

# Deploy an Express App to AWS Elastic Compute Cloud (EC2)

---

## Prepare Your Local Code for Deployment

*In this section, you'll set up Express to serve your HTML file.*

### Prerequisites

These directions assume that you have a folder containing files for a single webpage, namely an `index.html` file, a `styles.css` file, and a `main.js` file.

- **Note:** The file names do not have to be the exact same for every project, but they will be referred to by these names in this document.

### npm & git

In order to serve your HTML page, you'll want to install `express`. To do that, you'll need to initialize `npm` in the project and use it to install `express`.

1. In the terminal, `cd` into your project directory
2. Run `npm init -y`, which will set up a `package.json` file with the default settings
3. Run `npm i express` to get the `express` package installed to this project

Now you'll want to set up `git` so that you can track changes. We'll connect your local repo to a remote on GitHub so you'll be able to share your local code with your remote server.

1. Run `git init` to create a local repo
2. Add a `.gitignore` file and add `/node_modules` to the file so they don't get pushed
3. Run `git add .` to stage all your files for commit
4. Run `git commit -m "initial commit"` to create a commit
5. On GitHub, create a new repository
6. Copy the commands given for existing repositories
7. Paste the commands in your terminal

## Express

You have used `express` to set up API endpoints. You'll use it here again to serve your HTML page. Recall that servers wait for requests and then send responses. Those requests come by URL and then the server can send back whatever information was requested whether that's JSON or HTML.

1. Ensure that your HTML, CSS, and JavaScript files are in a folder titled `public`
  - a. This is a common pattern and it makes it easier to differentiate between your front-end and your server
2. At the root of your project directory, create a file named `server.js`
3. In the `server.js` file, import and set up express to listen for requests with the following 3 lines:
  - a. `const express = require('express')`
  - b. `const app = express()`
  - c. `app.listen(4000, () => console.log(`server running on 4000`))`
    - i. Refer back to the Backend lectures from week 4 if you're not sure what is happening here
4. Now comes the new stuff! There are different methods for serving static files from express, but we'll be using middleware since it simplifies the code a bit. Recall that middleware are just functions that run in between the request and the response. You'll use "top-level" middleware, which means that it'll run on every request made to the server.
5. Add the following line to your main server file and then read on for an explanation:
  - a. `app.use(express.static(`${__dirname}/public`))`

|                             |  |
|-----------------------------|--|
| <code>app.use</code>        | This indicates the use of middleware, this should be at the top/root/global level of your main server file since you'll want it to run on every request.                 |
| <code>express.static</code> | This is a built-in express method that expects a file path.  |
| <code>`\${...}`</code>      | This is a template string; in this case, it helps simplify the code's readability and eliminates the need to import the path variable.                                   |
| <code>__dirname</code>      | This is sort of like inserting the result of running <code>pwd</code> in your command line – it finds the full path of your server file. This is useful for running your |

|                      |   |
|----------------------|---|
|                      | server in multiple places (like your local computer <i>and</i> your deployment server) since the absolute path will change based on what system you're on.  |
| <code>/public</code> | <p>This is the path to your main front-end folder from your server file. Express will look for an <code>index.html</code> file in here to serve to the base ("<code>/</code>") endpoint.</p> <p><b>IMPORTANT: How will the CSS and JS files work?</b></p> <p>When express is serving your HTML in this way, it reads file paths as relative to your public folder. So, when you have <code>src="main.js"</code> in a script tag, Express will look for a file named <code>main.js</code> in the <code>public</code> folder.</p> |

## Test Locally

Now that you have your Express server set up to serve your new build folder, try it out locally!

1. Make sure you stop any current `nodemon` servers
2. Run `nodemon /your/filepath`
3. Open up the browser to the port that your Express server is running on – your app should be running there!

## GitHub

If everything is working go ahead and push your code to GitHub!

# Set Up An AWS Account (10 steps)

1. Head to the [AWS Homepage](#)
2. Click the orange “Create an AWS Account” in the top right corner
  - a. **Note:** you will need a credit card to sign up for your account but will be able to set up and run a server for free
  - b. Please view pricing information [here](#) to ensure that you don’t exceed the free limits
  - c. Many AWS Free Tier services are always free, some are available for a year free of charge, some charge after a certain amount of usage
3. Fill out the fields for your email address and account name (you can change these later)



## Explore Free Tier products with a new AWS account.

To learn more, visit [aws.amazon.com/free](https://aws.amazon.com/free).



## Sign up for AWS

### Root user email address

Used for account recovery and some administrative functions

### AWS account name

Choose a name for your account. You can change this name in your account settings after you sign up.

Verify email address

OR

Sign in to an existing AWS account

4. You'll then get a verification code sent to your email, enter it on the next page.
5. Now you'll be on the password page, create a password and click continue



### Explore Free Tier products with a new AWS account.

To learn more, visit [aws.amazon.com/free](https://aws.amazon.com/free).



## Sign up for AWS

### Create your password

✔ It's you! Your email address has been successfully verified. ✕

Your password provides you with sign in access to AWS, so it's important we get it right.

Root user password

Confirm root user password

**Continue (step 1 of 5)**

6. The next page will be all of your contact information - fill that out and continue
  - a. We recommend that you create a Personal account
7. Then you'll be moved to the Billing Information page - fill this out and continue. A note from the beginning of this document is here again as a reminder:
  - a. **Note:** you will need a credit card to sign up for your account but we will only use free services, many AWS Free Tier services are always free, some are available for a year free of charge, some charge after a certain amount of usage
8. Now you'll need to verify your identity, you'll just need a phone number for this step.

9. And finally, make sure you're choosing the free option and click "Complete sign up"



## Sign up for AWS

### Select a support plan

Choose a support plan for your business or personal account. [Compare plans and pricing examples](#)  
[↗](#). You can change your plan anytime in the AWS Management Console.

#### ☒ Basic support - Free

- Recommended for new users just getting started with AWS
- 24x7 self-service access to AWS resources
- For account and billing issues only
- Access to Personal Health Dashboard & Trusted Advisor



#### ☐ Developer support - From \$29/month

- Recommended for developers experimenting with AWS
- Email access to AWS Support during business hours
- 12 (business)-hour response times



#### ☐ Business support - From \$100/month

- Recommended for running production workloads on AWS
- 24x7 tech support via email, phone, and chat
- 1-hour response times
- Full set of Trusted Advisor best-practice recommendations



#### Need Enterprise level support?

From \$15,000 a month you will receive 15-minute response times and concierge-style experience with an assigned Technical Account Manager. [Learn more](#) [↗](#)

[Complete sign up](#)

10. In the middle of the page, or in the upper right corner, there should be a button that says “Sign into management console” or something similar – click that button and use the information you just set up to sign in as a Root user



## Sign in

☒ **Root user**

Account owner that performs tasks requiring unrestricted access. [Learn more](#)

☐ **IAM user**

User within an account that performs daily tasks. [Learn more](#)

### Root user email address

username@example.com

Next

By continuing, you agree to the [AWS Customer Agreement](#) or other agreement for AWS services, and the [Privacy Notice](#). This site uses essential cookies. See our [Cookie Notice](#) for more information.

