COSC349 – Assignment 2

William Duggan – 3409853

https://github.com/wduggan/cosc349_asgn2

Introduction:

For assignment two, I decided to build upon my application that I created in assignment one. This application is a very simple online store and administration application with a customer facing webserver, an administration webserver. For this assignment I used two Amazon EC2 Instances/Virtual Machines running Ubuntu to deploy and run each webserver, one Amazon RDS (Relational Database Service) running a MySQL server to host the database, and one Amazon S3 object storage service to store backups of the database and images for the websites. A demonstration video (2 minutes long) of the application working is available within the Git repository, to show evidence of functionality in the cloud.

Note: To watch the demo video of the application working, click on '349-asgn2-demo.mp4' in the Git repository, then click on 'View Raw' and the video will download for you to watch.

How I deployed my application:

Initially, I attempted to have my application be deployed to AWS using Vagrant. However, for some reason on my machine, Vagrant refused to pick up on my AWS credentials whenever I would try to 'vagrant up'. I set my AWS credentials within the "`~/.aws/credentials`" file where you are supposed to set them up, but I would just get errors saying "No environment variables are set, nor profile 'default' exists". Then I tried setting the credentials manually in the terminal using the AWS CLI (Command-Line Interface), but I still received a similar error message. Then finally, I tried putting the credentials within variables in the Vagrantfile itself, and still received errors saying that the "credentials could not be found". So since this was taking up a huge amount of work time, and all the research I did online seemed to lead to a dead end and no solution – I chose to setup my EC2 instances manually on AWS without Vagrant, along with the RDS and S3 which is manual anyway.

Below I describe the process of deploying my application. It is not all necessarily in order, as I jumped between setting up the EC2 and RDS at points.

Setting up two EC2 Virtual Machines for each Website:

I setup the two virtual machines for the websites manually using the EC2 dashboard on AWS. For each machine this involved:

- Choosing an Amazon Machine Image – Ubuntu
- Choosing and configuring the Instance Type which determines the capacity and ability of the machine – t2.micro

- Configuring a security group for the virtual machine to manage the firewall rules for the instance.

Once created, I could use SSH (Secure Shell protocol) to connect to the machines and make any changes. For both machines, I connected to them and installed Apache and PHP for the webservers using:

- `sudo apt-get install -y apache2 php libapache2-mod-php php-mysql`

While on the topic of installations, to accomplish features related to the RDS and S3 services (**which I will explain further down the document**), I also installed MySQL and Python and Boto3 onto the VMs.

- For MySQL:
  - `sudo apt-get -y install mysql-server`
- For Python and Boto3:
  - `sudo apt install -y python3-pip awscli`
  - `export LC_ALL="en_US.UTF-8"`
  - `sudo pip3 install boto3`

After installing Apache and PHP, I needed to copy my web files from the assignment one project over to the Virtual Machines using the SCP (Secure Copy Protocol). For each machine this involved:

1. Editing the conf files (on my machine) for each webserver to direct Apache to the directory within the EC2 instance that I wanted to put my web files in, so that it would know which directory to look for.
2. Using SCP to copy the conf files over to each respective EC2 instance.
3. Moving the conf files from the root of the EC2 instances into the `/etc/apache2/sites-available` directory (where the Apache conf files are kept).
4. Editing the website PHP/HTML files (on my machine) to access the endpoint of the RDS with the MySQL database inside, rather than to my old Vagrant VM IP addresses used in assignment one.
5. Using SCP to copy the website folders over to each respective EC2 instance and moving them into the `/var/www/` directory.
   - Because the conf files set the Apache server to get the files for the website from `/var/www/admin-website` for the admin site, and `/var/www/customer-website` for the customer site
6. Then I used the following commands:
   a. To enable my new website configuration files: `sudo a2ensite 'file-name'`
   b. To disable the default Apache website configuration file: `sudo a2dissite 000-default`
   c. To reload the Apache webserver configuration: `sudo service apache2 reload`

Obviously, this didn't all work immediately after copying my files over. But now that my files were on the machines, I had to incrementally develop and do any further editing of the web or conf files within the EC2 in a terminal editor such as nano.

Setting up the RDS:

I created an RDS using an AWS tutorial[1] on how to create a MySQL DB instance within the Amazon RDS console. This was straightforward and involved choosing the DB Engine type (MySQL), instance size, username, and password.

- Then I SSH into one of my EC2 virtual machines, so that I could use the Ubuntu/Linux commands and install a MySQL client onto the virtual machine to use to access the RDS.
    - This is because the tutorial I was following said to connect to the RDS database using a MySQL DB client instance.
- I connected to the RDS using the MySQL client with the command (substituting my RDS details in the <> brackets):
    - `mysql -h <endpoint> -P 3306 -u <username> -p`
- Once connected, I created a database as usual in MySQL and ran my `database-setup.sql` script to setup the RDS.

