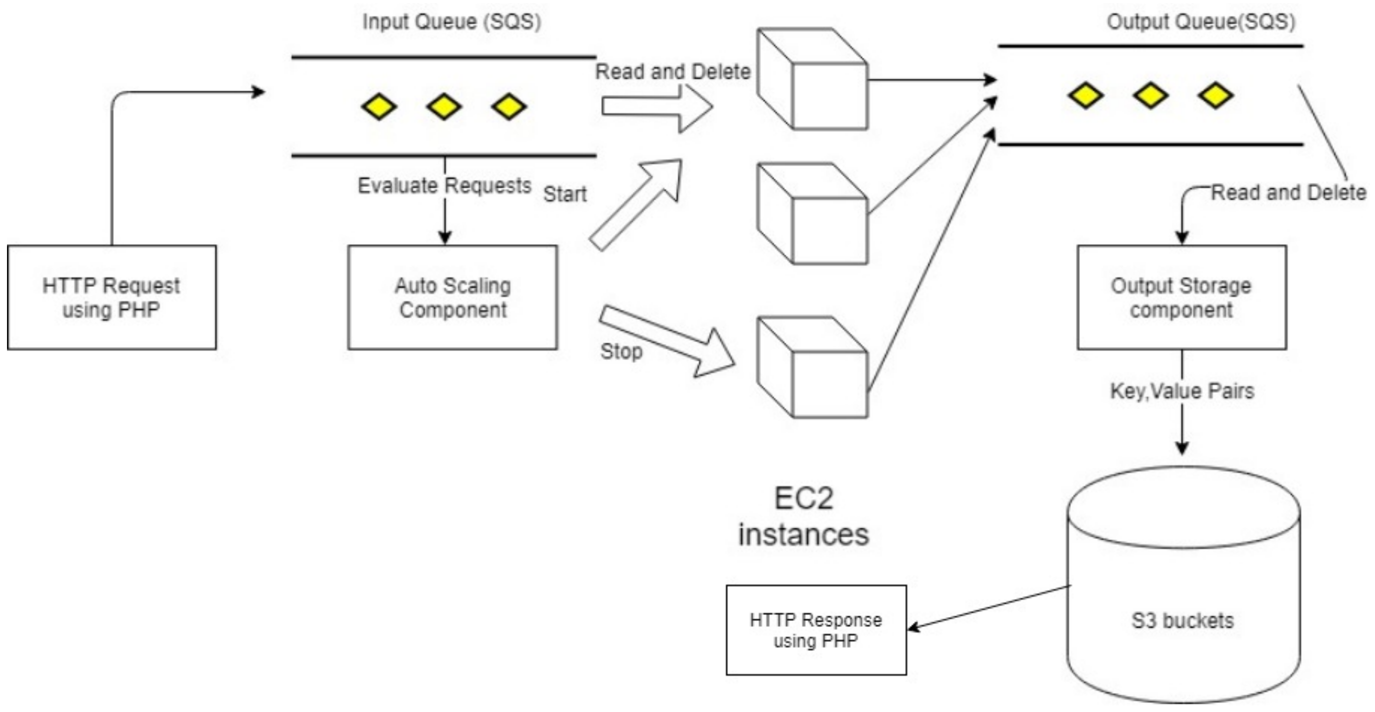
CSE 546 — Project 1 Report

Sachin Sundar Pungampalayam Shanmugasundaram – 1213230978
Satheesh Kurunthiah – 1212901792
Raj Sadaye – 1213175416
Shubham Gondane – 1213179615

1. Architecture



Description of components:

1. **HTTP Requests:**
   A PHP application deployed on apache server in EC2 instance takes input as the URL of the image to be used for image recognition and enqueues it into an SQS queue. A sample request can be as follows: http://18.144.21.65/cloudimagerecognition.php?input=https://images.pexels.com/photos/20787/pexels-photo.jpg?auto=compress

2. **Input SQS Queue:**
   The request is stored as a JSON object with 2 keys:
   ID: a randomly generated id for the object
   Input: contains the URL of the image provided
   This input queue is used by the autoscaling component for upscaling and downscaling the number of running instances based on the size of the queue

3. **Autoscaling component:**

   Uses CloudWatch to monitor the CPU utilization of running instances. Based on the CPU utilization and the number of request in input queue it decides whether to increase the number of instances running (upscale) or decrease the number of instances running(downscale). This is a crucial component of the system since it ensures that there is optimal resource utilization and helps us to save money since we are not keeping instances running without any reason.

