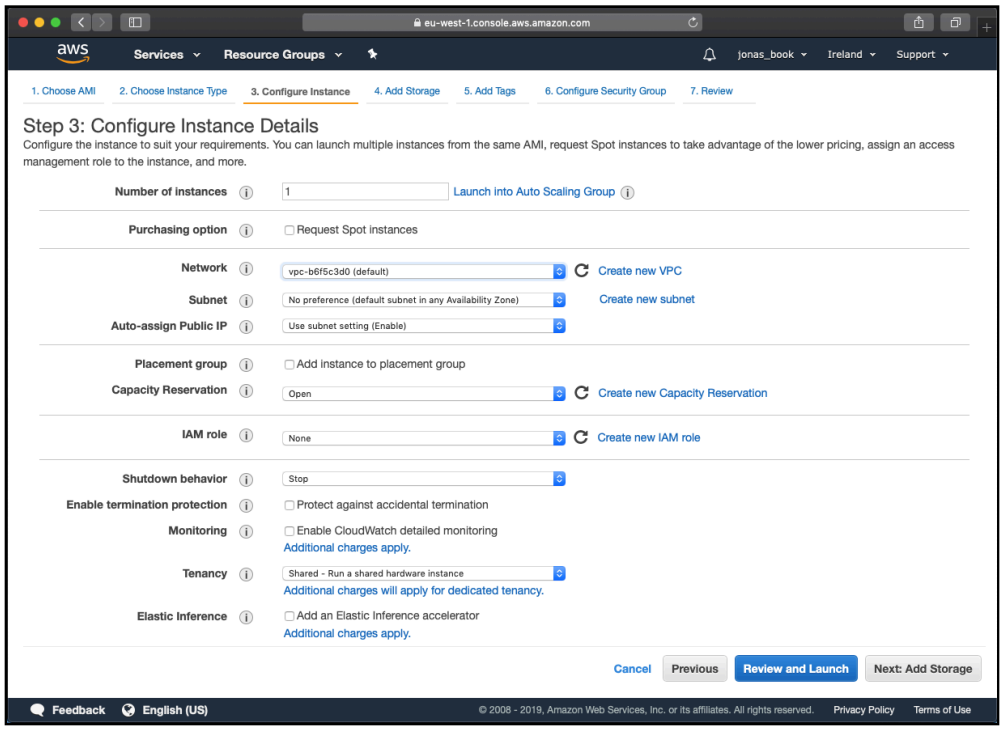
First, you must pick an Amazon Machine Image (AMI) as the base for your EC2 instance. To simplify finding the correct AMI, enter **18.04** in the search box and check **Free tier only**. Choose the one called **Ubuntu Server 18.04 LTS** by clicking **Select** in its row.



Next, you'll select your **Instance Type**. AWS highlights the default, **t2.micro**. This is **Free tier eligible**, meaning you get 1GB memory and 1vCPU for free for the first 12 months you have your account. You'll stick with this choice.

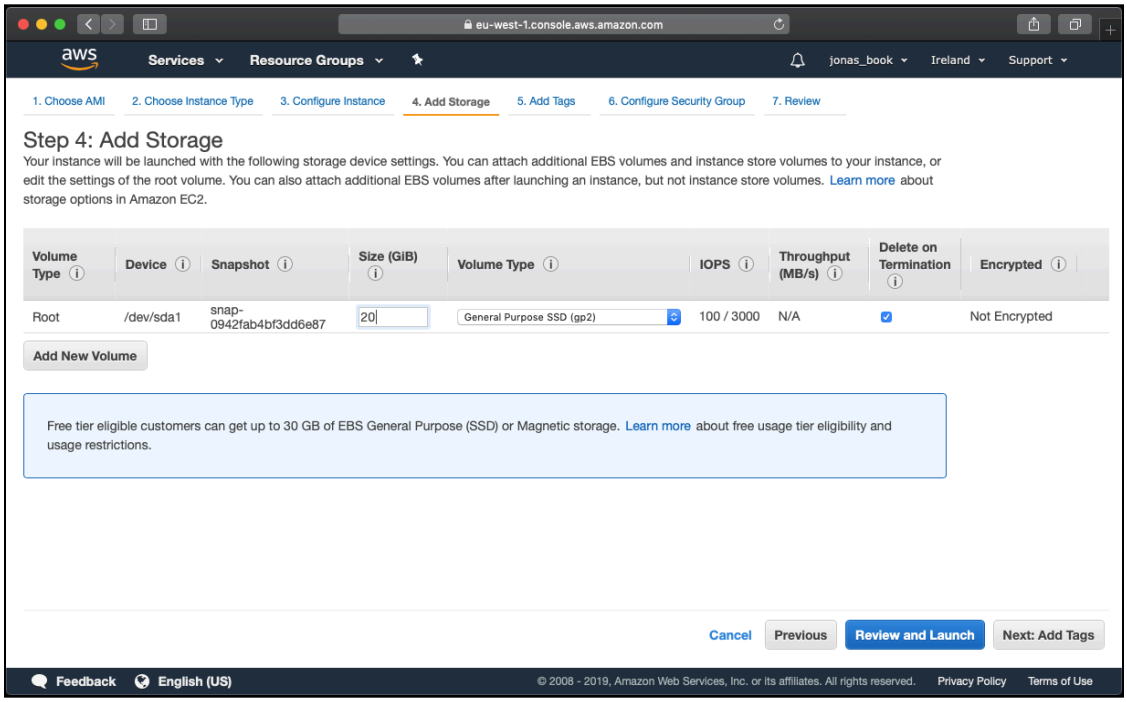
Click **Next: Configure Instance Details**.