After doing this, the RDS was now populated with tables which the webservers could access using the RDS endpoint.

Setting up the Amazon S3 Bucket:

I setup the Amazon Simple Storage System (S3) using the tutorial on getting started with S3[2]. The tutorial teaches you how to host a static website on S3, but I just used it to setup an empty S3 bucket. I made the bucket in the S3 console which is simply just a storage unit for objects that is publicly accessible so that my EC2 machines can interact with it. My intention was to have an S3 bucket which would store backups of the RDS database every so often. This would be done by the admin EC2 VM querying the RDS database and storing the result in a text file, then sending the text file to the S3 bucket.

- I created a shell script (`database-shell-query.sh`) to be run in the EC2 VM. This script connects to the MySQL client on the RDS, and runs a "`SELECT * FROM products;`" query on the database – then puts the result into a text file.
- Then I created a python script (`s3-input.py`) which uses the Boto3 library to access AWS from python. It looks for the S3 Buckets that exist under my AWS account, then takes the text file created by the shell script and 'puts the text file

[1] https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.CreatingConnecting.MySQL.html

[2] https://cosc349.cspages.otago.ac.nz/cache/s3-website-tute.pdf

object' into the given S3 bucket (349-db-backup). Therefore, creating a backup of the database within S3.

Problem:

However, I ran into some problems with implementing this feature. For me to access AWS from the python code, my AWS access credentials from my AWS Educate Account must be setup within the virtual machine.

Before trying all this within my actual EC2 virtual machine, I used a simple Ubuntu virtual machine deployed through Vagrant from a previous COSC349 lab to test these scripts out and see if I could pull from the database and upload to S3. I put my access credentials into the file "`~/.aws/credentials`" as usual in this VM and ran the shell and python scripts.

In the demo video in my Git repository showing off how the full system works; you can see that the scripts successfully send a backup of the database to the S3 bucket. The example video is using the Vagrant VM. However, when I tried to simply just put these files over to my EC2 instance and set it all up the exact same way with my AWS credentials in the EC2 VM, it doesn't work.

The shell script queries the database fine and places the results in a text file. But when I try to run the python script and interact with the S3 bucket, it fails to locate my AWS credentials. I received these errors:

- "`botocore.exceptions.nocredentialserror unable to locate credentials`"
    - Online sources discuss making sure that the credentials are set in the "`~/.aws/credentials`" file, and to try manually set them in the terminal. Then check with '`aws configure list`' that they are set.
    - After doing this, it says my credentials are set and exist. But then I get this error when retrying the python script:
- "`botocore.exceptions.clienterror an error occurred (invalidaccesskeyid)`"
    - Online sources then stated to specify the access credentials within the python file when accessing the S3 resource, as well as the profile name (default).
    - I then receive this error:
- "`boto3 the config profile (default) could not be found`"
    - Online resources then take me back to where I started and say once again to ensure that my credentials are configured, either in the credentials file or through the terminal.

So based off this, I believe that there is a clear issue with the AWS access credentials within EC2, because despite saying they are configured, it still gives errors saying they can't be found. I did extensive research to try fix this, but I hope that based off the fact I managed to get it working within a different virtual machine (with the same method for setting AWS

credentials) it will be accepted as a proof of concept - and shows that the S3 instance can and should interact with the rest of the system fine.

A compromise:

Just in case the above points and work I've done are not considered during marking for some reason, I decided that just to have an example to show of access to S3 from the webserver – I've added in a fake-company logo on every page in the websites. This image is stored within the S3 Bucket, and then accessed from my EC2 instance inside the HTML pages of the websites using the images URL.

I hope this extra effort will be taken into account, as I am showing an interaction between the EC2 and the S3 Bucket, and in my first attempt I demonstrated how an interaction between the EC2, RDS, and S3 could work with backups. This is all demonstrated in the demo video.

Application Design/Service Interaction:

My application has two Virtual Machines as Amazon EC2 Instances running Ubuntu, one for the customer-webserver, and the second for the admin-webserver. My application uses two further non-EC2 cloud services - one Amazon RDS (Relational Database Service) running a MySQL database, and one Amazon S3 storing backups of the database and website images.

I will start by explaining the RDS database service. The database server has MySQL installed on it and holds two different tables within the database – the 'products' and 'users' tables. The users table holds the username and password of the admin for the admin website, and the products table holds the productid, name, description, price, and quantity for different products in the online store.