4. **EC2 instances:**

   The autoscaling component launches snapshots of the same EC2 instance whenever load increases. Inside every EC2 instance is a java code that polls that continuously polls the input queue for request. The JSON object inside the queue is read by the java code and the URL of the request is extracted.

   Further, these 3 commands are run on the command line of the instance using ProcessBuilder:

   ```
   source ~/tensorflow/bin/activate
   cd /home/ubuntu/tensorflow/models/tutorials/image/imagenet
   python classify_image.py --image_file $URL --num_top_predictions 1
   ```

   Further the output of the command is captured in a string and is written to the output SQS queue as a JSON object with 2 keys the id and the output of the recognition. The jar is called using a shell script that runs whenever the instance is launched.

5. **Output Storage Component:**

   This component processes the output queue. It reads the JSON objects from the output queue and stores them as <Key, Value> pairs in S3 buckets.

2. Autoscaling

We are monitoring the input queue for the approximate number of messages present in the queue. In the beginning when the queue is empty there will be just one controller instance running. So, we create another instance which will be running alongside the controller.

Scale up:

1. When requests are populated in the queue depending on the number of requests and the number of currently running instances we create new instances such that the total instances limit of 20 is not exceeded.
2. We check if the number of currently running instances is less than maximum instance limit and if the instance count is less than the number of requests in the queue. If this condition is met we create a new instance.

Scale down:

1. When the number of running instances goes higher than the number of requests in the queue we scale down by terminating some instances. The instance which needs to be terminated is decided by its average CPU Utilization.
2. We are using the Amazon Cloud Watch service which monitors the instances in a window of 5 minutes. If the CPU Utilization of the instance falls below 5 percent in that window, then we terminate the instance.

So, every 10 seconds we are getting the new count of running instances and number of requests in the queue to decide whether to scale up or scale down.

3. Code

Cloudimagerecognition(PHP):

This PHP script primarily performs 2 functions:

1. Takes input via HTTP request and send it as a message to the input queue
2. Returns a response by accessing the S3 buckets.

It accesses the Input Queue by creating an SqsClient. Similarly, it accesses the S3 bucket using an S3Client for returning the response.

The request is made using: http://18.144.21.65/cloudimagerecognition.php?input=URL

The PHP script creates an object called $req and encodes it as JSON Object containing a randomly generated id and the URL. This object is then pushed onto the input queue. Further, it waits for a response from S3 and then decodes the response to get the recognition output.

Auto scaling part:

- **getMessageCount** - Returns the approximate number of requests in the input queue.
- **getInstances** - This method gives the instances which are either in running or pending states. We filter the instances based on the ami id which is different from the controller instance's ami id. Returns a list of instances
- **getInstanceAverageLoad** - This method gives us the average CPU Utilization of a particular instance. We are using the Cloud Watch service to get the CPU Utilization. The average CPU Utilization is returned by this method
- **createIntance** - This method creates new instances and returns the instance id
- **Terminateinstance** - This method terminates instances
- Scale - This is the method which does the autoscaling of instances. It takes into account the number of requests in the queue and number of instances currently running to determine if we need more instances. It calls the createInstance method to create a new instance. Then using the instances count and the request count it checks for the average CPU Utilization of all the instances to determine which instance needs to be terminated

Process message:

This function uses 2 SQS queues: Input_Queue and Output_Queue. The Queue URLs for both these queues are fetched using the sqs.getQueueUrl command. Eg: private static String inputQueueUrl = sqs.getQueueUrl(INPUT_QUEUE).getQueueUrl().

The receiveMsg function is called to read the messages from the input queue. A while loop runs continuously so that whenever the machine is idle, it will try to process a request from the queue. If there are no messages in the queue, then it waits for 5s and polls the queue again. Once it receives the message it will decode the message from JSON to a java string. Further a string variable TENSOR_PATH is used to store the path to the shell script that runs the jar file that will be stored on the instance.

The shell file takes in the URL of the image as a command line argument. Hence a string called as 'command' is created by appending the TENSOR_PATH and the URL of the image. The image file name is extracted from the URL and stored in a string variable 'imageName'. Then executeScript function is called which takes in the 'command' and returns the recognition output given by the deep learning model inside the EC2 instance. Inside the executeScript function, it creates a process and executes the command on the runtime of the instance. It waits for the command to execute and reads the output from the command line using a BufferedReader object. Finally, the BufferedReader object is converted to string and then returned.

