Overview

In this assignment you will be using a web application that is provided to you (MiniTwit - a simple Twitter clone) and deploying it in EC2 using many of the features you would want to use for a service of this type.  First you will set up your AWS account to be ready to create EC2 instances (virtual servers), and then you will create an instance and SSH to it.  Then you will run the MiniTwit application and see that you can reach and use the app from your web browser.  Next you will set up an EC2 instance to automatically start the MiniTwit application when the instance launches, so that you don't need to SSH in and run it manually.  In the next part, you will set up RDS (AWS' Relational Database Service) and modify the MiniTwit application to use RDS for it's database. (Among other things, this will allow multiple MiniTwit servers to share the same data - i.e., same user accounts, tweets, etc. - so that it's transparent to the user which server they're actually on.)  Then you will configure Elastic Load Balancing (ELB) to distribute user (browser) connections between multiple MiniTwit servers, and configure Auto Scaling to scale the number of MiniTwit servers out and in, depending on the load (i.e., how many browser requests are hitting them each second).  Finally, you will set up Auto Scaling for your RDS database as well, so that your MiniTwit servers and the database servers and both scale independently of each other.

This assignment is divided into several parts, but the parts are not all created equal. Don't think that you're half way done just because you completed Part 3!

Also, here are a few top-level documentation pages that you may use now and in future projects too.

AWS Documentation:

[https://aws.amazon.com/documentation/ (Links to an external site.)Links to an external site.](https://aws.amazon.com/documentation/)

Flask:

[http://flask.pocoo.org/ (Links to an external site.)Links to an external site.](http://flask.pocoo.org/)

Code for the MiniTwit application:

[https://github.com/pallets/flask/tree/master/examples/minitwit/ (Links to an external site.)Links to an external site.](https://github.com/pallets/flask/tree/master/examples/minitwit/)

Part 1: Setting Up EC2

In this part, you will create a single EC2 instance and run a simple web application on it using the Flask framework for Python. Before you start, you will need your AWS account fully set up, per the instructions that were emailed to you.

First, log in to AWS (https://console.aws.amazon.com) and go to the EC2 service. If you're in the right place, you should see a toolbar on the left with options like Instances, AMIs, Volumes, Security Groups, etc. The first thing you need to do is make sure you're connecting to the correct Region (i.e., the correct data center - Amazon has several datacenter worldwide for running AWS services). On the upper right, between your account name and "Support" it should say the name of a location (e.g., Oregon, Ohio, N. Virginia, etc.).  If it does not say Oregon, then click on it and choose the option "US West (Oregon)" from the drop down. It should now say Oregon, and you will be using the Oregon Region (data center) from now on.

The next thing you need to do is go to the "Key Paris" section (under the NETWORK & SECURITY heading). You should see a button at the top of the main pane called "Create Key Pair" - click on this. You will be prompted to give it a name - name it whatever you think makes sense. (It won't matter much, since you'll probably only have one key pair for now.) Your browser will then prompt you to download a file with a .pem extension - this file contains your private key, which you will use to SSH to your EC2 instances.  ***Save your .pem file somewhere safe - if you lose it, you cannot recover it!*** Should you lose your .pem file, your key pair will become useless; you will need to create a new key pair then, save the new .pem file, and associate all your EC2 instances with the new key pair.

Finally, configure a default Security Group. This is basically a firewall, which restricts what kind of network traffic from which hosts (machines on the network) is allowed to reach your EC2 instances. Click on "Security Groups" (under the NETWORK & SECURITY heading). There should be one security group already there with a name along the lines of "default".  Select that security group.  In the details section below the list of security groups, you will see "Description", "Inbound", "Outbound", etc.  Click on the "Outbound" tab.  It should say something like "All traffic - All - All - 0.0.0.0/0", which basically means instances using this security group can *send* any network traffic to anywhere. This is fine; what they're allowed to receive is the real security concern, so now click on "Inbound".  You should see a rule that says it will accept all traffic of all types incoming from a particular Security Group ID, and that ID happens to be the ID of this group - in other words, instances in this security group can send anything to other instances in the same security group.  We'll need to add a couple things to let us reach these instances from outside the cloud, though.  Click on the Edit button, and then use the Add Rule button to add the following incoming rules:

1. SSH - TCP - 22 - Custom - 66.194.72.0/24
   * this allows SSH to your instances from campus computers, such as cs1.seattleu.edu
2. SSH - TCP - 22 - My IP
   * this allows SSH to your instances from the computer you are currently using
   * Note: if you change to a different computer, this does not follow you! so make sure you're on the computer you want to use for your work when you set up this rule.
3. Custom TCP Rule - TCP - 5000 - Anywhere
   * this allows connecting to your instances from anywhere on port 5000, which we will use instead of the default HTTP port 80 in order to keep our service a little more private (although not really)

Now that all that's done, you probably won't need to mess with it any more. (Maybe occasionally, but not frequently...)

