# CS132 Lab Five: AWS

## — AWS —

Amazon Web Services, or AWS, is a suite of cloud services provided by Amazon. AWS provide tools for computing, storage, databases, management and more. The services are designed to be massively scalable, so they work well for small projects but can support huge sites as well. In CS132 we will be using EC2 to run our server and S3 to serve static content.

There are many other service providers out there -- your TAs use among other things Rackspace, Linode, and Digital Ocean. AWS is probably the largest of these services, and provided us with an education grant.

## — *Disclaimer* —

> *While Amazon offers a free tier of some of the AWS services as a trial, most usages incur some cost. Amazon has been generous enough to provide a $100 grant for you to use this semester. This should be ample to cover any reasonable usage of AWS during the course, but please be cautious! As this grant is to you personally, you will be responsible for any additional charges, and not Brown CS. Monitor your usage and you should be fine. We recommend you set up alerts to warn you about your usage.*

## — Setting up AWS —

Go to AWS and click the **Sign Up** button in the header. Either login with your existing Amazon account if you have one, or create a new one. Amazon has given you all $100 credits for EC2. Please read the instructions to access this credit (come to the front of the lab to get your code). Be sure to read the terms of service! There are restrictions to what the credit can be put towards. Claim your credit here.

### AWS Console

Take a look at the AWS console. From the homepage the console can be reached by clicking **My Account/Console > AWS Management Console** in the header. Take a second to check out some of the other services offered.

## — S3 —

S3 is Amazon's cloud-storage solution. S3 is similar to Dropbox or Google Drive in concept, but it is especially geared towards serving static files on the web. This is great for hosting static resources such as images, Javascript, CSS or static HTML pages. While you can serve static content via Express (or another framework) S3 is often faster and can scale to any request load. It's charged per gigabyte stored plus per-request (by number of requests and total bandwidth).

## — EC2 —

EC2 is Amazon's cloud-server solution. EC2 lets you run virtual machines on Amazon's cloud which we'll be using to host a simple Node server. AWS makes it easy to scale instances or fire up new ones. We won't need to do that right now, but it's nice to have the option. This scalability is one of the things that makes EC2 so popular.

## — Getting started with S3 —

Go to your AWS console and click on S3. You should see a big blue button in the upper left that says, "Create Bucket". "Buckets" are just top level folders in S3, but they have to be unique across all S3 buckets, so click the button and create a new one. Name it something like "yourlogin-cs132lab". Leave the region as US Standard and ignore logging for now (it can always be turned on later).

You should see your new bucket in the bucket list (tee-hee). If you select your bucket and click properties on the right side you should see the details of the bucket as well as several option tabs. Click around and look at the different options available. We're interested in 'Static Website Hosting'. Click to enable website hosting, we'll fill in the details later.

Now click on your bucket name to go into your bucket. Inside a bucket you can upload and manage files as well as create folders. Upload an image to your bucket. Feel free to use this as your image if you're not feeling creative: Miley

You can right click on the filename to rename it if you like. Now create a simple html page with a title and an image tag. The image tag should have your image name as the source like so:

```
<img src="/miley.jpg"></img>
```

When you're done, upload this html file to your bucket. Since you uploaded this to the same directory as image, you can use a relative path to your image like above.

Now, select both your files by checking the box to the left of each and right click one of them. You should see the option to "Make Public" which will allow others to access these files. Note that by default files stored on AWS are not publicly visible. Go ahead and do that.

Click 'All Buckets' in the upper left to go back to your bucket list, and view your bucket's properties by clicking on the magnifying glass icon next to the bucket name. Expand the Static Website Hosting section and select 'Enable website hosting'. Now fill in the 'Index Document' field in the Static Website Hosting option with the name of your html file. Now click the endpoint, which should look something like this:

`http://yourbucketname.s3.amazonaws.com/page.html`

You should see your static site! That's all there is to it.

(A technical sidenote: you can use a CNAME record of a subdomain to host static content on you own domain. For example, images.examples.com to images-bucket.s3.amazonaws.com. This is a very simple way of hosting an all-static website or the content thereof.)

## — Getting started with EC2 —

EC2 is Amazon's service for virtual private servers. These are operating systems which are actually running as programs in a so-called "host operating system" running virtualization software (which sounds really slow, but works quite well in practice.) The computers are called instances in Amazon's parlance, and they come in several types. For this lab you'll be using the standard small instance, `m1.small`, which costs $0.06 per hour.

The first thing you'll do is choose the operating system. You'll be using Ubuntu server edition, but a particular "image" rather than the standard install you would get if you put Ubuntu on a machine with a CD. There are many images out there, but you'll use one that comes with Node pre-installed.

The image is BitNami Node.js Stack. From your EC2 dashboard, make sure that your region is US East (N. Virginia) in the header bar, then click "Launch Instance". Under the "AWS Marketplace" section, search for "bitnami nodejs", and select the 64-bit option. Choose the `m1.small` instance, and click "Review and Launch".

This should bring up the configuration dialog for a new instance. You'll have to go through a couple steps to start it up:

- Edit the "Security Groups" section. Choose "Create a new security group", and name it whatever you like. By default, Amazon blocks incoming traffic to your instance, so you need to allow SSH, HTTP, and ping.
- Add the following rules to the security group:
  - Allow inbound SSH from anywhere `0.0.0.0/0`
  - Allow inbound HTTP from anywhere `0.0.0.0/0`
  - Allow inbound "Custom ICMP" using the "Echo Request" protocol from anywhere `0.0.0.0/0`
- Leave everything else as is, and launch your instance. This should bring up a dialog that prompts you to set up SSH.
- Choose "Create new key pair." Name it whatever you want and download the key. This is the key which will allow you to log in via SSH. Finally, click "Launch Instances"

- In your terminal, `chmod 600 /path/to/key.pem` (This is important: it sets the permissions of the keyfile so that only you can read it.)

