**Practice exercise**

**EC2 Exercise 1.1: Host a Static Webpage**

1. Launch an EC2 instance through the AWS console (i.e. build a “virtual laptop” to serve your website)
2. SSH into to the EC2 instance and install a web server (i.e. use the terminal to “log in” to the “virtual laptop” and interact with it)
3. Host a static webpage on the EC2 instance (i.e. “deploy” a static html file to the web)

**Launch an EC2 Instance**

(Note: These posts do not use screenshots. This is intended to require a bit of fumbling and deliberation as you navigate through the AWS console.)

If you haven’t created an AWS account yet, do that first.

Sign in to the AWS console and search for “EC2”. Navigate to the EC2 dashboard and click “Launch Instance”.

**1. Choose AMI**

In this step you will choose the operating system for your “virtual laptop” and the basic set of software that will come preinstalled.

Choose a free-tier eligible Linux option (**Amazon Linux AMI, SSD Volume Type**) and click “Select”

**2. Choose Instance Type**

In this step, you will choose how powerful the “virtual laptop” will be — things like the number of CPUs, amount of memory, type of hard disk/storage, and network speed.

Choose the option marked as free tier eligible (**General Purpose — t2.micro)**— it will be one of the smallest, least powerful options. Click next to configure instance details.

**3. Configure Instance**

In this step, you will accept all of the default options. Glance at the options to get a sense of the types of things that can be done here.

Click next to add storage.

**4. Add Storage**

In this step, you will choose the hard drive(s) for your “virtual laptop”, and how fast it will perform. By default, a hard drive is already attached (it is known as a root drive because it is capable of starting the operating system).

You can change the default drive’s configuration and/or add additional hard drives to your “virtual laptop” in this step.

We will accept the default options and move to the next step. Click next to add tags.

**5. Add Tags**

In this step you can add “tags” to help you to manage and administer your AWS resources. We don’t have a need for this, but add one for the experience.

Click Add Tag— enter “name” for the key and “WebServer” for the value.

Click Next to Configure the Security Group

**6. Configure Security Group**

In this step, you will choose the type of traffic your “virtual laptop” will allow from the outside. You need to allow two types of traffic — SSH (so we can “log in” to the virtual laptop) and HTTP (so we can view our webpage through the browser).

Select “Create a new security group”. Enter anything you want for the name and description

Keep the SSH rule that is already listed.

Click “add rule”. Select HTTP for the type and keep everything else as it is.

Click review and launch.

**7. Review**

Ignore the security warning (we will tear down this instance as soon as we are done) and click Launch.

A pop up window will ask you to select or create a key pair. A key pair is needed to securely SSH (“log in”) to our new EC2 instance.

* Select ‘Create a new key pair’
* Give the Key Pair a name — e.g. ‘ec2-key-pair’
* Click Download Key Pair
* Click Launch Instances
* Click View Instance to navigate back to the EC2 dashboard

**SSH into the EC2 instance and Install a Web Server**

You will see your new instance listed on the EC2 dashboard. Wait until the Instance State is ‘running’.

Selecting the instance (click the button next to the instance) displays information about the instance below. In this area, you will see the IPv4 Public IP address of your instance. Copy it to your clipboard.

Navigate to your terminal and do the following:

* Change the permissions on your key-pair file

First, save the downloaded key-pair .pem file to a directory of your choice (I put mine in my ~/.ssh/ directory). Then change the permissions with the following command. (If you don’t do this, the key-pair is considered “too permissive”, or unsafe, because it is ostensibly readable by other users. You won’t be able to SSH into the EC2 instance as a result)

chmod 400 <path\_to\_key\_pair\_file>

2. SSH into your new EC2 instance

ssh -i <path\_to\_key\_pair\_file> ec2-user@<public\_ip\_from\_dashboard>

Type yes to continue.

At this point, your terminal is now interacting directly with your EC2 instance (aka your “virtual laptop”) — rather than your physical machine.

