# CS5412 - Assignment 2: Using Amazon's EC2 and Elastic Beanstalk

## Objective

The objective of this assignment is to learn how to use EC2 and Elastic Beanstalk and learn about the pros and cons of utilizing tools like Elastic Beanstalk. In particular, you have to implement an elastic web server that acquires/releases computer nodes according to the current workload and a client in EC2 that issues request at this server with a high rate in order to observe the performance of the server under pressure.

## High Level Description

The Cryptography Lab that you are working for is short in computer machines. From a profiling procedure that you have previously done, you have discovered that 27% of the time, the computer machines in the lab run Primality Tests (`en.wikipedia.org/wiki/Primality_test`). Of course, this is not suprising given that you are in a Cryptography Lab. As an ex CS5412 student, you immediately step up and propose to use the Cloud in order to run Primality Tests. You argue that:

- You would not have to pay a huge amount of money immediately in order to buy new machines.
- The Cloud could easily handle the bursts that you observed in the profile test during afternoon.
- You would shut-down the machines (save energy and minimize cost) during the night when no-one runs Primality Tests.

Your colleagues are impressed and they immediately assign to you the task of porting an application that handles Primality Tests in the Cloud.

## Web Server with Elastic Beanstalk

Elastic Beanstalk is a service in AWS that provides elasticity to web applications with minimum programming effort from the users. The users need to provide their web application (server) and policies about when to add/release computer nodes. Then, Elastic Beanstalk runs your web server in AWS nodes, enforces your elasticity policies and automatically balances the workload to computer nodes. You should find more information in the available documentation from Amazon (`aws.amazon.com/documentation/elastic-beanstalk`).

Your work is to implement a web server that should handle the following tasks:

1. Register (username, netid, password): You should check that there are no previous users with the same username or/and netid.
2. Login (username, password): You should check that the username exists and the password matches the username.
3. Query (number) : Users should be "logged-in" in order to query. They should be able to input a number and the server should determine if the number is prime or not.
4. Logout : The current user is no longer "logged-in".

You are free to implement the web-server any way you want as long as it supports the above operations. Finally, you should port the server to Elastic Beanstalk and have a running server there. Notice that in case you run in multiple

instances you need to have shared space (hard disk or relational database) in order to store the passwords. Otherwise, the view of passwords that your instances have would not be consistent with each other. Elastic Beanstalk provides some options for this purpose (make sure to investigate before you try any solution).

All the server nodes should be *t1.micro* instances.

ATTENTION: You should check your options for implementing the server because Elastic Beanstalk does not support all the possible ways to implement a server.

HINT: Debug locally first, and then try to port the server in Elastic Beanstalk.

HINT: You can also utilize the worker queue approach that Elastic Beanstalk offers. It is not necessary though.

## Client with EC2

Program a client to make automatic sequential queries to the web-server you implemented. In particular, the client should take as input the following parameters:

- server url (url)
- username (string)
- netid (string)
- password (string)
- starting number (int)
- ending number (int)

and do the following operations:

1) Register with the username, netid and password.
2) Connect to the server with his/her credentials (login with username and password).
3) Perform queries for all the numbers between the starting and the ending numbers (you can skip even numbers). The queries should be sequential (meaning that you will make a query after you have a response from the previous one).
4) Logout.
5) Report the minimum, mean, median, maximum latency for the queries the client made.

All the client nodes should be *t2.micro* instances.

HINT: After you implement the client, you should try to test it with the web-server. Then, make sure it works and port it to AWS EC2.

You should find instructions here (`aws.amazon.com/documentation/ec2/`).

## Experiments

In the first experiment, the web server should consist of a unique computer node. Configure Elastic Beanstalk to run only one computer node (1-1). You should run Client One from:

a) Your personal computer.

b) A computer node in a different region in Amazon EC2 than the one your server runs.

c) A computer node in a different availability zone (but same region) in Amazon EC2 than the one your server runs.

d) A computer node in the same availability zone in Amazon EC2 than the one your server runs.

You should report the output (latencies) of these four clients and your explanation about the results.

In the second experiment, the web server should be configured to utilize more than 1 and up to 4 computer nodes (1-4). Additionally, you should configure Elastic Beanstalk to add a node when the CPU load is more than 80% and remove one when it is less than 50%. Finally, deactivate cross zone load balancing (it is ticked by default). Feel free to adjust the other parameters of the configuration as you find fit.

Instead of running one client now you should run 4 clients (2a, 2b, 2c, 2d) in the same availability zone (Amazon EC2) as the server. Answer the following questions:

a) What is the latency and the throughput (requests/sec or requests/min) that you expect to get (ideally) compared to the 4th client scenario in the previous experiment?

b) What do you actually get from the experiment? Report the diagrams that Elastic Beanstalk produces for Average Latency and Sum Requests and the output of your four clients (latencies). Try to explain the results.

c) Is the server protected against a Distributed Denial of Service attack (DDOS attack) that tries to exhaust the resources of the server by making queries? If yes (no), why (why not)?

d) Is the server protected against a Distributed Denial of Service attack (DDOS attack) that tries to exhaust the resources of the server by trying to make new connections? If yes (no), why (why not)?

e) Is this approach satisfying when it comes to handling bursts of queries that last a few minutes? If yes (no), why (why not)?

The inputs to the clients 1,2a,2b,2c,2d (username, netid, password, starting and ending numbers) will be available 72 hours before the deadline in a file called **inputs.txt**.

## Submission

Your submission should include the following items, in a .zip file:

1. Write-up of your project in pdf format, answering all the questions asked in previous sections. You should call this file **writeup.pdf**.
2. The source code of your project. You can use any of the options given in Elastic Beanstalk for the web server. The client code can be in any language you want.
3. Documentation for how to compile and run the program in plaintext format. If your project requires installation of dependencies in order to run, you should provide instructions on how to do so in this document. Call this file **README.txt**.

In the end, you should have a directory structure that unzips like this:
\submission
\submission \writeup.pdf
\submission \README.txt
\submission \src *