When you go back to your EC2 dashboard and navigate to "Instances", you should see the list of instances you own (which should be 1). You can give the server you just started a name. Wait for it to turn to the green "running" state and click on the entry to bring up details. You should see a bunch of data about the new instance, and most importantly the subtitle, which has the address of the machine. It should look something like this:

`ec2-184-72-92-237.compute-1.amazonaws.com`

Now ping your machine to make sure it's online:

```
$ ping -c 8 ec2-184-72-92-237.compute-1.amazonaws.com
PING ec2-184-72-92-237.compute-1.amazonaws.com (184.72.92.237): 56 data bytes
64 bytes from 184.72.92.237: icmp_seq=0 ttl=46 time=36.297 ms
64 bytes from 184.72.92.237: icmp_seq=1 ttl=46 time=79.169 ms
64 bytes from 184.72.92.237: icmp_seq=2 ttl=46 time=97.833 ms
64 bytes from 184.72.92.237: icmp_seq=3 ttl=46 time=38.458 ms
64 bytes from 184.72.92.237: icmp_seq=4 ttl=46 time=83.599 ms
64 bytes from 184.72.92.237: icmp_seq=5 ttl=46 time=72.363 ms
64 bytes from 184.72.92.237: icmp_seq=6 ttl=46 time=172.429 ms
64 bytes from 184.72.92.237: icmp_seq=7 ttl=46 time=74.162 ms

--- ec2-184-72-92-237.compute-1.amazonaws.com ping statistics ---
8 packets transmitted, 8 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 36.297/81.789/172.429/39.650 ms
```

If that worked, congratulations! You've successfully started your own server. (Remember to stop it! We'll cover that later.)

In order to access your machine you'll SSH in with the key you downloaded earlier. Run the following command, replacing the key path and the address of the machine (the stuff that comes after the @).

```
ssh -i ~/some/path/to/key.pem bitnami@ec2-184-72-92-237.compute-1.amazonaws.com
```

If you get a complaint about permissions here it's probably because you didn't `chmod` the key file earlier.

You should now have a shell, something like:

```
bitnami@domU-12-31-39-0A-A4-41:~$
```

Now you're going to set up a very simple application on the server.

First we have to shut down redis and Apache, which come preinstalled and are already running. These steps are specific to the Bitnami image we're using and are not generally relevant. To do so, run the following commands:

```
cd ~/stack
sudo ./ctlscript.sh stop redis
sudo ./ctlscript.sh stop apache
```

Now we're going to set up a new application, called `project`:

```
#Go to your home directory
cd ~
#And make a new folder
mkdir project
#And go into it.
cd project
#Install forever and express (two libraries for Node).
npm install forever
npm install express
#This is a nice little tool which comes with express to set up
#a basic application.
#You can ignore warnings about the directory not being empty.
./node_modules/express/bin/express .
```

Now edit the `package.json` file to have the right information.

If you don't have a preferred editor you can use:

`nano package.json`

`Ctrl-O` will write the file and `Ctrl-X` will quit.

Now `npm install` to install the packages listed in your `package.json`.

Finally, you should be able to run your application:

`sudo PORT=80 node app.js`

`sudo` runs the command as the root user, which is needed because it's using a port number less than 1024. PORT=80 sets an environment variable which is used on line 14 of `app.js` to set the port that the server runs on.

Now if you put your machine's address (like `ec2-184-72-92-237.compute-1.amazonaws.com`) into your browser you should see a "Welcome to Express!" page.

## — Running your application as a service —

You've got your application running, but currently when you launch the application it's attached to your SSH session. When you leave the server it will automatically close, which is bad! (Technical: when the SSH session closes the program gets a SIGHUP.)

Besides this problem, a proper server should do a number of other things like write its output to a log file which you can see later, start up when the system reboots and restart automatically (or raise an alarm) when it dies for an unexpected reason. Processes of this sort are often called daemons or services.

The traditional way to manage a daemon is through a script in `/etc/init.d`. A very simple `/etc/init.d` file is included in this repository as `project` here. Set it up with the following:

```
#Write the project script to /etc/init.d/project
sudo curl 'http://cs.brown.edu/courses/cs132/labs/aws/project' -o /etc/init.d/project

# Set the execute permission for the
# script
sudo chmod a+x /etc/init.d/project

# Update the init system to let it
# know there's a new script!
sudo update-rc.d project defaults
```

Now it's relatively easy to manage with a couple of commands:

```
#Start the server
sudo /etc/init.d/project start

#View the logs (stdout)
less /var/log/project.log
#stderr
less /var/log/project.err

#Follow the logs from the server as they
#update (Ctrl-C to exit)
tail -f /var/log/project.log

#Stop the server
sudo /etc/init.d/project stop
```

This version doesn't restart the process if it has an error and quits. The best tool for that job is probably forever, but combining forever and init.d gets a little complicated and doesn't generalize to running other applications. You can find a guide here.

## — Checkoff —

Show a TA your static page and "Welcome to Express" page (or anything fancier you care to come up with).

## — *Important*: Shut down your instance! —

> *When you're done with the lab it's very important to terminate your instance and delete your data from S3. This will stop them from continuing to charge you on a monthly basis! From the S3 dashboard, delete all the files from your bucket, and then delete the bucket itself. From the EC2 dashboard, go to your Instances list and select your active instance. Terminate it from the Actions dropdown. If you're planning to finish this lab at a later date it's OK to leave your server on for a couple days. At $0.06 per hour it's $1.44 a day, which is well within your $100 budget. But you absoutely must remember to shut it down!*