Part 2: Launching an EC2 Instance

Now that everything's set up, let's launch an EC2 instance and run a web service. Click on "Instances" (under the INSTANCES section), then click on the "Launch Instance" button. Now you will have to choose which AMI (disk image) to use for your EC2 instance's virtual hard drive. On the left are several categories - choose the "My AMIs" category, then check the "Shared with me" option.

You should see an image called "Flask Base". **If you do not see this AMI listed**, it may be because your account has not been fully added to the class yet - please wait for the instructor to email you that your account has been fully set up. (If the instructor has emailed you already and you still don't see it there, please contact the instructor.)

Select the "Flask Base" AMI. You will not be prompted to choose what size instance you want (remember "vertical scaling"). For this purposes, and most things we do in this class, we'll just use the *t2.micro* instance size. Select that and then click "Review and Launch".

You are now at a page that lets you review all the settings for the instance you are about to launch. Feel free to browse around and see what all the settings are - the default settings will be fine for the most part. One thing we do want to change, however, is the Security Group. Find the Security Groups section and click on the "Edit security groups" link at the right side of the page. Then select the "Select an **existing** security group" option, and then choose the default security group that you configured earlier. Now click on the "Review and Launch" button.

*Optional:* In the Tags section, click on the "Edit tags" link on the right side of the page. Add a tag with the key "Name" and then for the value give it whatever name you want this instance to have. This will show up as the instance name in the list of running EC2 instances. Then click on the "Review and Launch" button.

Now that you've configured the security group, and optionally given the instance a name, you can click on "Launch". A pop-up will appear asking about what key pair you want to use. Select "Choose an existing key pair" and then choose they key pair that you created earlier. (You still have the .pem file, right??)  Then check the check box, and click the "Launch Instances" button. Your new instance will now be created!

From the confirmation page, you can click on the instance ID to go back to the list of running instances in the Management Console. Alternatively, although a slightly longer path, you can select Services -> EC2, and then click on "Running Instances" (or select "Instances" from the menu on the left).

At this point, you may want to review what you just did. You will need to launch EC2 instances again in the future, and I will expect you to know how - future assignment instructions may just say "Launch an EC2 instance" without future details.

***Warning:****You pay for each hour of each EC2 instance that you use. The prices are very small, so it's not much to use an instance or two for an hour or two... but if you leave several instances running for hours and hours, then it will start to add up. Be sure not to leave EC2 instances running for long periods of time if there's isn't a good reason!*

Note: You can put EC2 instances into the "stopped" state instead of terminating them. You are not charged for "Stopped" instances, but charges will resume when you "start" them again.

Part 3: Running a Web Service

Go to the list of running instances. If your new instance is still starting up, you will need to wait for the Instance State to become "Running" (and then possibly a little longer than that still - when it changes to "Running" is when the OS *starts* to boot up). Once it's ready, click on it and find the "IPv4 Public IP" in the details section. This is the instance's IP address. (If the "Public DNS (IPv4)" has a value, you may also use that instead of the IP address.)

Use this IP address to SSH to your new instance. Note that you will need to tell SSH to use the .pem key file that you saved earlier. On Linux and MacOS, the ssh command follows this pattern:

ssh -i <pem key file> -l ec2-user <public IP>

Note that ec2-user is the default username for instances based on the Amazon Linux images. You do not need to enter a password because your private key (contained in the .pem file) authenticates you.

Using PuTTY is a little more complicated. As a workaround, you can use PuTTY to SSH to the cs1.seattleu.edu server and then SSH from cs1 to your instance using the command above. You will need to SCP your .pem file to cs1 first, so you have access to it when you SSH to your instance.

If you do want to try your hand at connecting directly to your EC2 instance using PuTTY, you can find some instructions at the following link. Note that you should enter "ec2-user" for the username instead of "root".