3. Elevate your privileges

sudo su

4. Update all of the packages on the instance

yum update -y

(note: if you are familiar with using homebrew on your Mac, you can think of the way we are usingyum here as similar to brew)

5. Install an apache webserver

yum install httpd -y

6. Start the webserver

service httpd start

7. Configure the web server to restart if it gets stopped

chkconfig httpd on

**Add a static HTML file to be served**

By default, the apache web server will display the index.html file found in /var/www/html directory in the root path of your website.

In this section you will create an index.html file to be served.

* Navigate to the directory

cd /var/www/html

2. Manually create an index.html file in this directory

Using your preferred editor (vi, nano, etc) create the index.html file:

nano index.html

Add valid html to the file (e.g.):

<html><body>My first EC2 instance</body></html>

Exit and save. Make sure that the file has content:

cat index.html

Navigate back to the EC2 dashboard in the AWS console and copy the Public DNS(IPV4) of your instance into your clipboard. Paste that address into your browser. If all went well, you will see the html that you just created!

**Clean Up**

Navigate to the EC2 dashboard, select your instance, and click on Actions. Select Instance State → Terminate. Confirm that you want to terminate and you’re done. This will automatically kick you out of the SSH session in your terminal.

**EC2 Exercise 1.2: Host a Static Webpage with Content from Github**

In this exercise you will clone a Github repo containing and index.html file into your EC2 instance, rather than manually creating the index.html file on the instance itself.

This exercise is designed to reinforce the skills covered in the last exercise, and further emphasize that interacting with your EC2 instance is essentially the same as interacting with your physical machine (albeit a Linux one).

In this exercise, you are going to do the following:

* Save an index.html file to a repository on Github
* Launch an EC2 instance through the AWS console
* SSH into to the EC2 instance and install a web server
* Install Git on the EC2 instance
* Clone a Repo from Github onto the EC2 instance

**Save an index.html file to a repository on Github**

On your local machine, initialize a git directory and create an index.html file. Add valid html to the file and push the repo to Github.

**Launch an EC2 Instance**

Follow the steps in the ‘Launch an EC2 Instance’ section of [Exercise 1.1](https://medium.com/@KerrySheldon/ec2-exercise-1-1-host-a-static-webpage-9732b91c78ef). However, in the final step, you will NOT need to create a new key-pair. You can select and utilize the key-pair file from your previous exercise.

**SSH into the EC2 Instance and Install a Web Server**

Follow the steps in the ‘SSH into the EC2 Instance and Install a Web Server’ section of [Exercise 1.1](https://medium.com/@KerrySheldon/ec2-exercise-1-1-host-a-static-webpage-9732b91c78ef).

**Install Git on the EC2 Instance**

At this point you should still be in your EC2 instance. Install git with the following command

yum install git -y

**Clone a Repo from Github onto the EC2 instance**

As you may recall from [Exercise 1.1](https://medium.com/@KerrySheldon/ec2-exercise-1-1-host-a-static-webpage-9732b91c78ef), the apache web server will display the index.html file found in /var/www/html directory in the root path of your website.

You want to clone the repo you created earlier into that directory.

* Navigate to the directory

cd /var/www/html

2. Attempt to clone the contents of the repository into this directory using SSH.

(note the . at the end of the command to put the contents of the repo into the current directory)

git clone [git@github.com](mailto:git@github.com):<your\_git\_user\_name>/<repo\_name>.git .

This will not work because Github will not find an appropriate SSH key on your EC2 instance to authenticate your access. Just like your physical computer, you need to generate public and private SSH keys on the instance and add the public key to Github.

3. Generate your SSH keys on the EC2 instance

Run the following commands to generate your SSH keys on the EC2 instance (If you want to learn more about SSH key generation, check out this [page](https://www.ssh.com/ssh/keygen/)):

ssh-keygen -t rsa -C "your-email@gmail.com"

eval "$(ssh-agent -s)"

ssh-add ~/.ssh/id\_rsa

cat ~/.ssh/id\_rsa.pub

Copy the key that is displayed from the last command to your clipboard.

4. Add the Public SSH key to your Github Account

Follow the steps described [here](https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/).

5. Clone the repository into you EC2 Instance

Now you should be able to clone the contents of the repository using the following command.

git clone [git@github.com](mailto:git@github.com):<your\_git\_user\_name>/<repo\_name>.git .

