D7001D Network programming and distributed applications – LAB4

Prepared By: Ahmed Afif Monrat, Syed Asif Iqbal, Daniyal Akther

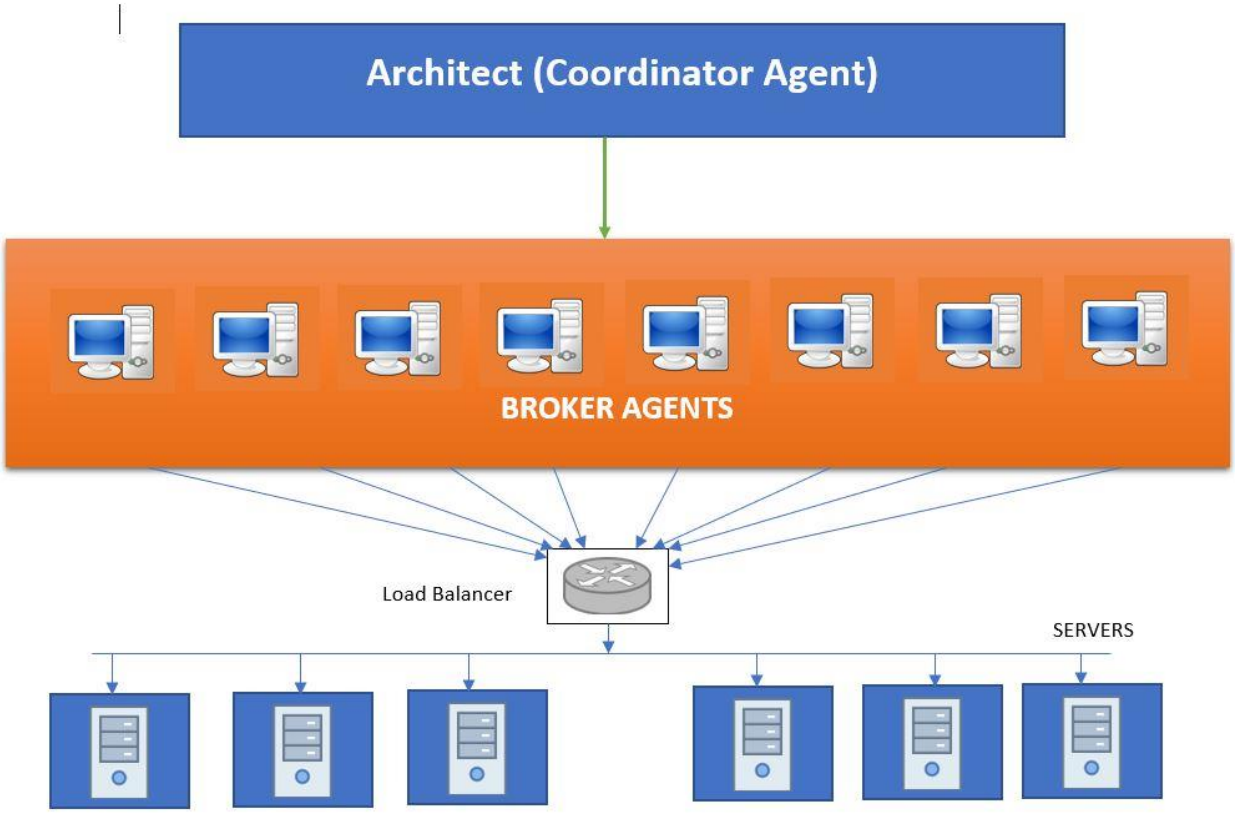
Objectives:

1. Create a multi-agent system, which in a coordinated manner performs actions to achieve certain goal.
2. Create a scalable server architecture, which dynamically adapts to an increasing load and study its properties.