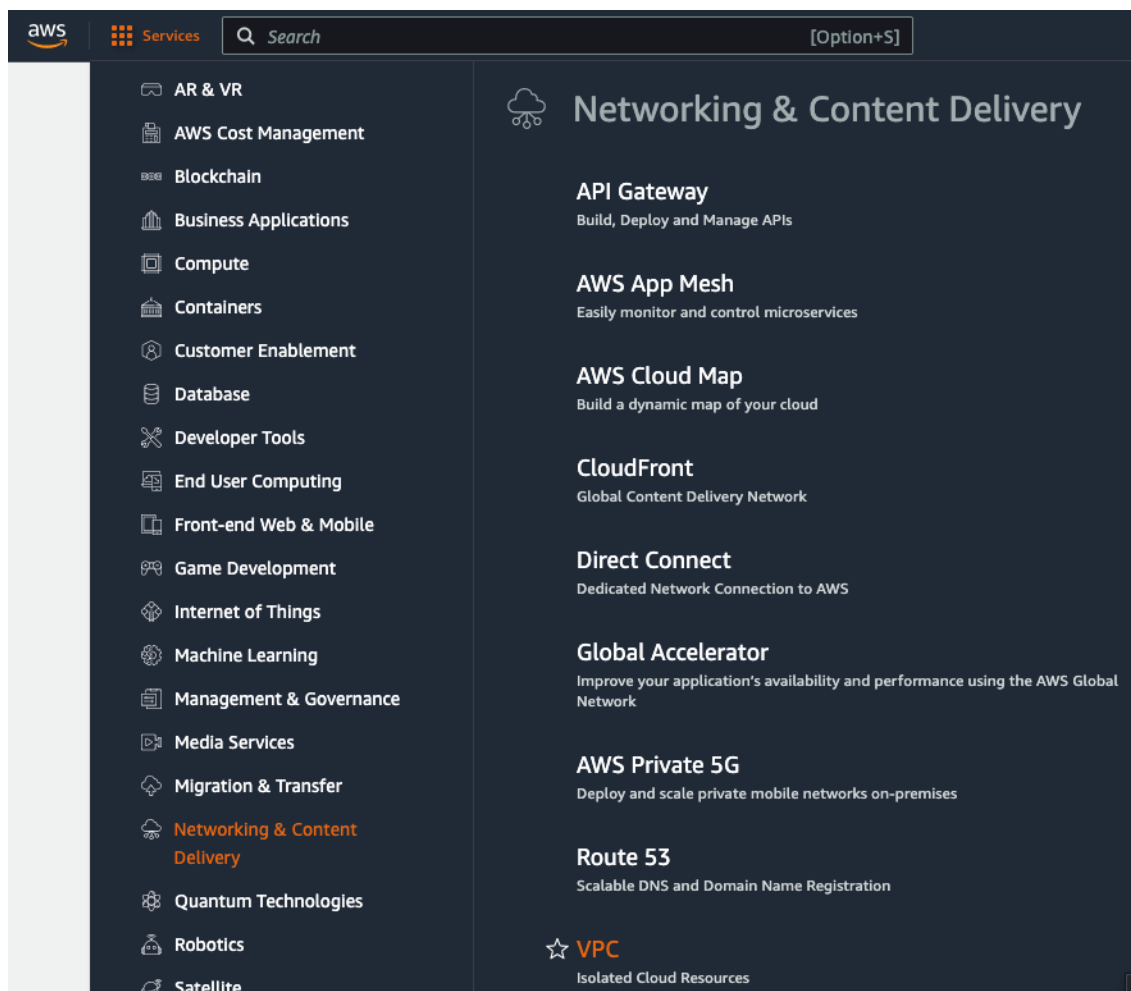
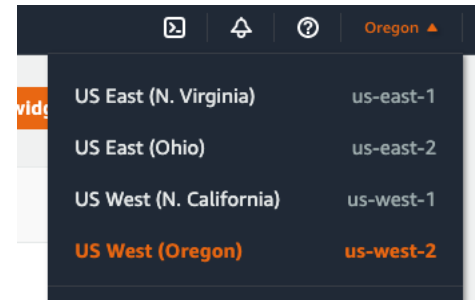
\_\_\_\_ New to AWS? \_\_\_\_

Create a new AWS account

## Set Up a Security Group

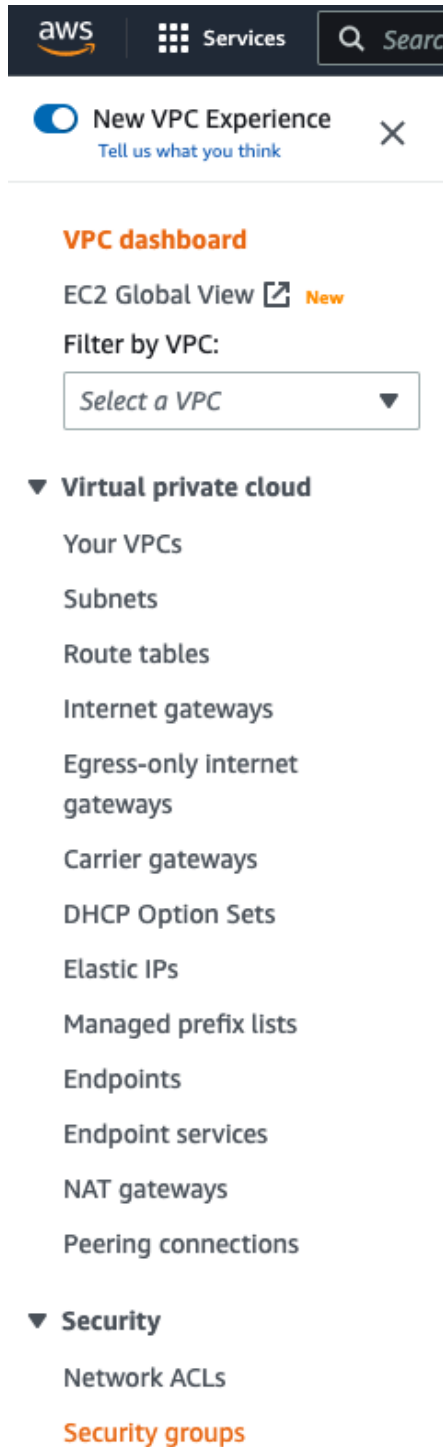
*The AWS service(s) we'll be using require a security group. We'll set up a development one now that you can use on all of your student projects. Security groups are used on Amazon services to define who can and can't access the services. Sometimes this means developers and teams and sometimes it means applications that should be available to the whole internet.*

- On the AWS Console Home page, ensure you're in the region that you would like your services to run in. You can view the region in the upper right on the nav bar. To make things easy, we recommend that you choose the one closest to you. The region is important because the services we'll use are tied to the region. So if you make a server in the Oregon region and then you're viewing the Ohio region later, you won't be able to see your Oregon server.
- Then in the upper left, click on Services. On the left-hand menu, click "Networking & Content Delivery" and then on the right side, click "VPC". (You could also search for "VPC" in the search bar.)
  - VPC** stands for "Virtual Private Cloud" – it's your own virtual network! One is automatically created for you when you make an account.