[https://linuxacademy.com/blog/linux/connect-to-amazon-ec2-using-putty-private-key-on-windows/ (Links to an external site.)Links to an external site.](https://linuxacademy.com/blog/linux/connect-to-amazon-ec2-using-putty-private-key-on-windows/)

You should now have a Linux command prompt on your EC2 instance. It has been prepared with a web services ready for you to run. This web service has been created using the Python [Flask  (Links to an external site.)Links to an external site.](http://flask.pocoo.org/)framework - more on that later. For now, you can run the "MiniTwit" web application by entering the command "flask run --host=0.0.0.0"

Now open a web browser and enter your instance's IP address with :5000 appended to the end. (This tells it to use port 5000, instead of the default HTTP port 80.) For example:

1.2.3.4:5000

You should see the MiniTwit "Public Timeline" page in your browser - this is the application that is running on your instance! You can play around with it a little bit now to get a feel for the app. It's a simple clone of Twitter - you can create user accounts, log in, post message that appear on your timeline (as well as the public timeline), and follow other users (click on their name to see their timeline, and then there's a follow link) - posts from anyone you follow will also appear on your personal timeline.

You can find the code and README for the MiniTwit application in the flask/examples/minitwit/ directory on your instance, or at this URL:

[https://github.com/pallets/flask/tree/master/examples/minitwit/ (Links to an external site.)Links to an external site.](https://github.com/pallets/flask/tree/master/examples/minitwit/)

***Note:****Some people get a "500 Internal Server Error" response when connecting in the browser, and Flask will report a bunch of errors on the server side. This is because the database hasn't been initialized. We are still investigating why the database automatically initializes for some people (including me) but not for others. For now, if you encounter this error, just run the command "flask initdb" before "flask run --host=0.0.0.0" to manually initialize the database. You will probably also need to add this to the "user data" script in the next section when you set it up to launch the app automatically.*

Part 4: Starting the App Automatically

One thing you may have noticed is that you need to SSH to your instance and manually run the MiniTwit application each time you launch a new EC2 instance. Not only is this annoying extra work, but manual steps like these will become a problem when we want to set up Auto Scaling, which will automatically start new instances for us when the load is high... but it cannot perform manual steps, so we need everything to happen automatically when we start a new instance. Namely, we need the MiniTwit web app to start automatically.

Go back to your EC2 console and start launching a new instance following the same steps as before, but stop at the Review and Launch page - not not click the final "Launch" button just yet!  Find the Instance Details section and click on the "Edit instance details" link. Now scroll down to the bottom of the Instance Details page and expand the "Advanced Details" section. The Advanced Details contain a section called "User data" - user data is how we tell an instance to do something automatically when it starts up.

In this case, make sure the "As text" option is selected. We're going to just type the commands into the provided text box. (For future reference, the "As file" option lets you upload a script file to run, instead of manually typing it into the text box.) The first line you type should be #!/bin/bash which tells the script to run using the bash shell, so that the rest of what we type will be interpreted just as if it was typed at the Linux command prompt. After that, you can start typing commands for it to run. For this assignment, we'll type this:

#!/bin/bash  
echo Starting MiniTwit...  
export FLASK\_APP=minitwit  
flask run --host=0.0.0.0  
echo MiniTwit is done.

Not that the "echo" commands are optional - they just output that text into the log file that is kept on the server instance. (In the event that something goes wrong and you need to look at the log file, this will help you find the relevant section with your startup script's output faster.)  In case you ever need to look at it, the startup log is located on the server here:

/var/log/cloud-init-output.log

Now finish configuring the instance (if you haven't already) and launch it. Go into the EC2 Management Console and find the instance's Public IP or Public DNS, then enter that in your web browser (and remember to add :5000 at the end). You should see the MiniTwit application running, and you never had to SSH in to this instance!

(Note that it takes a minute to start up. After the instance enters the "running" state, the OS still needs to finish booting, and then Flask needs to start up and run the web app. So if you can't connect to the MiniTwit app right away, you may need to wait another minute. It shouldn't take *too* long, though.)

You can find more information about user data and how to run startup scripts on your EC2 instances here:

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html

Part 5: Using the Relational Database Service (RDS)

Now that you have two instances each running MiniTwit, you can play around with both... you'll notice that they're both completely separate! Each one has its own completely separate set of posts, user accounts, etc. That's because each one is using a database that is running locally on the server, so they have independent databases with different sets of data stored. If we're going to use the cloud to scale out by creating more instances to handle more load, we want all those server instances to be running the *same* application, with the same posts, users, etc. What we need to do is to separate the database out of our EC2 instances so that all the web app servers connect to the same database, instead of each having their own.