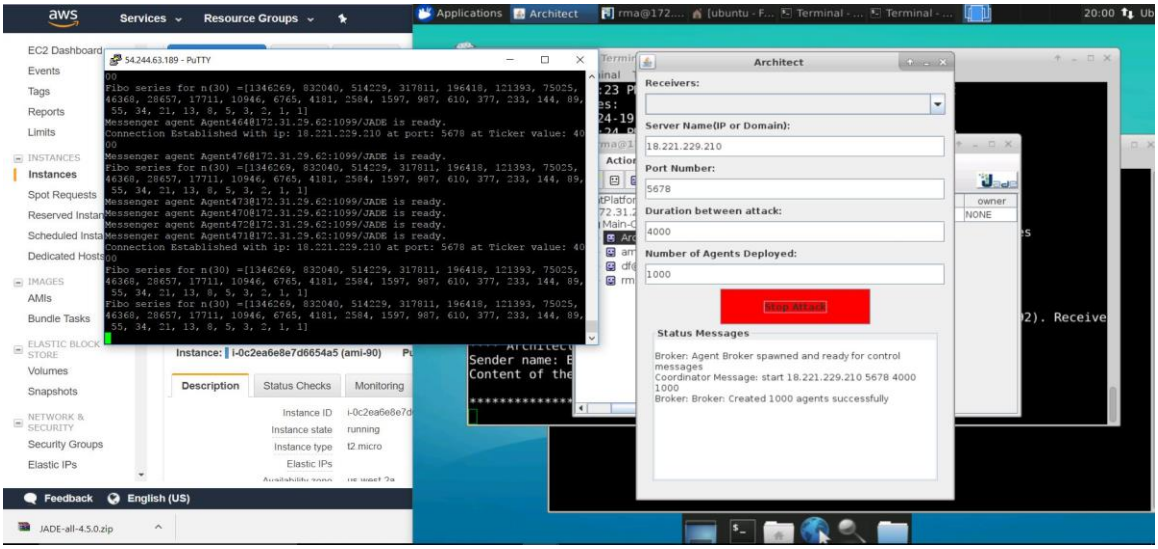
System Implementation

We have created a multi-agent system in which there is one Coordinator Agent (The Architect), which has a GUI interface through which one should be able to control the population of attacking agents and their parameters. The Architect (Coordinator) Agent creates an army of Soldier Agents (Agent Smiths). Agent Smith has a Ticker Behavior that periodically performs certain action. The duration of the period is a configurable parameter. The action, which Agent Smith performs is opening a TCP socket towards a TCP server.