3. On the left side of the VPC dashboard, under “Security”, click “Security groups.”




The screenshot shows the AWS VPC dashboard. At the top, there is a navigation bar with the AWS logo, a 'Services' menu, and a search bar. Below the navigation bar, there is a 'New VPC Experience' banner with a 'Tell us what you think' link and a close button. The main content area is titled 'VPC dashboard' and includes a link to 'EC2 Global View' with a 'New' tag. A 'Filter by VPC:' dropdown menu is set to 'Select a VPC'. The left sidebar is expanded, showing a list of VPC resources under the 'Virtual private cloud' section and a 'Security' section. The 'Security groups' link is highlighted in orange.

aws Services Search

New VPC Experience  
Tell us what you think

**VPC dashboard**

EC2 Global View  New

Filter by VPC:

Select a VPC ▼

▼ **Virtual private cloud**

- Your VPCs
- Subnets
- Route tables
- Internet gateways
- Egress-only internet gateways
- Carrier gateways
- DHCP Option Sets
- Elastic IPs
- Managed prefix lists
- Endpoints
- Endpoint services
- NAT gateways
- Peering connections

▼ **Security**

- Network ACLs
- Security groups**

4. In the top right corner, click the orange "Create security group" button.
5. Fill out the basic details. Leave the VPC as is - this is the default VPC created for you when you made your account

### Basic details

Security group name [Info](#)

Name cannot be edited after creation.

Description [Info](#)

VPC [Info](#)

6. Then under "Inbound rules", you'll create 3 rules. You'll be letting traffic in from anywhere since this will be a hosted website and should be accessible to the internet. You'll also add your specific IP as a backup so that it's always allowed.

| Rule | Type        | Source        |
|------|-------------|---------------|
| 1    | All traffic | Anywhere-IPv4 |
| 2    | All traffic | Anywhere-IPv6 |
| 3    | All traffic | My IP         |

### Inbound rules [Info](#)

| Type <a href="#">Info</a> | Protocol <a href="#">Info</a> | Port range <a href="#">Info</a> | Source <a href="#">Info</a>                                 |
|---------------------------|-------------------------------|---------------------------------|---|
| All traffic               | All                           | All                             | Anywhere-I...<br>0.0.0.0/0                                  |
| All traffic               | All                           | All                             | Anywhere-I...<br>::/0                                       |
| All traffic               | All                           | All                             | Custom<br>Custom<br>Anywhere-IPv4<br>Anywhere-IPv6<br>My IP |