Make sure the index.html file is there and contains content:

cat index.html

Navigate back to the EC2 dashboard in the AWS console and and copy the Public DNS(IPV4) of your instance into your clipboard. Paste the address into your browser. If all went well, you will see the html that you pushed to Github at the beginning of this exercise!

**Clean Up**

Navigate to the EC2 dashboard, select your instance, and click on Actions. Select Instance State → Terminate. Confirm that you want to terminate and you’re done. This will automatically kick you out of the SSH session in your terminal.

**EC2 Exercise 1.3: Host a Static Webpage behind a Load Balancer**

(This post is part of the [AWS for the Bootcamp Grad](https://medium.com/@KerrySheldon/aws-for-the-bootcamp-grad-79dfd91d0ff8) series. The series consists of exercises to build familiarity with various aspects of the AWS ecosystem. Again, all of these posts are “exercises” for introductory exposure to AWS — they are NOT represented as best practices.)

**Background**

In this exercise, you will host a website behind a load balancer. The exercise will demonstrate how the load balancer distributes traffic across the multiple EC2 instances that are attached to it. It will also demonstrate how the load balancer behaves when one of the instances becomes unhealthy/unable to respond to requests.

Specifically, you will do the following:

* Access metadata for an EC2 instance
* Create multiple EC2 instances that run a shell script on launch
* Create a load balancer and attach the EC2 instances to it
* Monitor the behavior of the load balancer when one of the EC2 instances becomes unhealthy

**Access Metadata for an EC2 Instance**

In the later steps of this exercise, you will attach multiple EC2 instances to a load balancer and observe how traffic is routed to each of the the “attached” instances. To observe that behavior, you will display the IP address of the EC2 instance that served the request.

In this step, you will create an EC2 instance simply to learn how to access EC2 metadata. Follow the steps in the ‘Launch an EC2 Instance’ section of [Exercise 1.1](https://medium.com/@KerrySheldon/ec2-exercise-1-1-host-a-static-webpage-9732b91c78ef) to create an instance.

Within an instance, metadata is available at the following URL: http://169.254.169.254/latest/meta-data/

Before you SSH into your new EC2 instance, try to run the following command from your terminal (note: you will need to kill it with a Ctrl-C because it will hang).

curl http://169.254.169.254/latest/meta-data/

After you kill that command, SSH into the EC2 instance as you have in previous exercises. Then run the same command.

This time, you will get a response listing available meta-data items. For example, you will be able to get the public IP address of the instance with the following command:

curl http://169.254.169.254/latest/meta-data/public-ipv4

Feel free to access other metadata items from the list. Once you are done, exit the SSH session and terminate this EC2 instance from the AWS console. You will be creating new instances in the next section.

**Create Multiple EC2 Instances that Run a Shell Script on Launch**

In this step, you will create two new EC2 instances from the AWS console simultaneously. You will follow the same steps as described in the ‘Launch an EC2 Instance’ section of [Exercise 1.1](https://medium.com/@KerrySheldon/ec2-exercise-1-1-host-a-static-webpage-9732b91c78ef) — with the exception of the following two changes to *Step 3 Configure Instance Details*:

* Enter 2 for “Number of Instances”
* Expand the “Advanced Details” section and paste the following text into the “User Data” input box:

#!/bin/bash

yum update -y

yum install httpd -y

service httpd start

chkconfig httpd on

cd /var/www/html

echo "<html><body>IP address of this instance: " > index.html

curl http://169.254.169.254/latest/meta-data/public-ipv4 >> index.html

echo "</body></html>" >> index.html

You should recognize many of these commands from the previous exercises. In those exercises, you entered these commands once you SSH’d into the instance. By adding this script as “user data”, these commands will be run when the instance is launched (see [documentation](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html) here).

A few of these commands that are new/different:

* #!/bin/bash : if you’ve written a bash script before, you will recognize this line. #! is known as a [shebang](https://en.wikipedia.org/wiki/Shebang_%28Unix%29) and this line tells the EC2 instance to use the bash shell to run the rest of the commands (aka the script).
* **NO** sudo su : in the previous examples, you ran sudo su within the instance. In this case, the script is run as the root user and this not needed.
* The final three commands build the index.html file that will be served at the root path of your webpage. The second of these lines is familiar from the previous step in this exercise. This command redirects and appends the output of the curl command into the html file. In this case, the output is the public IP address of the instance.