Then the sendMsg function is called with id, imageName and output as the parameters. The sendMsg function sends a message to the output queue containing the recognition result as a JSON object.

OutputQueue:

Performs these tasks:

1. Read the messages from the output queue.
2. Process it and put it in a S3 bucket.
3. Once object is stored in the bucket, delete the message from the queue.
4. If some error occurs, do not delete the message so that it will be processed again.
5. If the approximate number of messages in the queue is greater than 10, keep on reading
6. messages from the queue.
7. If it is less than 10, sleep for 10 seconds before reading the queue again. This will reduce the polling of empty or small queue.

5. Project status

All required specifications in project description are met and working as expected.

- Requests are received via HTTP and Image Recognition service is provided via public IP address http://18.144.21.65/cloudimagerecognition.php?input=URL
- Application will parse the input URL and invoke the image recognition on the received image URL
- Application will handle multiple concurrent request by scaling up / down based on number of requests
- Results are stored in Key, Value (Image name, result) pair in S3

5. Individual contributions

5.1 Satheesh Kurunthiah – 1212901792

Responsibility

- Implement PHP application that will handle incoming request from outside world and send the result as response
- Final integration of all 4 components (PHP, Instance manager, Process message, Store message) in EC2 instance

Design

During initial discussing for setting up of server that can handle request several methods like Node JS and python were discussed and finally PHP is chosen because deploying it in apache server is more convenient than others. Also, who should delete the processed message from the Output queue were discussed like should the PHP application should do, or Store message application should do. Finally, it was decided that Store message should do as it had once advantage over the other that once a request is processed and stored in S3 then PHP application need not wait for entire roundtrip of the request rather can respond more quickly.

Pre-requisites

Apache server has to be installed in EC2 instance so that the PHP application can be deployed on top of it. In order to use AWS functionality AWS-PHP SDK has to installed in the EC2 instance as well. Autoloader functionality of PHP is used so that all required 3rd party vendor for the AWS-PHP to run will be downloaded based on usage. AWS Client has to be configured in both EC2 instance with valid credentials from Cloud user group.

Implementation

The PHP application, for each request will generate a unique random id and parse the image file URI and send it as a JSON message into the Input Queue. It then waits until the request has been processed by remaining three components and result is available in S3. In order to allow incoming messages in EC2 security groups have to be added. This include Custom TCP which allows messages through port 80 from anywhere. The remaining three components are made as executable JAR and deployed in the server as services so that whenever an instance is launched the program start running without someone having to do it manually. In order to do this a script file is written which will in turn set the JAVA path and launch JAR file using Java -jar <file-name> command. This script has to placed inside /etc/init.d/ folder and its settings has to made to defaults by running update-rc.d command. This will make the script to run automatically when the system is booted. Two snapshots were taken. One with Image Recognition and Process Message application, other with Instance Manager, PHP application and Store Message application. These two instances will run even if there are no messages are in Input Queue.

Testing

PHP application while developing was tested using built in PHP standalone server in localhost. CURL was used to send HTTP GET request to the PHP application and the results are validated with Tensor flow output. After integrating all components, the entire round-trip testing was done using the test scripts provided in the class.

5.2 Sachin Sundar Pungampalayam Shanmugasundaram – 1213230978

Responsibility

Implement a Java application that reads the output messages from queue and stores it in S3 bucket as a key, value pair.

Design

The output SQS queue will store the result from the image recognition instances. The messages are stored in JSON format with id, input and output fields. The Java application will subscribe to the queue and start reading the messages. For each message, it will store the input and output in S3 bucket as a key, value pair. If an error occurs during processing the message, the message is not deleted so that it will be processed again; else, the message is deleted from the queue.

Implementation

The storeOutput class handles reading the queue and storing the output to the S3 bucket. The queueUrl of the output queue where the image recognition instances store the output is used to access the queue. For reading messages from the queue along with the queueUrl, “MaxNumberOfMessages” is set to 10 so that each time a read request is made, the maximum number of message is greater than 1 and less than 10(queue reading limit); “WaitTimeSeconds” is set to 10 seconds. This makes the read request to wait for a maximum of 10 seconds before it returns an empty response. For getting queue attributes, set the attribute “ApproximateNumberOfMessages” which returns an approximate number of messages that are present in the queue.