We *could* use EC2 to run instances that have a database installed on them, so that we'd have two different types of instances: the ones running the web app itself using Flask (and no database), and the ones running the database only. However, the AWS Relational Database Service (RDS) provides us with a managed database service that runs in the cloud, which has several advantages: it easily scales, provides high availability and data resiliency, and we don't have to manage the servers ourselves (e.g., dealing with updates and patches to the OS and database software over time).

Here is the overview page for RDS:  
[https://aws.amazon.com/rds/ (Links to an external site.)Links to an external site.](https://aws.amazon.com/rds/)

and here is the getting started documentation:

[https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP\_GettingStarted.html (Links to an external site.)Links to an external site.](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.html)

To create a RDS database, go to the RDS service in the AWS Management Console. Choose "Instances" from the left menu, then click on the "Launch DB instance" button. You can choose which DB software you want to use - for this assignment, we'll use Aurora, Amazon's own database engine. Be sure you also select "MySQL 5.6-compatible" edition, then click the "Next" button. On the next page, choose the "db.t2.small" instance type. Then give your database a name (remember this is for the MiniTwit app), and choose a username and password for connecting to the database.

On the final page, choose the "Default VPC" for your Virtual Private Cloud (VPC) and "No" for public accessible (i.e., the DB should *not* be publicly accessible! Only your EC2 instances will be able to connect to it.). For the Availability Zone, you can choose "No preference" and for the VPC security groups, you should choose "Select existing VPC security groups" and then the default. In the Database Options section, give the database a name, and leave the rest of the options as their defaults. Everything after that can also stay at its defaults. Then click the "Launch DB Instance" button!

It will take a minute to start up, but once it does, take a little time to explore the RDS Console and see what it shows you about your new database.

There is also some documentation on starting new RDS instances here:

[https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP\_GettingStarted.CreatingConnecting.Aurora.html (Links to an external site.)Links to an external site.](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.CreatingConnecting.Aurora.html)

***Warning:****You are charged per hour for each RDS instance you have running. Just like your EC2 instances, you shouldn't leave RDS instances or clusters running for hours and hours without a good reason. Be sure to shut down your RDS instances and clusters when you're done for the day. (Yes, that does mean you lose your data, but that should be okay for class assignment purposes.)*

***Note:****Aurora databases don't currently support the "Stop" option. You will need to "Delete" them instead - this means all the data stored in them will be deleted. This probably isn't a big deal for the assignment, since we're just creating test data, not real users or tweets. However, if you do want to preserve your data, you can take a "snapshot" of your database, and then restore that snapshot when you start up new instances again.*

**Updating MiniTwit to use RDS**

Now you need to change the MiniTwit code to access the RDS database instead of the local one on the EC2 server instance. Since we chose the "MySQL 5.6-compatible" version of Amazon Aurora, we write our Python code to connect to it the same way we'd write Python code to connect to any MySQL database.

There is a quick overview here:

[https://stackoverflow.com/questions/372885/how-do-i-connect-to-a-mysql-database-in-python (Links to an external site.)Links to an external site.](https://stackoverflow.com/questions/372885/how-do-i-connect-to-a-mysql-database-in-python)

More detailed tutorials here:

[https://www.tutorialspoint.com/python/python\_database\_access.htm (Links to an external site.)Links to an external site.](https://www.tutorialspoint.com/python/python_database_access.htm)

and here:

[http://zetcode.com/db/mysqlpython/ (Links to an external site.)Links to an external site.](http://zetcode.com/db/mysqlpython/)

And official documentation here:

[http://mysql-python.sourceforge.net/MySQLdb.html (Links to an external site.)Links to an external site.](http://mysql-python.sourceforge.net/MySQLdb.html)

If you need to know more about Flask so you can understand the MiniTwit code, there is a quickstart overview here:

[http://flask.pocoo.org/docs/0.12/quickstart/ (Links to an external site.)Links to an external site.](http://flask.pocoo.org/docs/0.12/quickstart/)

If you SSH into your EC2 instance and go to the flask/examples/minitwit/minitwit directory, you will see two relevant files there: schema.sql and minitwit.py

* schema.sql - The database schema. You won't need to modify this, but you may want to look at what tables it's using and what fields are in each table.
* minitwit.py - The source code for the MiniTwit application. You will need to modify this code to replace the SQLite database accesses with MySQL database accesses, and connect to your RDS database instance.

To find the name of the connection endpoint that you should connect to, go to the "Clusters" section in the RDS Console. You should have just one cluster there, for your minitwit database; click on it. Find the "Cluster endpoint" - this is what you'll use to connect to this database. (Details on this are explained in the RDS documentation linked above.)