The customer-webserver EC2 VM can be accessed through its IPv4 or IPv6 addresses online. Public IPv4 DNS – http://ec2-18-206-54-169.compute-1.amazonaws.com. The VM has Apache installed on it for the http web server, and PHP installed for the functionality to communicate with the RDS database server. The customer page opens at a welcome screen, which can then be navigated to a "View Products" page, where the data from the 'products' table in the database is displayed – using PHP to access the database via the endpoint of the RDS (asgn2-db.cddfazz6mjhm.us-east-1.rds.amazonaws.com). The purpose of this VM webserver is for customers to view the products in the database and then 'buy' a product.

The admin-webserver EC2 VM can also be accessed through its IPv4 or IPv6 addresses online. Public IPv4 DNS – http://ec2-184-73-128-114.compute-1.amazonaws.com. The admin VM also has Apache and PHP installed on it. The admin webserver opens to a login screen, at which the correct username and password (admin, admin) must be entered to access the site's content. The webserver VM sends the forms input to the RDS database via the endpoint, and checks if the values entered matches the content within the 'users' table - if so, the admin is permitted entry. The website displays the product data on the site, but also allows for an admin to add/send new data to the RDS database - providing a simple

form of database management functionality from the webserver. Changes made on the admin site can be seen on the customer site.

The S3 Bucket can store different objects. It will store backups of the database in a text file, which it has received from a virtual machine that has queried the database, put the result in a text file, and passed it on. The S3 bucket also holds images, such as the company's logo which is displayed on the HTML pages in the EC2 by accessing the S3 object.

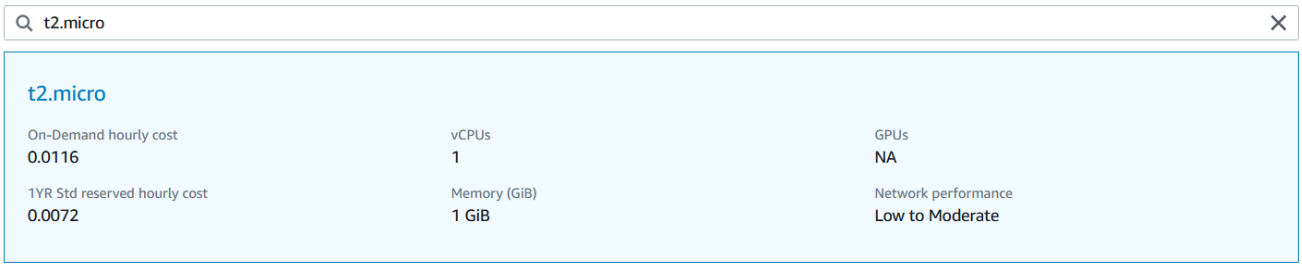
Watch the video in the Git repository for a demo.

Running Costs of the Application in the Cloud:

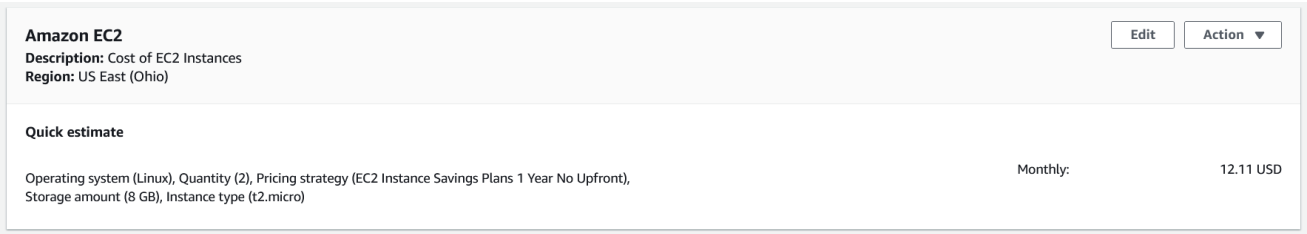
All services are continuously running on the cloud, to offer 24/7 access to users. To calculate the cost of the ongoing running of the application in the cloud, I used the AWS Pricing Calculator, which estimates the running costs of all your selected AWS products and services in USD.