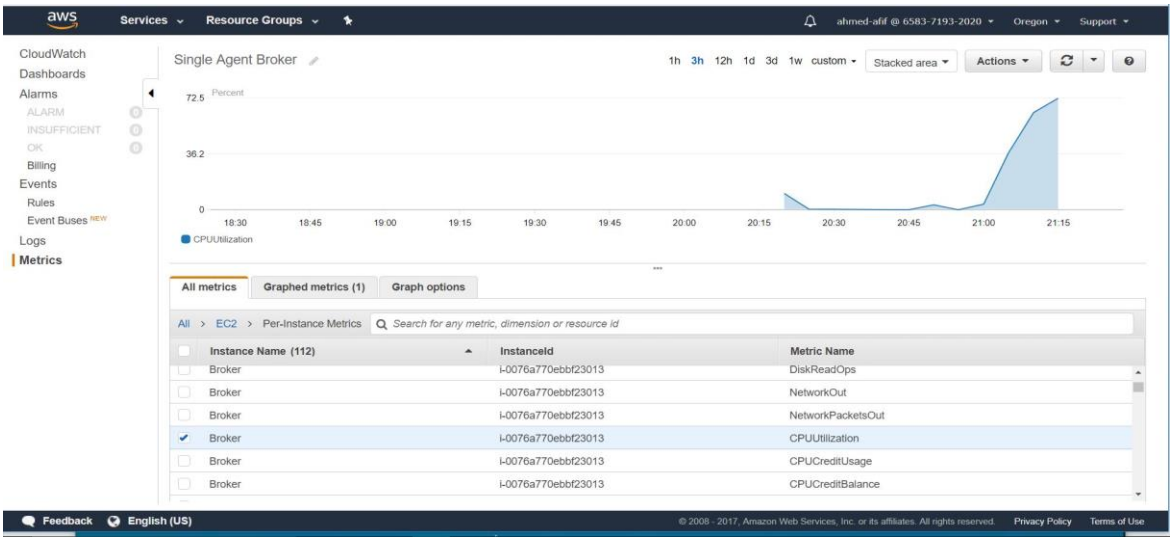
System Architecture



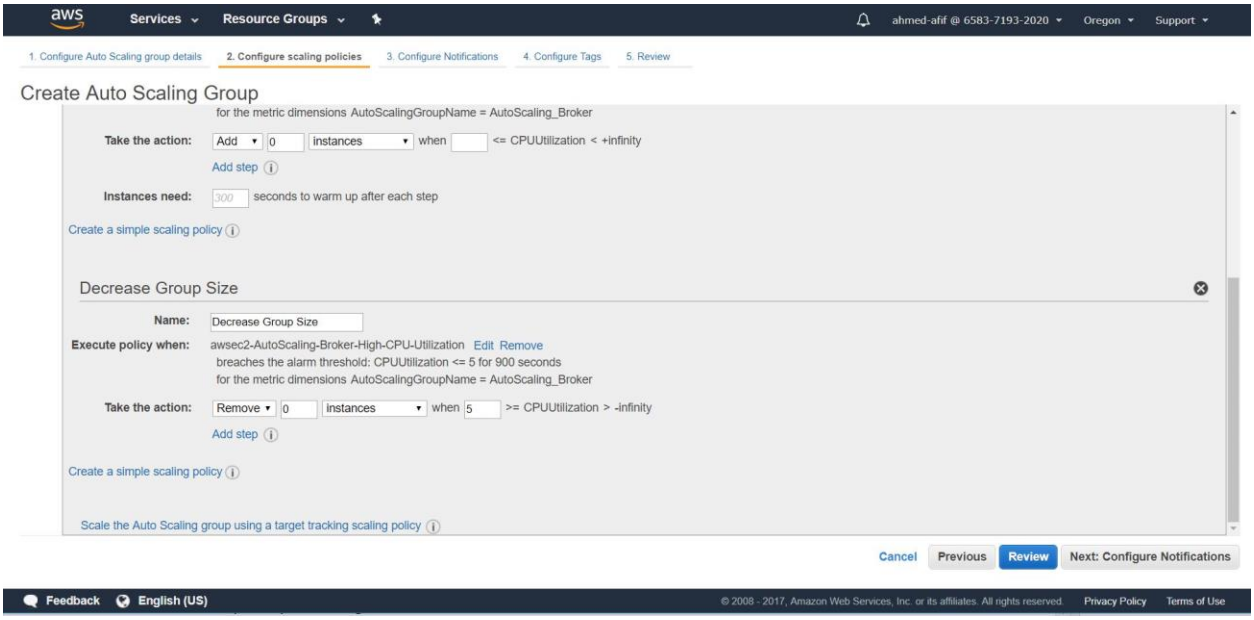
While designing the architecture, we consider master-slave architecture where the master will be the Coordinator agent that can creates several agent brokers which will create numerous number of agents to perform the specific task, that is to attack a server. We have used three AWS account for Coordinator, Agent-Broker and the server respectively. Coordinator will command the broker agents to create more agents through a GUI interface. The Broker machine will get the parameters for the server IP to attack, port number, ticker duration and the number of agents need to create from this Coordinator GUI. After this it will create agents and initiate the attack. To make this architecture scalable we have used auto scale group in AWS account for Broker-Agent. When the Coordinator will give command the broker agents will start to create agents and the auto scaling group has a threshold for CPU utilization as if its more than 70% of its capacity it will automatically add one more instance to create and launch the attack. However, when the rate of attack will be minimal that is if the CPU utilization goes down to 5%, it will remove 2 servers. This feature makes this application more scalable and reliant as it will provide single node failure tolerance and has the environment setup to launch more attacks through multiple servers.



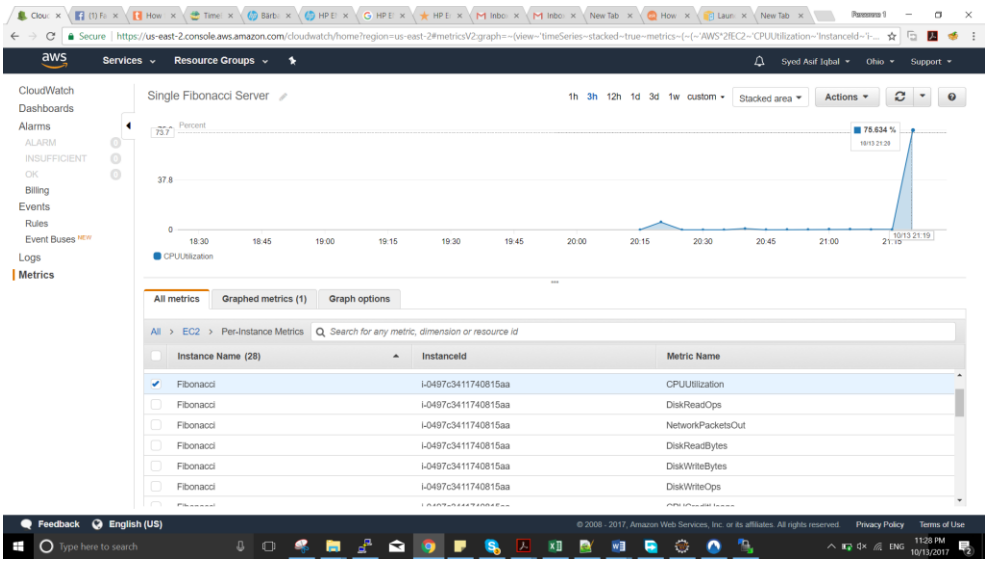
By using AWS CloudWatch, we can see the behavior patterns of the agent's attack with respect to CPU utilization.