***You're not done yet, but your modified minitwit.py file is what you'll upload to Canvas to submit this assignment.***  So you may want to SCP a copy of it off your EC2 instance and onto a machine that you can access Canvas from. We will have a demo session where you can show me the rest of what you got working, including setting up your RDS database and EC2 instances with Auto Scaling and load balancing (the next part of this assignment).

**Saving Your Changes as a New Amazon Machine Image (AMI)**

Now that you've made you changes, you'll want to create a new Amazon Machine Image (AMI) that has your changes on it so that you can create new EC2 instances from that image and have your new code ready to run. Go to the EC2 console and select the running instance that has your updated code on it. From the Actions menu, go to Actions->Image->Create Image. Give it a name (and optionally, you may provide a description) and then click "Create Image". If you then click on the AMIs section of the Console (under the IMAGES heading), you will see your new image there with the status "pending" - this means your image is in the process of being created. (Warning: Do not terminate the EC2 instance you used to create this AMI while the image is still "pending"!)  Once the status changes to "available", your AMI is create and ready for you to use it to create new EC2 instances!  You can either select your AMI here and click the "Launch" button, or you can go to the EC2 Instances console, click "Launch Instance", then select "My AMIs" and you'll see your AMI listed there for you to choose from.

*Note: There is a very small per-GB charge for storing AMIs. It doesn't add up to much, but it's probably best not to keep around dozens of old AMIs that you aren't using anymore. Be sure to clean up your old, unused ones - but use caution that you don't accidentally delete on you still need, or that has the only copy of some significant change! Keeping just a couple AMIs around for the duration of the quarter won't be a big deal, though.*

If you need to make more changes later, you can always start from this AMI, make changes to your instance, and then create another new AMI. However, if you are working on your assignment and just need to pause for the day and come back to continue later, it may not be necessary to create an AMI for that. If you stop your instance (change it to the "stopped" state) then it will keep its data\* when you "start" the instance again and resume, and you do not pay for "stopped" instances. You should only need to create and AMI if either 1. you need to "terminate" the instance instead of just "stopping" it for some reason, or 2. you need to create more new instances based of the same image (hard drive state) as this instance.

\* This is only true for EBS volumes. Some instances have a local hard drive, which gets wiped if you stop and restart. The instances we're using right now do not have local hard drives, however; their entire filesystem is stored on an EBS volume. (A general rule of thumb for instances that do have local drive is to use the local drive only for temporary data, and keep anything you don't want to lose on an EBS volume, or in one of the other data storage / database services available.)

Part 5: Setting up Auto Scaling and the Elastic Load Balancer (ELB)

Before starting this part, read this documentation page so that you understand the end goal we're trying to reach by the end of this part:

[https://docs.aws.amazon.com/autoscaling/latest/userguide/WhatIsAutoScaling.html (Links to an external site.)Links to an external site.](https://docs.aws.amazon.com/autoscaling/latest/userguide/WhatIsAutoScaling.html)

**Elastic Load Balancing**

First we'll learn about load balancing. This can be useful any time you have multiple virtual servers sharing the load of an application, whether or not you are using Auto Scaling. (Although you can technically use Auto Scaling without load balancing, it *usually* doesn't make sense to do so.)  Start by reading these two documentation pages:

[https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/what-is-load-balancing.html (Links to an external site.)Links to an external site.](https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/what-is-load-balancing.html)

[https://docs.aws.amazon.com/elasticloadbalancing/latest/network/introduction.html (Links to an external site.)Links to an external site.](https://docs.aws.amazon.com/elasticloadbalancing/latest/network/introduction.html)

You may still have 2 or more EC2 instances running; if you do not, then start 2 up now. Now follow these instructions and set up a Network Load Balancer for the instances you have running (but finish reading these instructions first so you know what settings to use).

[https://docs.aws.amazon.com/elasticloadbalancing/latest/network/network-load-balancer-getting-started.html (Links to an external site.)Links to an external site.](https://docs.aws.amazon.com/elasticloadbalancing/latest/network/network-load-balancer-getting-started.html)

Name it whatever you want (but note that this name will be the first part of the domain name that users will enter in their web browsers).  Then follow the settings in the documentation instructions and the default settings, *except* for the following:

* Leave listener as TCP port 80
  + this will allow users to reach MiniTwit using the default HTTP port 80
* You should only have one VPC available to choose from right now
* Select *all* the Availability Zones
* Ignore the Elastic IP
* For the Target Group, also use TCP port 5000
  + because MiniTwit is using port 5000
  + with listener port 80 and target group port 5000, users will connect to port 80 but that will be routed to port 5000 on your MiniTwit servers

When you get to the part of the instructions that says to test your load balancer by putting it's DNS Name in your browser, you *do not* need to append :5000, because we set the listener port to TCP 80.