On this page, you can set up various details for your instance. For this example, simply leave everything as it is.



Click **Next: Add Storage**.

On this page, you configure the volume for your app. Change **Size** to **20**; this will give you plenty of space for your app.

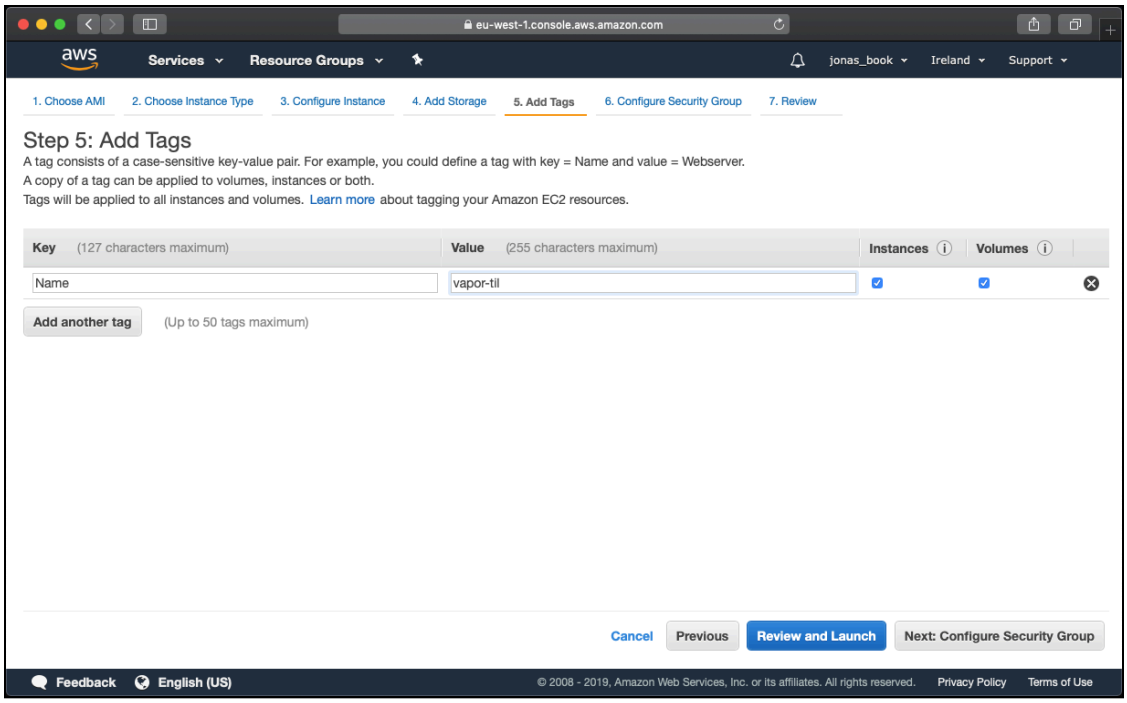


Click **Next: Add Tags**.

This page allows you to add tags to your instance. This step is optional but doing so will simplify managing your AWS resources as your usage grows. Click **Add Tag** and enter the following values:

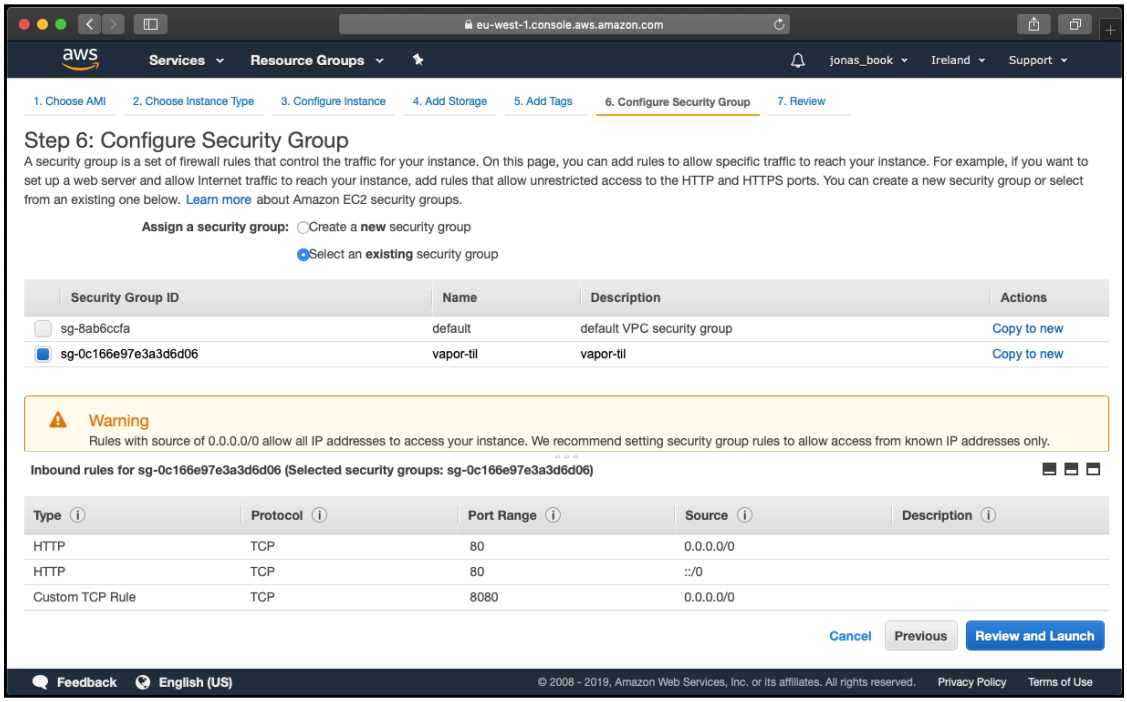
- **Key**: Name
- **Value**: vapor-til

This gives the instance the name **vapor-til**.



Click **Next: Configure Security Group**.

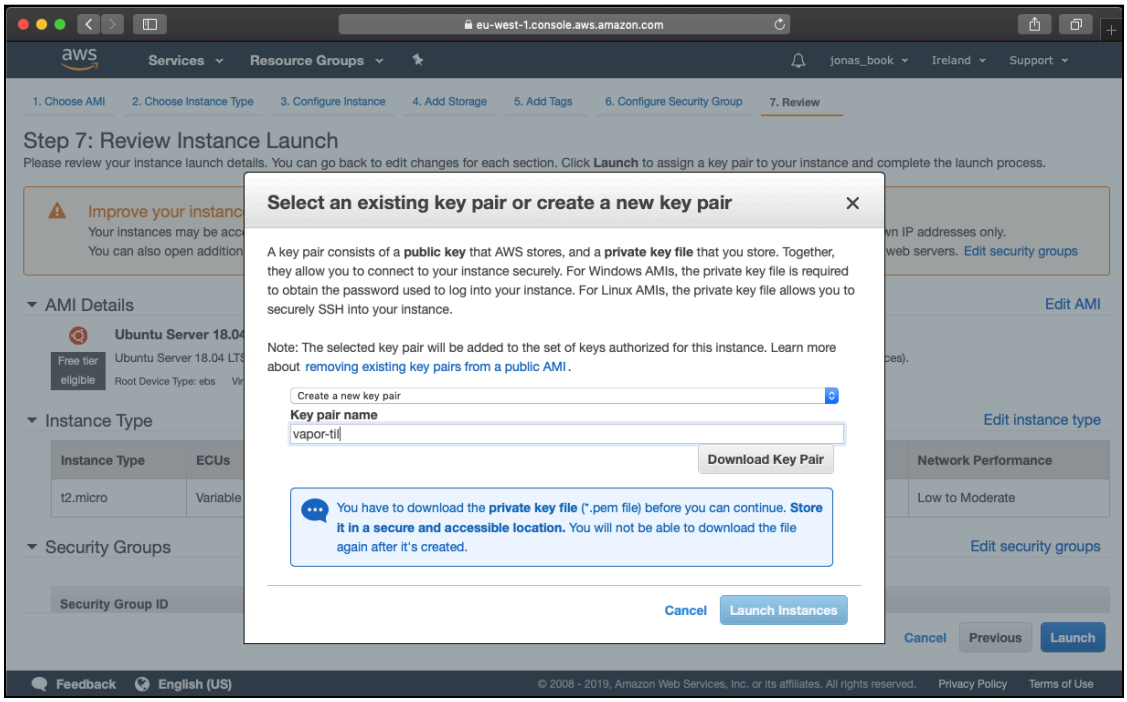
On this page, you'll attach the security group you created earlier to your instance. Click the **Select an *existing* security group** radio button. Then, select your **vapor-til** group:



Finally, click **Review and Launch**.

On this page, you can verify the options you chose previously. When you're satisfied, click **Launch**. AWS will prompt you to either select an existing key pair or create a new one. You need a key pair to allow you SSH access to your instance, so don't skip this step. If you create a new key pair, remember to click **Download Key Pair**.

Once you have configured and saved your key pair, click **Launch Instances**.



AWS will confirm that it is starting your instance. Click **View Instances** to return to your instance summary page. If you're quick enough, your instance will show an **Instance State** of **Pending** with a yellow indicator. After a little while, it will show as **Running** and have a green indicator.

Copy the **IPv4 Public IP**. You'll use this to login to your instance.

SSH requires that your private key be marked as read-only to its owner — that would be you — with no access to anyone else. If the file has any other protection set, SSH will refuse to use it. In Terminal, enter following command set protect your private key:

```
chmod 600 /path/to/your/ssh/key
```

> Generally, SSH keys and other related files should be in the hidden directory **~/.ssh**. If you didn't put your key there, please consider doing so before setting its protection.