Complete the rest of the EC2 launch process as you have in previous exercises.

**Create a load balancer and attach the EC2 instances**

Make note of the availability zone of the EC2 instances that you just created — you will need to know it as you set up your load balancer.

From the EC2 dashboard, select ‘Load Balancers’ from the sidebar on the left. Click ‘Create Load Balancer’, then ‘Create’ an Application Load Balancer.

**Step 1: Configure Load Balancer**

Enter a name of your choice for the load balancer (e.g. MyLoadBalancer).

Under availability zones, select the availability zone of your EC2 instance and one other of your choice (in this example, this second selection is moot. See [here](https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-subnets.html) for more information on availability zones for load balancers ).

Click Next: Configure Security Settings.

**Step 2: Configure Security Settings**

Don’t worry about the warning, click Next: Configure Security Groups.

**Step 3: Configure Security Group**

Select the Security Group that you’ve been using for creating your EC2 instances.

Click Next: Configure Routing.

**Step 4: Configure Routing**

In the this step, you are telling the load balancer how to determine if an ‘attached’ instance is healthy and capable of receiving requests (The attached instances are known as targets, and together they are known as a ‘Target Group’).

Enter a name of your choice for the target group (e.g. MyTargetGroup).

The Health Check section defines how the load balancer will determine whether it should route traffic to the instances in the target group. It will route traffic to an instance as long as it is ‘healthy’.

The protocol and path section of this step dictate the type of request that will be made to determine the health of the instance. Accept the default options, meaning that an HTTP request to the root path will be utilized to determine the health of the instance.

In advanced health check settings, you can describe the process that should be followed in order to evaluate the health of the instance. Hover over the information icons to get more details for each setting. You can change the healthy threshold, unhealthy threshold, timeout and interval options to the ‘lowest’ possible setting.

Click Next: Register Targets

**Step 5: Register Targets**

In the instances section of this page, select both of your EC2 instances and click “Add to registered”. Then click Next: Review and then Create.

**Observe the Behavior of the Load Balancer**

It will take time for your load balancer to provision. From the Load Balancer dashboard, wait until the State of your new load balancer is ‘active’. Copy/paste the DNS name of the load balancer into your browser. If all went well, you should see the following html: *IP address of this instance: <an\_ip\_address>*

Refresh the page multiple times and note how the IP address changes. This demonstrates that the load balancer is routing traffic to both of your instances.

After you’ve done that, SSH into either one of the EC2 instances and run the following commands to remove the index.html file that is served at the root path of your website.

sudo su

rm /var/www/html/index.html

Return to your browser and refresh the page multiple times. Now, you should only see the IP address of the ‘other’ EC2 instance.

As you may recall, the load balancer was configured to check the health of the instance at the root path. Because you deleted the index.html file, the EC2 instance is not sending a success response from the root path, and the instance is considered unhealthy.

Go back to the AWS console. Click on ‘Target Groups’ in the sidebar of the EC2 dashboard. Click on the ‘Targets’ tab and you will see that one of your EC2 instances has an ‘unhealthy’ status. Click on the ‘Monitoring’ tab and can see the graph of Unhealthy Hosts and Healthy Hosts.

**Clean Up**

Navigate back to the Load Balancers section of the EC2 dashboard. Click Actions → Delete on the Load Balancer you created. Navigate to the Target Groups section. Click Actions → Delete on the Target group you created

Navigate to the instances section of the EC2 dashboard, select both instances, and click on Actions. Select Instance State → Terminate. Confirm that you want to terminate and you’re done.

**S3 Exercise 1.4: Host a Static Website on S3**

(This post is part of the [AWS for the Bootcamp Grad](https://medium.com/@KerrySheldon/aws-for-the-bootcamp-grad-79dfd91d0ff8) series. The series consists of exercises to build familiarity with various aspects of the AWS ecosystem. Again, all of these posts are “exercises” for introductory exposure to AWS — they are NOT represented as best practices.)