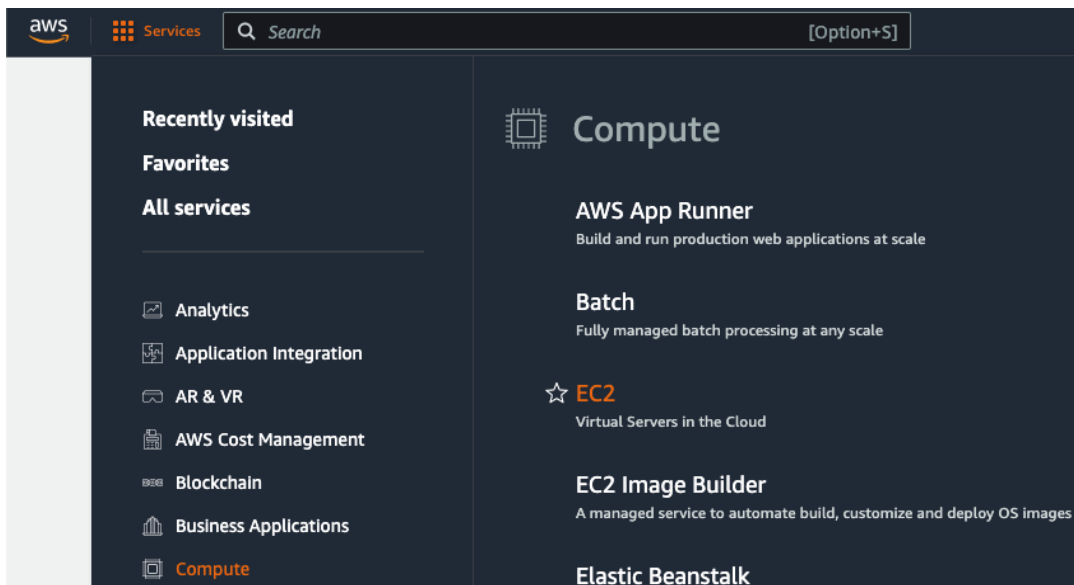
Add rule

7. Scroll down to the end and click the orange "Create security group" button in the bottom right.

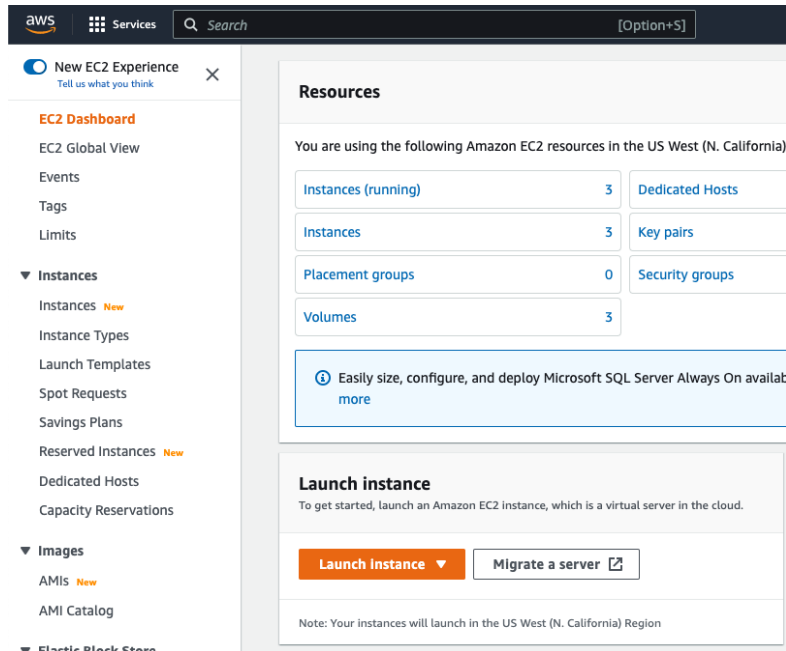
## Set Up an EC2 Instance

*In this section, you'll prepare an EC2 (Elastic Compute Cloud) instance and ensure that you can connect to it. An EC2 instance is a virtual server that's hosted in a cloud. Imagine a big data center filled with lots of computers, that's the cloud. Each of these computers can run multiple services (kind of like having a PC that can dual-boot into Windows or a Linux distro). And the virtual server is an environment running on one of those computers. So basically you'll be interacting with a remote virtualized computer through the command line.*

1. Ensure that you're in the correct region in the top right of the nav bar. You should be in the same region where you created a security group previously.
2. In the [AWS Console](#), navigate to the EC2 dashboard by clicking "Services" > "Compute" > "EC2" or by searching "EC2" in the search bar.

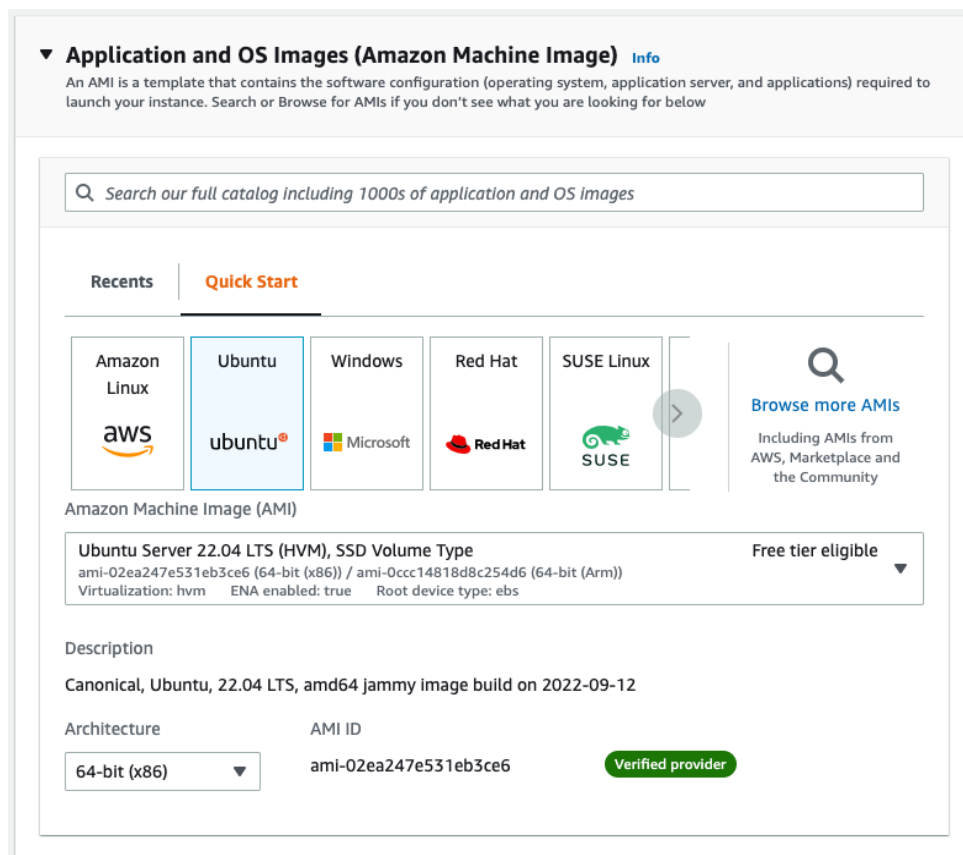


- Part way down the page, click the orange “Launch instance” button



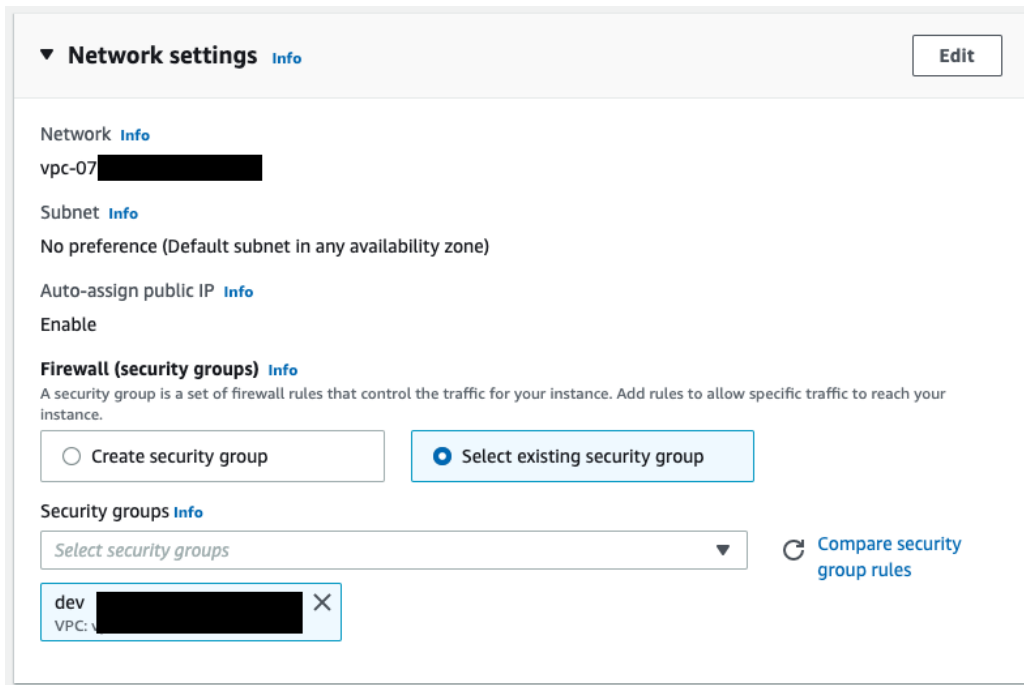
The screenshot shows the AWS Management Console interface. On the left, there's a navigation menu with options like 'EC2 Dashboard', 'Instances', 'Images', and 'Elastic Block Store'. The main content area is titled 'Resources' and shows a summary of EC2 resources in the US West (N. California) region. Below this, there's a 'Launch instance' section with a prominent orange 'Launch instance' button and a 'Migrate a server' button. A note at the bottom states: 'Note: Your instances will launch in the US West (N. California) Region'.