**Creating a Launch Template**

Before you can set up Auto Scaling, you'll need to create a Launch Template. A Launch Template stores all the settings that you normally enter manually in the Management Console when you launch an instance.  Launching an instance from a Launch Template is then a one-click process. Remember that we earlier learned how to make the MiniTwit app start automatically as well, so putting these together, we will be able to go from nothing to a running MiniTwit with no manual intervention.

Read Step 1 in the following Auto Scaling documentation (but don't actually create a launch template until you read the rest of the instructions that tell you what settings to use):

[https://docs.aws.amazon.com/autoscaling/latest/userguide/GettingStartedTutorial.html (Links to an external site.)Links to an external site.](https://docs.aws.amazon.com/autoscaling/latest/userguide/GettingStartedTutorial.html)

Before you start, you will the AMI ID of the image you created earlier, and the "Key pair name" of your key pair.

Now create a Launch Template using the documentation. You may choose a name for it yourself, but use the following configuration settings:

* AMI ID = the ID of the image you created
* Instance Type = t2.micro
* Key pair name = the name of your key pair
* Network type = VPC
* Add a Tag with the Key "Name" and a Value that is whatever you want the instances to be named; then be sure "Tag Instances" and "Tag Volumes" are checked
* (Note: You shouldn't have to enter anything in the Security Groups section; it will default to the default Security Group, which *should* be the one you created earlier.)
* Expand the "Advanced details" section and fill in the "User data" with the same script we used for the user data earlier - remember this is what makes the MiniTwit app start automatically when your instance boots up.

**Setting up Auto Scaling**

To recap what we have so far:

* an RDS database to store your MiniTwit data
* an updated AMI with your modified MiniTwit code to connect to the RDS database
* a Launch Template that lets you launch an EC2 instance from your AMI without any manual configuration steps, and User Data (set in your Launch Template settings) that automatically runs MiniTwit when the EC2 instance finishes booting
* a Network Load Balancer that provides a single domain name for users to connect to, and routes those user connections to the different instances in the associated Target Group

The final thing is to set up Auto Scaling so that it automatically scales out/in with the load on the MiniTwit servers, and connects all the new MiniTwit instances to the Load Balancer so that new instances when it scales out automatically start getting used.

Start by going to the Launch Templates section and select your launch template. Then choose Actions->Create Auto Scaling group.

Now give your Auto Scaling Group a name. You might be able to leave Subnet blank, but if it complains about that then just add all the available subnets. Now expand the "Advanced Details" and check "Receive traffic from one or more load balancers". Leave "Classic Load Balancers" blank (the load balancer we created is no the "Classic" type), but in "Target Groups" select the Target Group you created with your load balancer. Finally, choose "ELB" for the Health Check Type - this will cause Auto Scaling and the Elastic Load Balancer to agree on whether any server is down or not - then click the "Next" button.

On this page, we will select "Use scaling policies to adjust the capacity of this group". Tell it to scale between 1 and 3 instances, for this assignment. Give the Scale Group a name, and then set the Metric type to "Average CPU Utilization" and Target value to "80" (for 80%). This means that if server load is about 80% CPU Utilization, more servers will be started; if it is less, some servers will be stopped (to a minimum of 1 and maximum of 3). Now click "Review".

Double check your settings and then click "Create Auto Scaling group".

Now test out your MiniTwit app, and look around the different sections of the Management Console to see what all the pieces you have are.  Understand all the pieces and how they work together.  (Be prepared in a demo to answer a questions like, "What is the relationship between A and B?")

***Warning:****Auto Scaling automatically creates EC2 instances for you. You are billed for the instance-time you use, so be careful of leaving Auto Scaling running after you're done working!  If you shut down all your instances, but leave Auto Scaling running, it will just start up more!*

Submission

Submit your modified **minitwit.py** code on Canvas (as an attached file). I will be in touch about the procedure for signing up for a demo time. In the demo, you will show what you configured in AWS (including the ELB and Auto Scaling), and answer some questions to show that you understand what you did. (So don't just follow the instructions above without thinking about it! The point is to learn so that you can use the tools on other projects in the future.)

Rubric