Now, in Terminal, enter the following command:

```
ssh -i /location/to/your/ssh/key ubuntu@your-aws-ip
```

This will log you in and take you to a shell prompt in your instance.

To simplify accessing your instance, you can create an entry for it in **~/.ssh/config**. Use your favorite text editor — nano, vi, Sublime Text are all good choices — to add the following to that file:

```
Host vapor-til
    HostName <your public IP or public DNS name>
    User ubuntu
    IdentityFile </path/to/your/key/file>
```

Now, you can connect to your instance by entering the following command in Terminal:

```
ssh vapor-til
```

Many of the commands in rest of this chapter require root access. To get root access, enter the following command:

```
sudo su -
```

> The following commands all assume you are logged in to your EC2 instance and have root access.

On a new system, it's always a good idea to make sure all packages are up to date. To update your system, enter the following commands:

```
apt-get update
apt-get upgrade
```

Install Swift

To build your Vapor app, you must install Swift on your EC2 instance. The easiest way to install and update Swift on Ubuntu is to use the Vapor APT repository.

What is APT?

APT (Advanced Package Tool) is the default package manager for Debian-based Linux systems, like Ubuntu. The Vapor team maintains an APT repository for Swift and the Vapor Toolbox.

Install Swift from APT

To install Swift from the Vapor APT, enter the following commands:

```
# 1
wget -q https://repo.vapor.codes/apt/keyring.gpg -O- | \
  apt-key add -
# 2
echo "deb https://repo.vapor.codes/apt $(lsb_release -sc) main"\
  | tee /etc/apt/sources.list.d/vapor.list
# 3
apt-get update
# 4
apt-get install swift ctls
```

s

Here's what this does:

1. Add the signing key from the Vapor APT repository. This allows apt-get to verify the packages it downloads from the repository.
2. Add the correct APT repository for your Ubuntu version.
3. Update APT's local cache.
4. Install Swift and the SSL C bindings.

You can verify your installation by entering:

```
swift --version
```

Setting up your application

To set up your application, you will first clone it from GitHub.

When building and running your app, you should not be root. To downgrade to normal user access and build the TILapp example from the rest of the book, enter the following commands:

```
# 1
exit
# 2
git clone https://github.com/raywenderlich/vapor-til.git
# 3
cd vapor-til
# 4
swift build -c release
```

Here's what this does:

1. Downgrade to the user `ubuntu`.
2. Clone the vapor-til project from GitHub.
3. Change to the vapor-til folder.
4. Build the project in release mode.

After building the project, you can try to start the app by entering:

```
./.build/release/Run
```

This won't work because you haven't set up a database or the necessary environment variables.

Setting up a PostgreSQL server

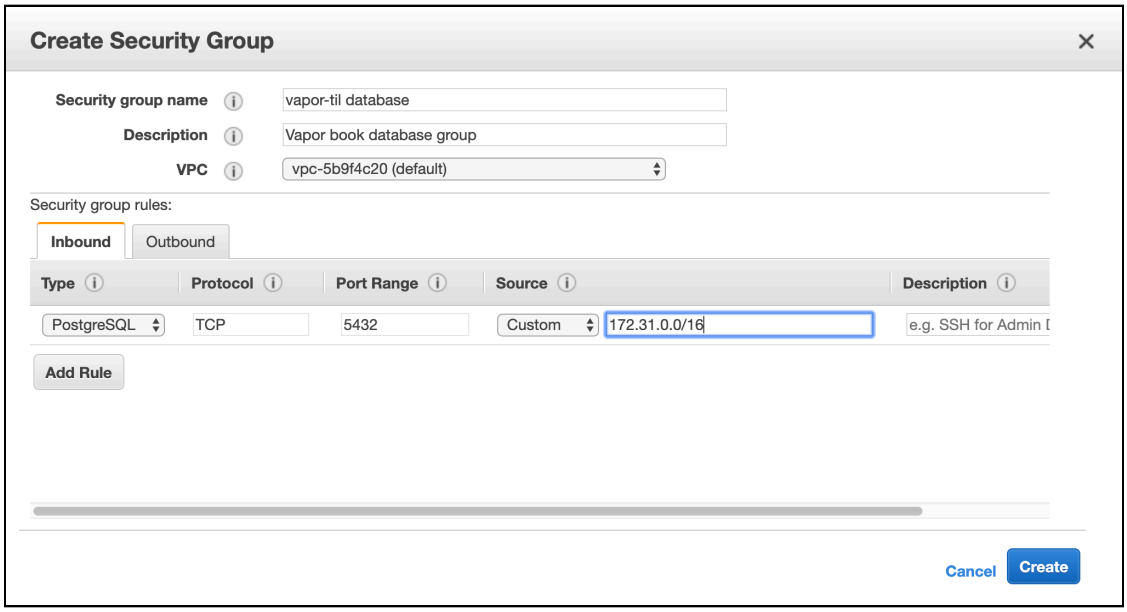
For your database, you will use Amazon Relational Database Service (RDS). This AWS database service is based on AWS S3 and can emulate a number of popular relational database systems including PostgreSQL.