Running Costs While Idle:

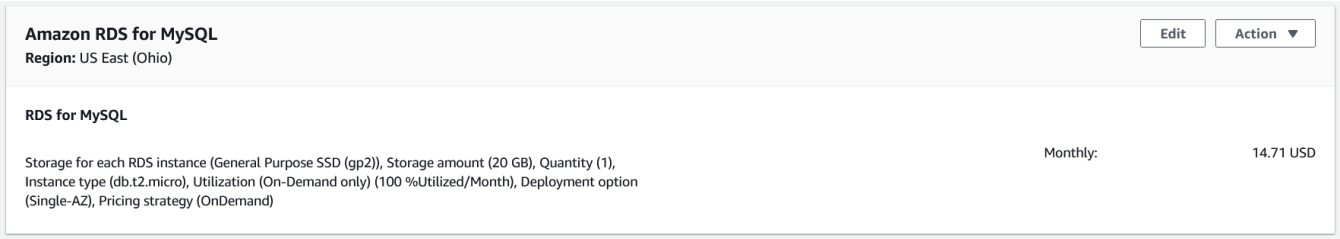
For each individual EC2 instance, the calculator determined that the hourly cost would be:



The monthly cost of both EC2 instances is:

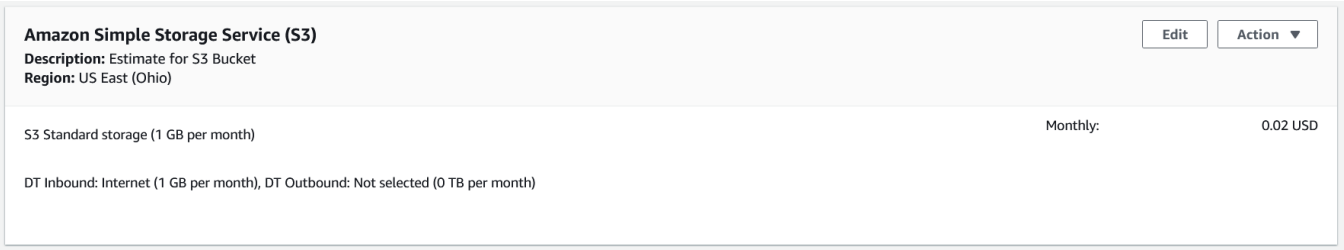


The monthly cost of the RDS Database is:



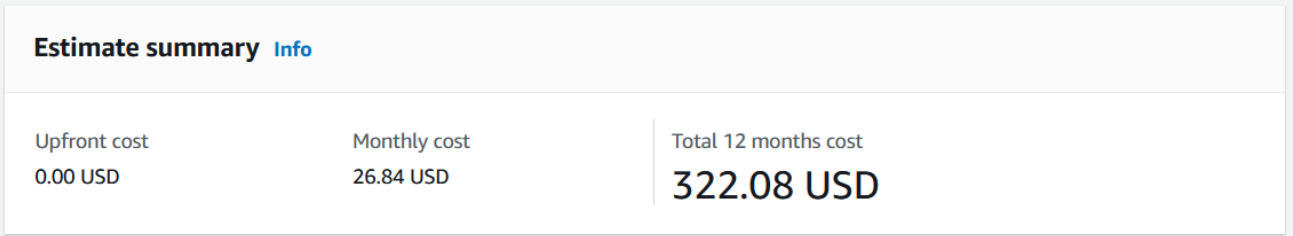
Estimating the cost of the S3 Bucket was more difficult to determine as it is all based off how much you have stored in the bucket. The bucket I used for this assignment currently only has around 24 Kilobytes (KBs) of data which is tiny, but the AWS calculator only

calculates prices for S3 based off Gigabytes or Terabytes of data. So, the lowest I could go was 1 GB of data per month, which resulted in the cost of:



Based off this, and the fact that my S3 has such little data in it, I can assume that the cost of my S3 will be so low that it is nearly free, and won't affect the overall cost at all.

After all these values above have been added up, the overall idle cost of my application is approximately:



Costs from use:

As for the costs from use, I don't believe I am accumulating enough traffic from light use of the application to cause a cost to be accumulated. I tried using the functionality of the application to query the database, as well as sending backups of the database to the S3 bucket – but none of these seemed to cause a change in the balance of my AWS account within the Workbench for my account. If there is a cost accumulating through use, I would estimate that it is probably so small that it isn't making a change to my balance immediately.

How to use the application:

To use the customer website, enter the following into any web browser:

- http://ec2-18-206-54-169.compute-1.amazonaws.com or http://18.206.54.169

The customer page opens at a welcome screen, which can then be navigated to a "View Products" page, where some pre-loaded products/data from the database is displayed to the user – as well as any new data added by the admin.

To use the administration website, enter the following into any web browser:

- http://ec2-184-73-128-114.compute-1.amazonaws.com or http://184.73.128.114

The admin webserver opens to a login screen, at which the correct username and password (admin, admin) must be entered in order to access the site's content. The website then

displays the product data on the site (the same as the customer website), but also allows for an admin to add new products to the database and to the page. This is done from entering information and submitting the form on the ‘Add Products’ page. Changes made on the admin site can be seen on the customer site.