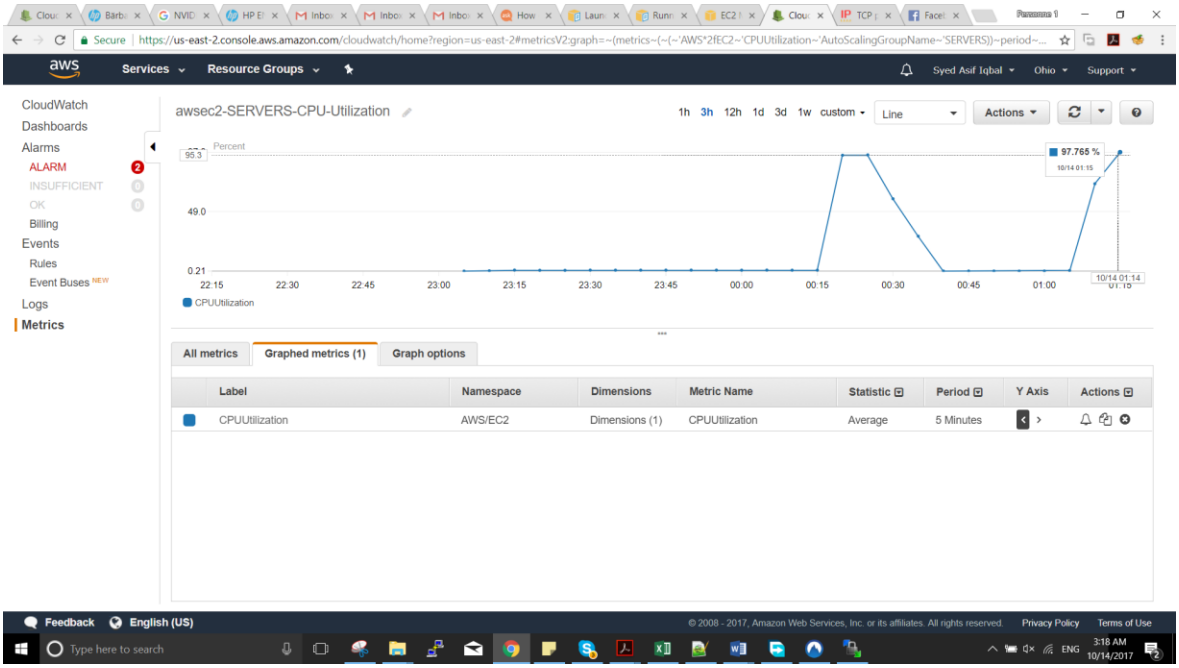
For server part, from the architecture we can see that, there is a load balancer which will redirect the http request that's coming from different agents or user to the server. It will save the server from getting too much task to handle at single instance. All the attacks from agent broker will come through this single point link (Load balancer DNS) and then distributed to all server nodes. As a result, the server will never go down and as it will proportionally distribute the tasks among several servers. There will be an auto scaling group for the server part which will deal with scalability problems. There is threshold to initiate more servers automatically if needed and we can also see the trends of CPU utilization using CloudWatch here.



Basically, the agents are requesting the server to process Fibonacci series during their ticker time which might be 3000-5000 milliseconds.



While monitoring and measuring the server performance, we tried different no. of agents to perform the tasks within different ticker time and observed the performance of the MAS. We provided up to 10000 agents to launch the attack. When the number of agents was less the server performed efficiently. When we increased the no. of attacks it took time to initiate more servers to process the load yet it performed quite good when all the added servers was up and handling the load.



Observations:

1. From this project we came to know how distributed system works and why its beneficial for complex large computing process.
2. The system architecture was able to handle the challenge to process large no. of computing processes using scalable features.
3. Fault tolerant.