Before creating your database, you need to configure another security group. Click **Services** at the top of your AWS page and enter **VPC** in the search bar. This will take you to the VPC dashboard. Click **Your VPCs** to show your VPC (Virtual Private Cloud) information. In the description section, make a note of your **IPv4 CIDR**. It will be something like **172.31.0.0/16**.

Now, return to your EC2 Dashboard and click **Security Groups** in the list on the left. The resulting screen should now look familiar. Click **Create Security Group**. Name the group **vapor-til database**.

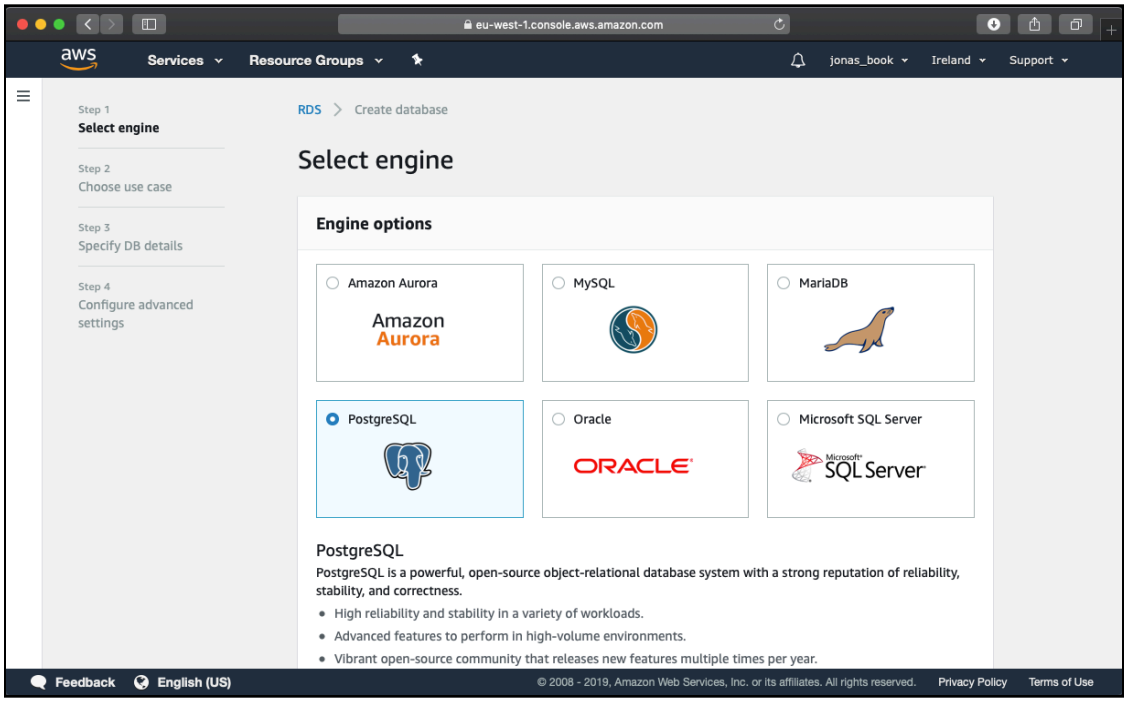
On the **Inbound** tab, click **Add Rule**. Select **PostgreSQL** from the **Type** dropdown and enter your IPv4 CIDR in the **Source** box.

Your screen should look something like this:

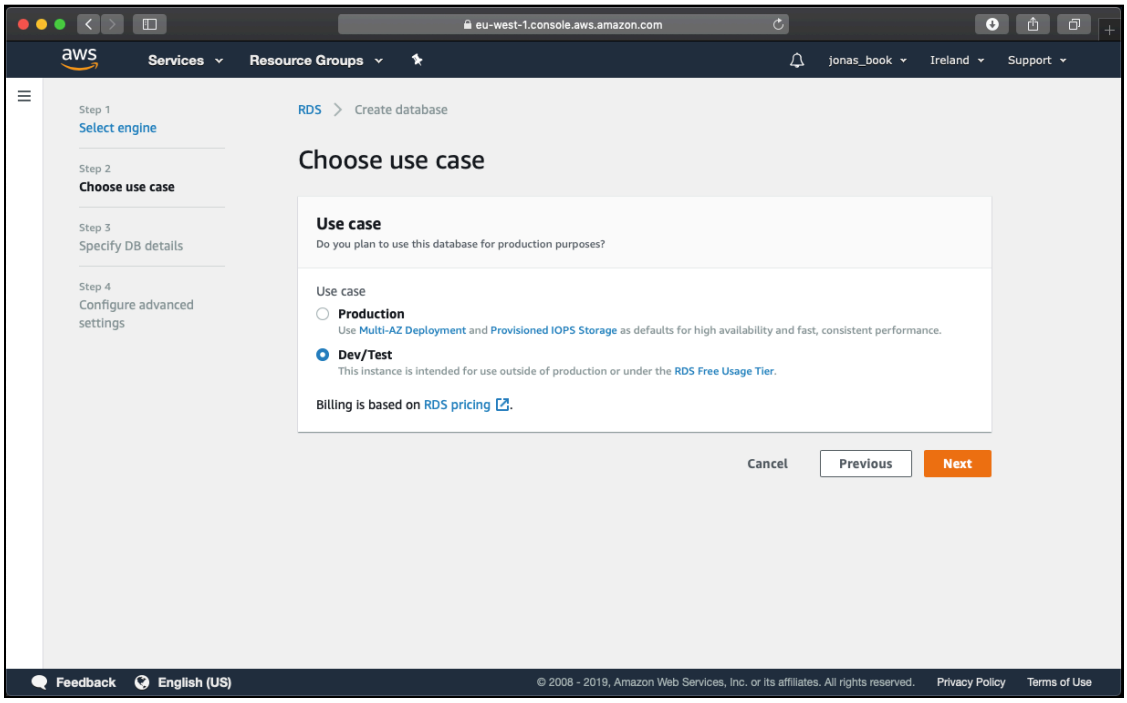


Click **Create**.

In the AWS Console, click **Services** and find **RDS**. Click **Create Database**. This will display the **Select engine** page. Choose **PostgreSQL** as your engine and click **Next**.



Next, you must choose your use case. For this tutorial, select **Dev/Test**. Check **Next**.



Now, you're asked to specify some details about your database. Under **Instance specifications**, enter the following information:

- **License model**: postgresql-license
- **DB engine version**: PostgreSQL 10.6-R1 (any version 10 is fine)
- **DB instance class**: db.t2.micro
- **Multi-AZ deployment**: No
- **Storage type**: General Purpose (SSD)
- **Allocated storage**: 20

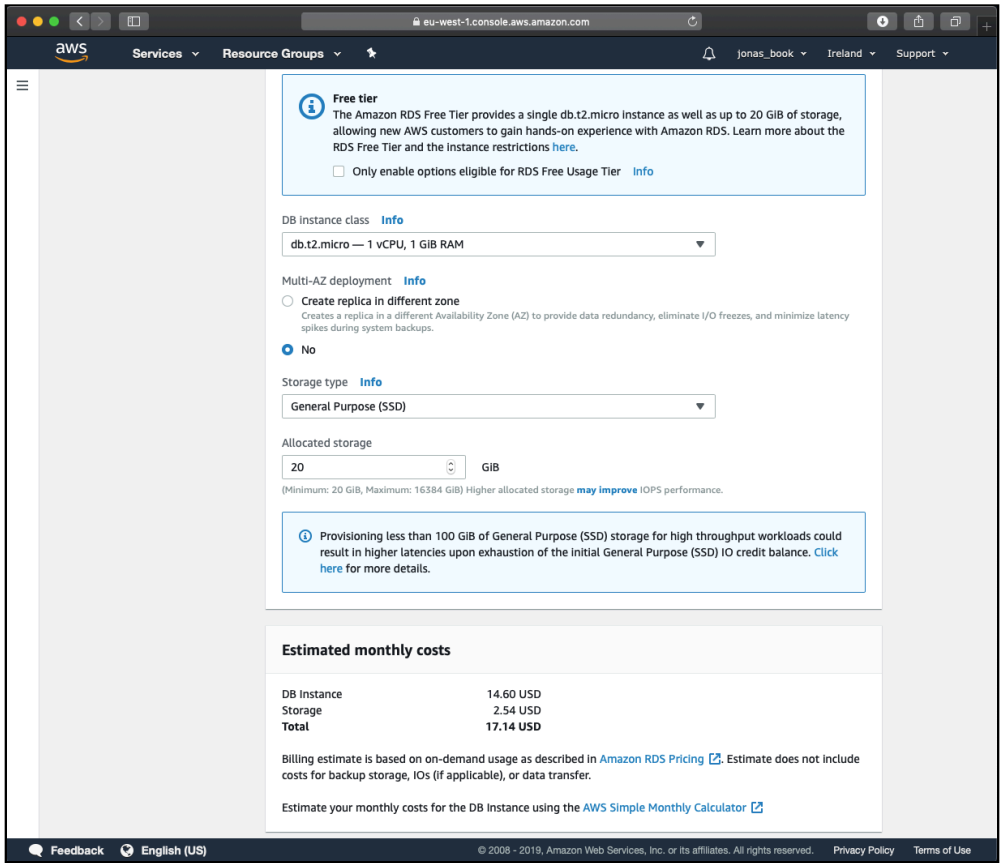
> While **db.t2.micro** is free tier eligible, if your account no longer qualifies for free tier access, completing this chapter *will* cost you money. Be sure to stop or terminate your EC2 and RDS instances when you're done to avoid continuing charges.