- In the Name and tags section, give your instance a name. This could be the same name as the app you'll be hosting or something more general.
- In the Application and OS Images section, select “Ubuntu” and then ensure that you've chosen **Ubuntu Server 22.x**. You can leave the Architecture drop down as 64-bit.



The screenshot shows the 'Application and OS Images (Amazon Machine Image)' section. It includes a search bar and tabs for 'Recents' and 'Quick Start'. Under 'Quick Start', there are cards for various operating systems: Amazon Linux, Ubuntu, Windows, Red Hat, and SUSE Linux. The 'Ubuntu' card is selected. Below the cards, the details for the 'Ubuntu Server 22.04 LTS (HVM), SSD Volume Type' AMI are displayed, including the AMI ID (ami-02ea247e531eb3ce6) and a 'Verified provider' badge. The architecture is set to '64-bit (x86)'.

6. Leave the Instance type as t2.micro – this is the biggest type that's free tier eligible and will work well for student projects.
7. For the Key pair, select "Proceed without a key pair" from the drop down. If you'd like to learn more about using a key pair for authentication, you can head to the [AWS Docs](#).
8. In the Network settings under Firewall, choose "Select existing security group" and then choose your security group from the drop down. The example group is named "dev" but you may have named yours differently.



**▼ Network settings** [Info](#) Edit

Network [Info](#)  
vpc-07 [REDACTED]

Subnet [Info](#)  
No preference (Default subnet in any availability zone)

Auto-assign public IP [Info](#)  
Enable

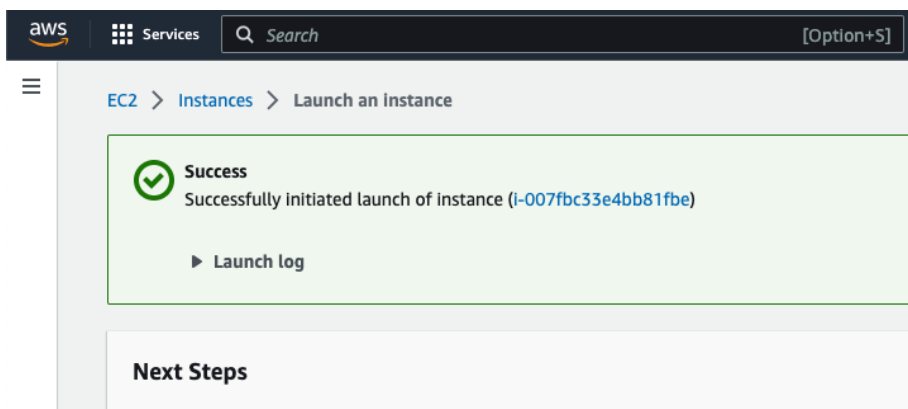
**Firewall (security groups)** [Info](#)  
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

☐ Create security group ☒ Select existing security group

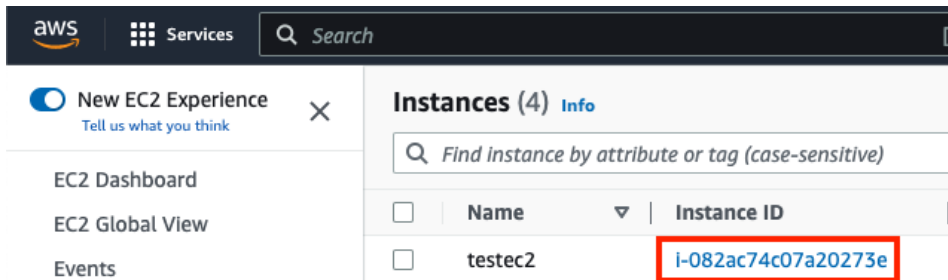
Security groups [Info](#)  
Select security groups ▼ Compare security group rules

dev [REDACTED] X  
VPC: [REDACTED]

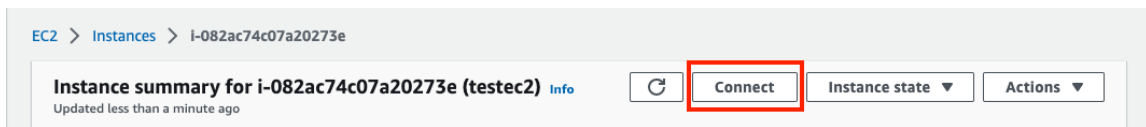
9. Leave the Configure storage and Advanced details sections as is.
10. Click the orange "Launch instance" button in the Summary section (either at the very bottom or on the right hand side in a menu, depending on your screen size).
11. You should be taken to a confirmation page that looks like this:



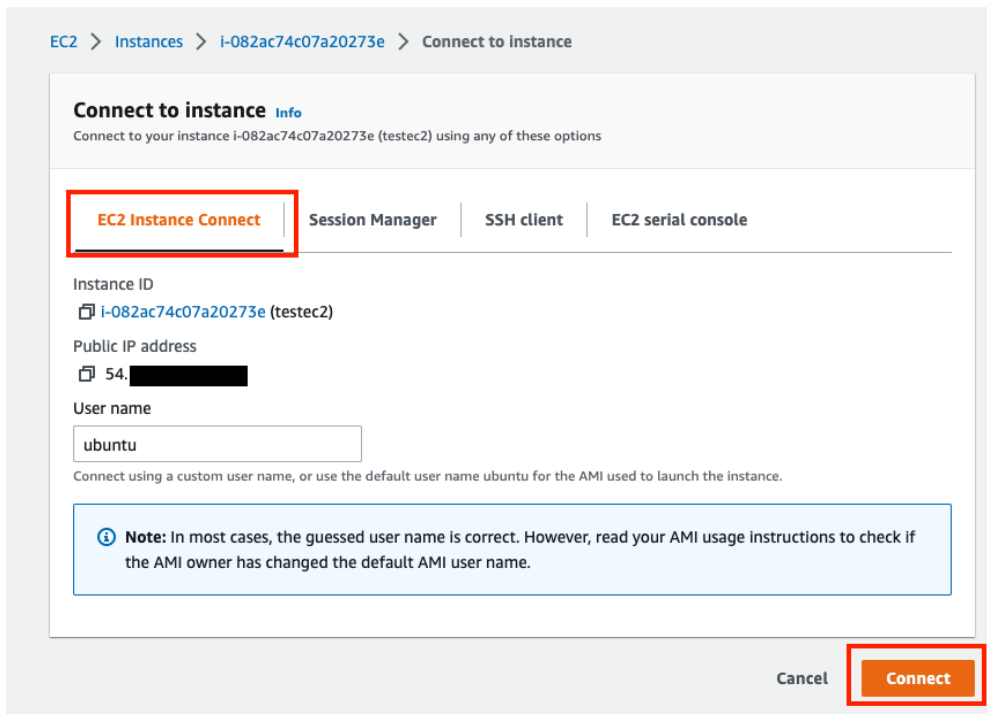
12. Click the “Instances” link in the breadcrumb nav (after “EC2”, before “Launch an instance”), which should take you back to a table with your new instance listed in it.
13. Click the “Instance ID” link in the table to see the information and settings for your new instance:



14. On the instance page, you can explore all sorts of settings. There’s nothing you need to change right now, so let’s go ahead and test the connection. Click the “Connect” button in the upper right. **You may need to wait a few minutes for this button to be enabled.**



15. This should bring you to a page that looks like the screenshot below. Ensure that you’re in the “EC2 Instance Connect tab” and click the orange “Connect” button in the bottom right.



16. If all goes well, you’ll be brought to a terminal window within the AWS console! This is where you’ll be interacting with your server.

## Quick Recap on Connecting to Your Instance

*Here's a reminder on how you'll get back to your instance connection.*

1. In the AWS Console, get to the EC2 Dashboard from the "Services" menu or the search bar.
2. Click "Instances" in the left menu.
3. Click the "Instance ID" link of the instance you want to connect to.
4. Click the white "Connect" button in the upper right of the instance detail page.
5. Click the orange "Connect" button on the connect page.

# Prepare Your Remote Server

*Congrats! You've (essentially) got yourself a new computer. Now you need to go through and set up everything similarly to what you did to your local machine at the beginning of Foundations.*

## A Note on `sudo`

All of the commands in this section are prepended with the word `sudo`. This is allowing you to run commands as an administrator. There are multiple ways to do this, as there are with most things in the tech world, and you can do it another way if you're comfortable with it. However, if you've never done this kind of thing before, stick with using `sudo`. It will keep your server a little safer since you won't be running as the root user and it will be easier for staff to help you debug since they'll know exactly what commands you've done. Onward!

## Installing Node and npm

*Remember when you were setting up your server and chose Ubuntu 22? Ubuntu is a Linux operating system and its terminal is Unix-based which is what Macs are based on too. Worry not, Windows users, it will feel familiar to you as well. This is great because the only way you'll be interacting with your server is in the command line. And the first thing you'll do there is install node and npm.*

1. Connect to your EC2 instance.
2. Run the following 9 commands one at a time in the order listed.
  - a. **Note:** `apt` is a command line utility used to manage packages on Ubuntu (and other Linux distributions) – you will use it to install `npm` and then use `npm` to install and manage other packages

| # | Command                      | Explanation  |
|---|------------------------------|--|
| 1 | <code>sudo apt update</code> | This will check for available updates and list them for you. It's a good idea to do this before upgrading to |



|   |   |   |
|---|---|---|
|   |   | check for any potential errors.   |
| 2 | <code>sudo apt upgrade</code>           | This will upgrade any packages that need it. This is done as a precautionary step.  |
| 3 | <code>sudo apt install nodejs -y</code> | This is where you're actually installing node! The "y" flag just auto-agrees to the default settings.   |
| 4 | <code>sudo apt install npm -y</code>    | And now you're installing npm.  |
| 5 | <code>sudo npm i -g n</code>            | This will install a package called "n" globally, which helps manage node.<br>(The expanded version of this command would look like this: <code>sudo npm install --global n</code> )   |
| 6 | <code>sudo n stable</code>              | This will update node to the latest stable release in case using <code>apt install</code> got an older version.   |
| 7 | <code>sudo npm i -g npm</code>          | This will update npm to match the version of node you have.   |
| 8 | <code>sudo npm i -g nodemon</code>      | This will install nodemon globally so you can use it to run projects while testing.   |
| 9 | <code>sudo npm i -g pm2</code>          | This will install a package called "pm2" which is similar to nodemon, but it can run apps indefinitely. If you tried to run your app with nodemon and closed out of your connection, the process would quit running and your website would be down. The pm2 processes will run in the background, even when you're not connected. |

3. You can check that everything is installed correctly by running the command with

`--version` after it.

- `node --version`
- `npm --version`
- `nodemon --version`
- `pm2 --version`

## Connect Your GitHub Account

*In this section, you'll create an SSH key pair on your server and save it to your GitHub account so that you can run remote git commands without having to enter your login credentials every time. SSH stands for "Secure Shell" and it's an encrypted protocol that allows you to connect to your server – that's what AWS is using when you click "Connect" and you get connected to your remote computer's terminal. Using SSH keygen, you'll create a pair of encrypted keys - one public and one private. You'll share the public key with GitHub and then when you attempt to run remote git commands, your server and GitHub will make sure the keys match up.*

1. Connect to your EC2 instance.
2. Run the following command: `ssh-keygen -t rsa`
  - a. The `-t` flag stands for type and `rsa` is the type – it's the acronym of the cryptography algorithm that will be used to generate the key. The algorithm's full name is Rivest-Shamir-Adleman.
3. You will be asked for a location and filename. Just press enter to use the defaults.
4. You will be asked to enter a password. It's an **invisible entry field, so you won't see the password or even placeholder characters**.
  - a. It cannot be recovered. Make sure it is something you can remember.
  - b. If you do forget it, or enter it incorrectly, you will have to delete this key pair and make a new one. You'll then have to replace everywhere that you use the key. This isn't a big deal for this use case, but it can be very annoying when you're using a key to authenticate multiple services.
5. Two files will then be created in the `.ssh` directory in the home folder of your computer (i.e., in `~/.ssh`). These files are `id_rsa` and `id_rsa.pub`, your private and public keys, respectively. You will notice a randomart image of your key like the one to the right.

```
Your identification has been saved in demo_rsa.
Your public key has been saved in demo_rsa.pub.
The key fingerprint is:
cc:28:30:44:01:41:98:cf:ae:b6:65:2a:f2:32:57:b5 user@user.local
The key's randomart image is:
+--[ RSA 2048 ]-----+
|      o=.             |
|    o  o++E          |
|  + . 0oo.           |
| + 0 B..             |
|  = *S.              |
|    o                |
+-----+
|
```

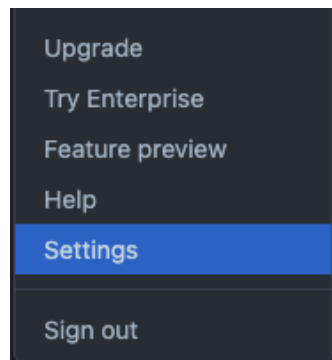
- Now you'll get the public key to give to GitHub. Enter the following commands one at a time in the order listed.