**Background**

S3, or Simple Storage Service, is an AWS service that provides an interface for storing and retrieving data from anywhere on the web. Compared with EC2, S3 feels more approachable for a beginner, due to its familiar terminology and concepts. Specifically, the parallels between the structure of S3 and the file system on your local computer make it comparatively easy to pick up.

The fundamental entities stored in S3 are known as **objects** (as opposed to files) and they are stored in **buckets** (as opposed to folders). Buckets have a domain name, and the objects in the buckets have a key value (e.g. the name of the object). For example, an object with the key value of myPhoto.jpg stored in the my-bucket bucket would be accessible at: *http://s3.amazonaws.com/my-bucket/myPhoto.jpg*

In this exercise, you will do the following:

* Create an S3 bucket
* Configure the S3 bucket to host a static website
* Upload html files to the S3 bucket from the AWS console
* Host a static website with a client-side script from S3.

**Before You Begin**

Create index.html and error.html files locally. Copy the following into the index.html file:

<h4>I am a:</h4>

<button onClick=showDogContent()>Dog</button>

<button onClick=showCatContent()>Cat</button>

<h1 id="content"></h1>

<script>

const contentEl = document.getElementById("content")

const dogContent = "Woof Woof Woof"

const catContent = "Meow Meow Meow"

function showDogContent() {

contentEl.innerHTML = dogContent

}

function showCatContent() {

contentEl.innerHTML = catContent

}

</script>

Note that a client-side script on this file inserts additional html content when one the buttons is clicked.

Copy the following into the error.html file:

<h1>Whoops</h1>

**Create an S3 Bucket**

Navigate to the S3 dashboard from the AWS console. Click *Create Bucket*and enter a bucket name of your choice (bucket names must be unique across all AWS accounts) and select the region closest to your location. Click *Next*. Click *Next*again on the Set Properties section.

In Set Permissions step, select *Grant public read access to this bucket*under the Manage public permissions section. Click *Next*then *Create bucket.*

**Configure the Bucket to Host a Static Website**

From your S3 dashboard, click the name of the bucket you just created. Click the *Properties* tab and click *Static website hosting*. Select *Use this bucket to host a website*to configure the static website.

The endpoint for your static website is shown at the top of the form. This is your website URL. For index document, type index.html and for error document, type error.html.

This means that the index.html object stored on this bucket will be served at the root of your website. If there is a problem serving that file, the error.html object stored on this bucket will be served.

Click S*ave*.

**Upload HTML files to the S3 Bucket**

Now you will upload the HTML files for your website.

Click the *Overview* tab for the bucket and then click *Upload.*Click *Add files* and select the index.html and error.html files you created earlier. Click *Next.*

In Set Permissions step, select *Grant public read access to this object(s)*under the *Manage public permissions s*ection. Click *Next,*maintain all of the default options in the Set Properties step and click *Next.*Finally, click *Upload.*

You will see both of your files listed in your bucket.

**Visit Your Website**

Navigate to your website URL (reminder: you can find from the AWS console it by clicking *Properties → Static website hosting*from the S3 bucket*).*Your index.html content should appear. Click on the buttons to execute the client-side script.

To see the error page, navigate back to the AWS console. Click the *Overview* tab in your bucket and click on the link for the index.html object. Click on the *Permissions*tab and click the radio button under *Public access.*Deselect *Read Object*and click *Save*. Navigate back to your website URL and refresh the page. You will now see the error.html content, as the website is unable to access the private index.html file.

Navigate back to your AWS console and make the index.html object public again. Then click the *Overview*tab.

Notice that there is a link shown at the bottom for the index.html object. This link is known as the REST URL for the object (as opposed to the website URL you used earlier). If you click this link, you will see your index.html content — just like you did from the website URL. However, this URL lacks some of the behavior provided at the website URL (such as error handling). See this [page](https://docs.aws.amazon.com/AmazonS3/latest/dev/WebsiteEndpoints.html) for more information on the difference between these URLs.

**Clean Up**

Navigate to your S3 dashboard. Click the bucket symbol next to your bucket name. Then click *Delete Bucket.*Type the name of the bucket, and click *Confirm.*