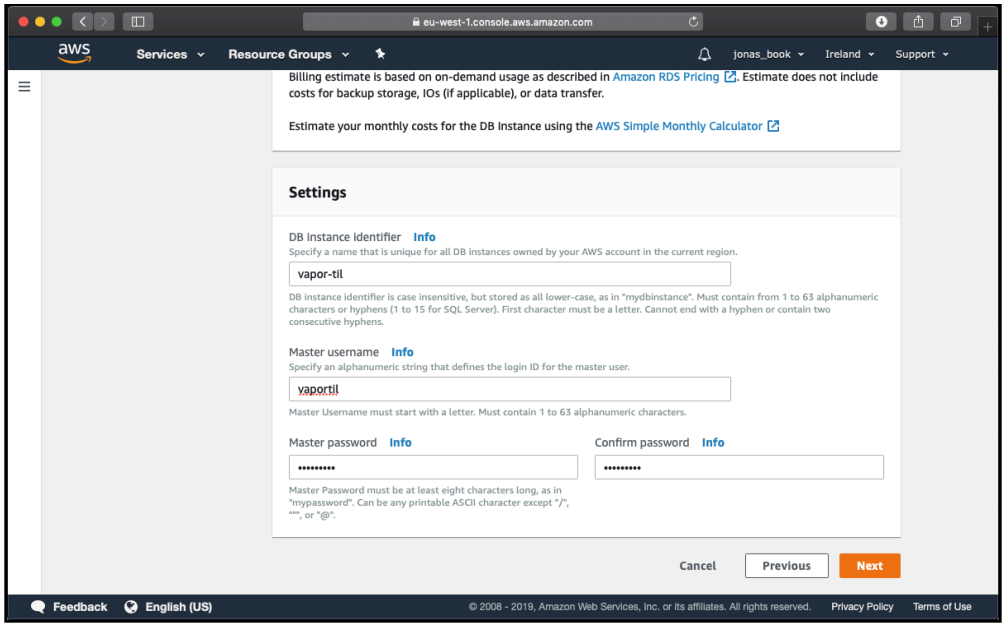
Under **Settings**, enter the following information:

- **DB instance identifier**: vapor-til
- **Master username**: vaportil

- **Master password** and **Confirm password**: your choice

Click **Next**.





Under **Network & Security**, set **Public accessibility** to **Yes**. This will allow you to access the database from your local machine, should you so desire.

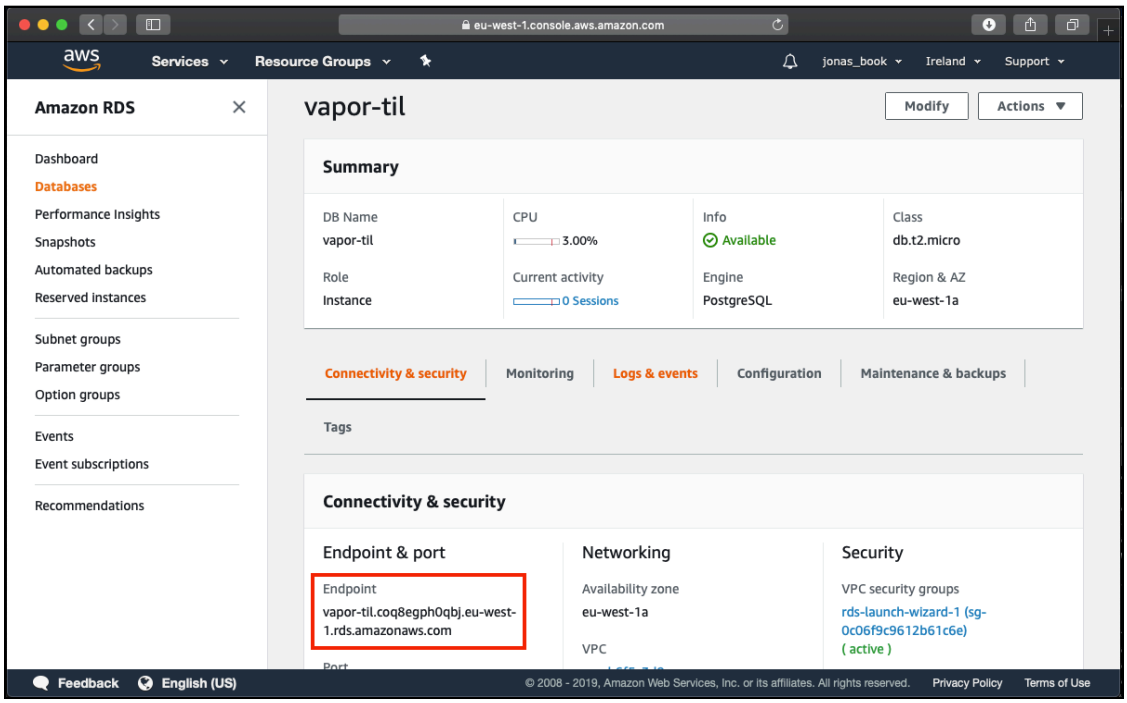
> If you do wish to access your database from your local machine, you'll need to add a rule to your security group to permit the access.

Set **VPC security groups** to **Choose existing VPC security groups**. Click the **X** next to **Default** to remove that group and add **vapor-til database** from the dropdown. Leave the other settings at their defaults.

Finally, under **Database options**, enter **vaportil** as the **Database name**.

Scroll to the bottom of the page and click **Create database**. It will take a some time for this to complete. Click **View DB instance details**.