| # | Command                     | Explanation   |
|---|-----------------------------|---|
| 1 | <code>cd ~</code>           | Ensure you're at the root level of your user.   |
| 2 | <code>ls -a</code>          | List everything, including hidden files, that's stored here (the "a" flag stands for "all"). You should see <code>.ssh</code> here! If you don't, something might have gone wrong with your keygen process, please reach out in the queue for help. |
| 3 | <code>cd .ssh</code>        | Now change directories into the <code>.ssh</code> folder.   |
| 4 | <code>ls</code>             | See what's in this folder. You should have an <code>id_rsa.pub</code> file.   |
| 5 | <code>cat id_rsa.pub</code> | This will print the file to the console for you! Cat is short for concatenate.  |

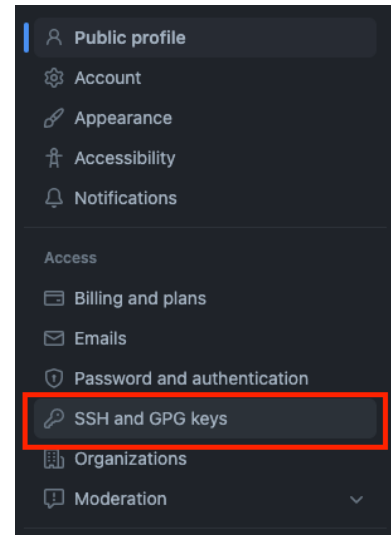
- Highlight and copy the **whole** key – you'll be entering it on GitHub momentarily.
  - Start copying at the `ssh-rsa` and end with the `user@servername`.

```
ubuntu@ip-172-31-13-16:~/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDAQDs6dk9/1EHnwEmkmVrYxJXxd6EtzdUmhZpRYGbsp4weIrmk+hZOpQGN+t+h
OVGrXuTU6HDGwlniGWyrMDJdyJE3fQqtegu4T3Q6Q+3uvqGSnCrFSc/YZzMuQE8HseuYF6FKcQETfXwtFxDpNEcanFz08k1lA
7PJfOmFHLfNK/+su870cBcsH13YvSulawGVHL7gm8/nDE6wphnEIwQhRzwyMKhBy3VJfKpdyg5gD/RsoFKGVXC1DTpm5bHk1
FHSkL1WgYtZh7Kd5ts5TpHbTwN0AkVR1iOHw8FPbwm9gkVRR3GfCVZQkI+JFyxcZxtjlI5tlnRmUYOHsLvENxm9pYRSa80bP7
00bcBazXdn0Qx05rfpFtq8n7KpSzStDQ60V5gfYY5MjGUALW745zZMhKR0yfkj5wxfVYq3BiQRzrK9PNY6YgAlgcsxEBdzBVR
Y/OvG51vkov2WGLGTFXPkU9HpAm15rMBRYxA40bvAQG+h9FLk/7D3f3Vsi30W19mQppn8rYNpKtODdBYI2V7lsLjt0weJk1JP
cfRLfw3M+qMP3juxujuVxVqv9rt8XdavtqqzdelpmBHugdKIrzCEUL09cBs/bp0Id5CUzYEZBpEJvK4dgexots1KG6/ebeWI
+ep1hdJlNBik8MORF+ZJ4dUjBp7EOQvTnL2rI03QkUQ== ubuntu@ip-172-31-13-16
ubuntu@ip-172-31-13-16:~/.ssh$
```

- Head to your GitHub account. Click the dropdown by your profile picture in the top right corner and choose "Settings."

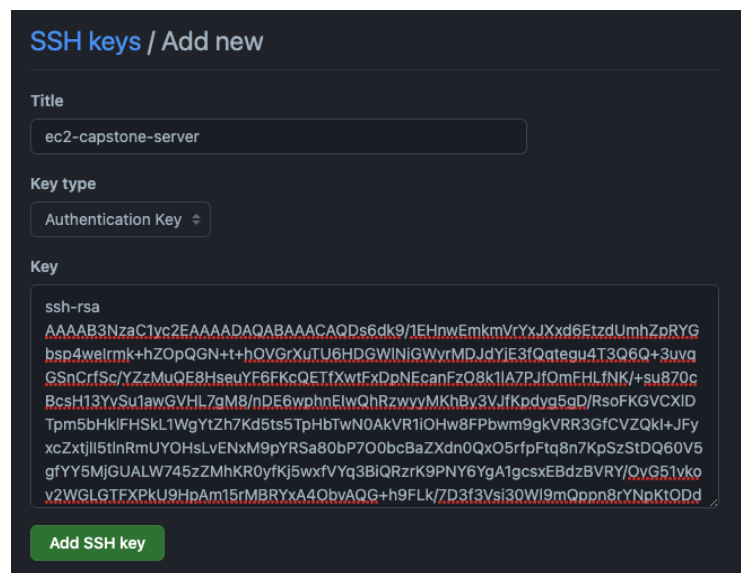


9. In the “Access” section on the left, click on “SSH and GPG keys.”



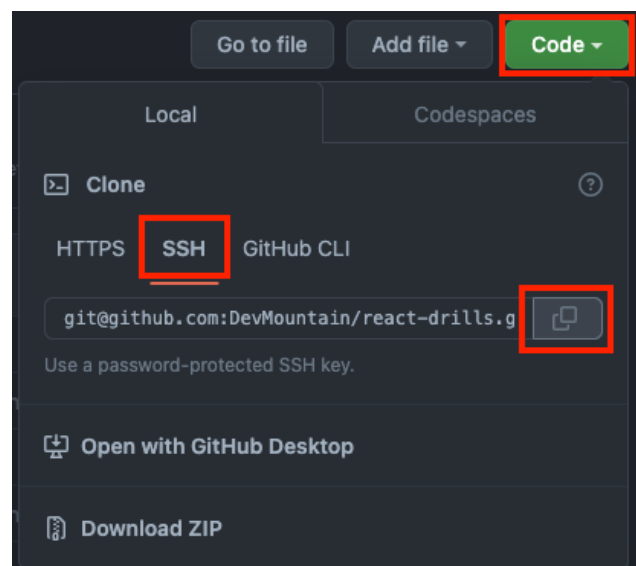
10. Click the green “New SSH key” button in the upper right.

11. In the form on the next page, give your key a title, ensure the type is “Authentication key”, and paste in your key from the server.



12. Now let's make sure that it works. To start that process, you'll need to get the SSH URL from your project repo.

- Head to your project's repo on GitHub and click the green “Code” button.
- Switch over to the “SSH” tab.
- Copy the SSH URL.



13. Back in your EC2 terminal, run the following commands:

- a. `cd ~` to make sure you're at the root
- b. `git clone git@github.com:username/repo-name.git` (use the SSH URL you just copied)

14. Your repo should now be on your server!

# Get App Up and Running

*Now it's time for the exciting part! By the end of this section you should have your app running on your server and you should be able to access it by IP address.*

## Prepare Remote Code

1. Ensure your project cloned down from GitHub correctly.
  - a. If not, go back over the section above.
2. Change directories into your project folder.
3. Run `npm i` to get all your dependencies installed.

## Prerouting

*All internet traffic is transmitted in packets. Prerouting is a rule type that defines what should happen with a packet before it gets routed. You'll define a prerouting rule in your server's iptables - the place where internet protocol rules live.*

1. Read the explanation of the command in the table below.
2. Copy and paste the command into your EC2 terminal, **replacing the 4000 at the end with the port that your app is using.**

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 4000
```

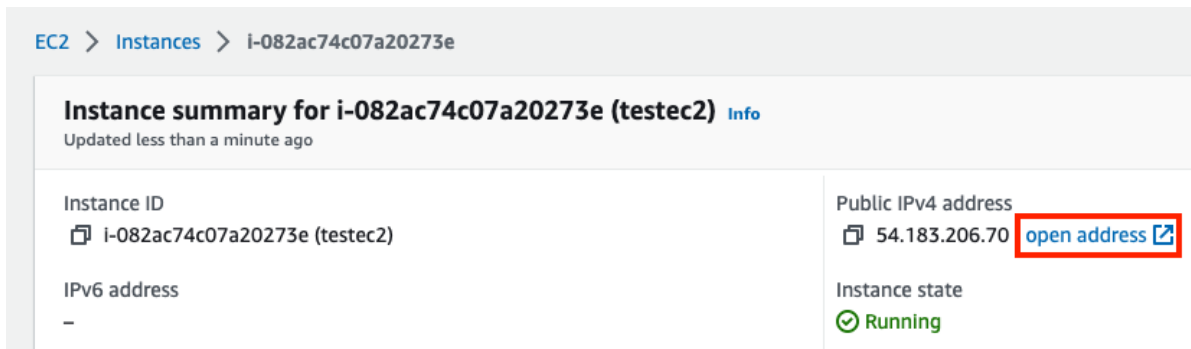
| Part                  | What it's telling the computer                                      | Notes   |
|-----------------------|---|---|
| <code>sudo</code>     | Run this command as an admin  |   |
| <code>iptables</code> | I want to make changes to my iptables                               |   |
| <code>-t nat</code>   | Specifically, I want to make changes to the "nat" table             | <code>-t</code> stands for table, it can also be written as <code>--table</code><br><code>nat</code> stands for Network Address Translation |
| <code>-A</code>       | The change I want to make is going to append something to the table | <code>A</code> stands for append  |

|                              |   |   |
|------------------------------|---|---|
| <code>PREROUTING</code>      | Append the change to the PREROUTING portion of the table        |   |
| <code>-p tcp</code>          | This rule should apply to requests coming from the tcp protocol | <code>-p</code> stands for protocol, it can also be written as <code>--protocol</code>  |
| <code>--dport 80</code>      | Apply the rule to requests coming into port 80                  | <code>--dport</code> stands for destination port<br>80 is the default port for HTTP traffic   |
| <code>-j REDIRECT</code>     | The rule is that I want to redirect those requests...           | <code>-j</code> stands for jump, it can also be written as <code>--jump</code><br><code>-j</code> requires a target, REDIRECT is the target |
| <code>--to-ports 4000</code> | ...to port 4000   |   |

## Test With Nodemon




*Before you get to running the app with pm2, you'll use nodemon since you're familiar with it.*

1. In your EC2 terminal, make sure you're at the root of your project folder (where you would normally run nodemon).
2. Using the file path to your server file, run `nodemon server.js`
3. If everything is running correctly, head back to your AWS Management Console. On your instance's detail page, click "open address" next to the Public IP address. **Once it opens, you'll probably need to delete the "s" from "https" at the beginning of the address.**



EC2 > Instances > i-082ac74c07a20273e

**Instance summary for i-082ac74c07a20273e (testec2)** [Info](#)  
Updated less than a minute ago

|   |  |
|---|--|
| <b>Instance ID</b><br> i-082ac74c07a20273e (testec2) | <b>Public IPv4 address</b><br> 54.183.206.70 <a href="#">open address</a> |
| <b>IPv6 address</b><br>-  | <b>Instance state</b><br> <b>Running</b>                                  |

4. Your project should be running!
5. If it's not, check for errors in your EC2 terminal and in the devtools. If you need help, reach out in the queue before moving on to the next section.

## Run App With pm2

*Finally! This is the last set of instructions to get your app up and running. There is one set of instructions after this that cover how to update the hosted version of your site when you make changes.*

1. In your EC2 terminal, kill the nodemon process if it's running.
2. At the root of your project, run: `pm2 start [FILEPATH] --name [NAME]`
  - a. `[FILEPATH]` is where the file path to your server file goes
  - b. `[NAME]` can be whatever you'd like to call your project
  - c. For example: `pm2 start server.js --name deployment-demo`
3. You should get a nice message from pm2 if everything is successful. Head back to your instance's IP address to check if your app is still running. If it is, **you have successfully deployed your app!**
4. If not, try to figure out any errors you're getting and then head to the queue for support.

## Update Your Deployed Project

*What happens when you want to make a change to your deployed project? Have no fear, you'll just need to do a mini-version of what you did to deploy it initially. Reference the above sections for any commands you're not sure about.*

1. Make changes locally in VS Code – run and test your app locally as normal.
2. Git add, commit, and push your changes.
3. Connect to your EC2 instance.
4. `cd` into your project directory.
5. Run `git pull origin main`.
  - a. This assumes that you're working on the main branch.
  - b. If this doesn't work, try `git fetch` and then pull again.



6. Run `npm i` if you've made changes to your dependencies (or if you're feeling a little stitious).
7. Run `pm2 restart [NAME]`.
  - a. `[NAME]` should be the name that you gave the project when you initially started the pm2 process.
  - b. If you don't know the name, run `pm2 list` to see what's running with pm2.
8. You're back in business!

## Reference Links

- [Amazon EC2 Key Pairs](#)
- [AWS EC2 Docs](#)
- [Express static](#)
- [GitHub SSH Docs](#)
- [iptables](#)
- [PM2 Quickstart Docs](#)
- [SSH Wikipedia page](#)
- [TCP Wikipedia page](#)
- [Virtual Machine Wikipedia page](#) (your EC2 instance is a virtual machine)