Under the details of the RDS instance, find the **Endpoint** in the **Connectivity & security** section and make a note of it. You'll need it shortly.



Installing and configuring nginx

nginx is a popular web server, typically used as a proxy server in front of other web apps. For Vapor apps, this is useful because it provides additional features such as compression, caching, HTTP/2 support, TLS (HTTPS) and more.

ruThe example here is very simple and just gets you going. However, it is easy to customize to allow for more features.

Begin by installing nginx on your EC2 instance. SSH into your EC2 instance and enter the following commands:

```
sudo su -
apt-get install nginx
```

For setting up nginx config, create a file in **/etc/nginx/sites-available** called **vapor-til** and add the following content to it with your favorite editor:

```
server {
  listen 80;

  root /home/ubuntu/vapor-til/Public;
  try_files $uri @proxy;

  location @proxy {
    proxy_pass http://localhost:8080;
    proxy_pass_header Server;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_connect_timeout 3s;
    proxy_read_timeout 10s;
  }
}
```

Then run the following commands

```
# 1
rm /etc/nginx/sites-enabled/default
# 2
ln -s /etc/nginx/sites-available/vapor-til \
  /etc/nginx/sites-enabled/vapor-til
# 3
systemctl reload nginx
```

Here's what this does

1. Disable the default site.
2. Enable your vapor-til site.
3. Reload nginx to activate your changes.

Before you can test all of this, you need a way to start your app.

Running your app as a system service

You want your app to run when your instance boots and to restart if it crashes due to a critical error. The easiest way to accomplish this is to integrate it as a system service. The versions of Linux that Swift — and, therefore, Vapor — supports use a service called `systemd` to accomplish this.

You'll need to add two files to your running system: One which contains all the environment variables your app needs and one which defines your app's service.

> You can do it all in one file but this division makes it simpler to adjust environment variables should that become necessary.

SSH to your EC2 instance and become root by entering:

```
sudo su -
```

Now, use your favorite editor to create **/etc/vapor-til.conf** and add the following to it:

```
DATABASE_HOSTNAME='<your AWS RDS endpoint>'
DATABASE_USER='vaportil'
DATABASE_DB='vaportil'
DATABASE_PASSWORD='<your chosen password>'
SENDGRID_API_KEY='test'
GOOGLE_CALLBACK_URL='test'
GOOGLE_CLIENT_ID='test'
GOOGLE_CLIENT_SECRET='test'
GITHUB_CALLBACK_URL='test'
GITHUB_CLIENT_ID='test'
GITHUB_CLIENT_SECRET='test'
```

This sets the environment variables that TILapp uses to find its database and to integrate with other services. They are identical to the environment you've been creating in Xcode in other chapters.

> To have all of the pieces of TILapp working correctly, you'll need to substitute valid values for all of the SENDGRID, GOOGLE and GITHUB values. See chapters 22–24 for how to configure these. For now, any non-empty string will allow the app to run.

Next, use your favorite editor to create **/etc/systemd/system/vapor-til.service** and add the following to it:

```
# 1
[Unit]
```

```
Description="Vapor TILapp"
After=network.target

# 2
[Service]
User=ubuntu
EnvironmentFile=/etc/vapor-til.conf
WorkingDirectory=/home/ubuntu/vapor-til
# 3
Restart=always
# 4
ExecStart=/home/ubuntu/vapor-til/.build/release/Run \
  --env production

[Install]
WantedBy=multi-user.target
```

Here's what this does:

1. `systemd` refers to an item it manages as a **unit**. This section defines the unit for your app and specifies that it can't start until after the network is started.
2. Specify the parameters for your app's service. You can have the app run as any valid user. Notice that you use the configuration file you created earlier to describe the environment in this section.
3. Tell `systemd` that it should always attempt to restart your app if it fails.
4. Specify the command line that `systemd` will execute to start your app. You can add additional arguments here should that be necessary.

After saving the changes to the service definition file, you must tell `systemd` to read it in order to have it recognize the service. Enter the following command:

```
systemctl daemon-reload
```

This will make `vapor-til.service` available as an option. With the following commands, you'll discover that standard keyboard shortcuts, such as tab completion, work with your new service just as they do with existing system services.

To start your app and enable it to start automatically after a reboot, enter the following commands:

```
systemctl start vapor-til.service
systemctl enable vapor-til.service
```

Your app should launch. You can check its status by entering:

```
systemctl status -l vapor-til.service
```

Once your app is running, you should be able to access it by entering your EC2 instance's Public DNS name (the same one you use for SSH) into your browser.

Should you need to restart your app manually, use the following command:

```
systemctl restart vapor-til.service
```

And, if you wish to stop your app, the following command does the trick:

```
systemctl stop vapor-til.service
```

Where to go from here?

You now have the basics of how to set up a Vapor app on AWS. There are many more things AWS allows, such as scaling, IP pooling, automatic backups, replication and so on. Spend some time with the AWS documentation and tutorials to learn more.

When you're finished with your EC2 instance and RDS database from this chapter, be sure to **Delete** the database instance and **Terminate** the EC2 instance so AWS will delete them and you avoid paying any (additional) charges for them.