Once all parameters are set, start reading the messages from the queue. For each message which is in JSON format, get the values of id, input and output fields. Store the value of the id field in a Map so that duplicate messages from SQS can be ignored. If the message is new, store the input and output in S3 bucket as a key and value pair respectively. Once the message is stored in the S3 bucket without any error, delete the message from the output queue using the message ReceiptHandle. If any error occurs before storing the message, do not delete the message from queue, so that it will be processed again and stored in the S3 bucket.

To reduce the number of empty read requests to the output queue, check if the approximate number of messages present in the queue, is less than 10. If it is not less than 10, there are more messages present in the queue, so continue reading the messages and storing them; else make the process to sleep for 10 seconds so that the queue might get some messages. Doing this will reduce the number of empty read responses.

Testing

Unit testing of the storeOutput was done for various scenarios. With no messages in the queue where the application waits for 10 seconds before the read response is sent and sleeps for 10 seconds as there is no message in queue. When the queue had less than 10 messages, the read request returned the messages without delay and after processing, slept for 10 seconds. When the queue had more than 10 messages, the process executed without any wait or sleep it processed all the messages.

5.3 Raj Sadaye – 1213175416

Responsibility

- Implement a module that reads from the input SQS queue
- Implement a module that runs the image recognition command on the virtual machine
- Implement a module that writes output of image recognition onto the output queue
- Store the snapshot of instance

Design

- All three modules were implemented in Java since it is convenient to run a jar on the virtual machine whenever the instance is launched
- The jar was called using a shell script which was stored on the instance.
- Whenever a new instance is launched, it has the same contents as that of a snapshot that has the shell script and the jar

Pre-requisites

- Java should be installed on the instance
- The java code required two external jars:
    1. java-json.jar
    2. json-simple-1.1.jar
- AWS Client has to be configured in both EC2 instance with valid credentials from Cloud user group.

Implementation

- The code inside the jar reads JSON objects from SQS queue and decodes them to find the id and input URL. A while loop polls the queue for messages
- A process builder object runs the commands necessary to call the image-recognition python file using URL as the parameter. This will run on the command line of the EC2 instance.
- Another process captures the output for the image-recognition task using Stringbuilder object.
- The id of the image and the output for image recognition is written to the output SQS queue as a JSON object.
- The jar is uploaded to the EC2 instance. Also, a shell script is written to call the jar file. The shell script is places inside the init.d folder so that it keeps running when the instance is running

Testing

PHP application while developing was tested using built in PHP standalone server in localhost. CURL was used to send HTTP GET request to the PHP application and the results are validated with Tensor flow output. After integrating all components, the entire round-trip testing was done using the test scripts provided in the class.

5.4 Shubham Gondane - 1213179615

Responsibility

Implement the autoscaling part that will monitor the input queue and dynamically start or stop instances based on the number of requests in the queue.

Design

Initially we decided to go with CloudWatch service provided by the AWS to monitor the average CPU Utilization of the instances. CloudWatch provided different time frames over which we could monitor the instances like 1 minute or 5 minutes. We also had the flexibility of choosing how many time periods do we need to watch before deciding to take an action.

Implementation

The entire code for this task is written in java. So, the two main things to handle were the queues and the instances. To get the approximate number of requests in a queue a method was created that takes the queue URL as the input and gives the number of requests in the queue as the output. Now for the EC2 instances part I needed to create instances, monitor them and terminate them. A method was written for creating instances such that all the instances created would have the same AMI id. The method returned the instance id to store it in a list of instances. To monitor the instances, I had to get the number of instances in running state or the pending state. For each instance, the average CPU Utilization was returned by another method that used CloudWatch for the same. Similarly, to create instance method a terminate instance method was implemented that terminated the instance whose instance id was passed to the method.

Now in the method that performs autoscaling there is a loop that runs always, and it checks for the number of requests in the queue and the number of instances currently running every 10 seconds. So, that if the number of requests in the queue is greater than the number of instances running then and if the number of instances is less than the maximum allowed we spawn a new instance. Also, if the number of requests is less than the number of instances running we terminate the instances based if the CPU Utilization is less than 5 percent.

Testing

Initially we wanted to use CloudWatch to terminate the instances but at the end decided to implement our own code for instance termination. But we faced some challenges during this part because we were using alarms to that would terminate the instances if the CPU Utilization falls below certain threshold. But the alarms required that the CPU Utilization remain below threshold for at least 1 period of 5 minutes to terminate the instance. So, we decided to go with our own code to terminate as soon as CPU Utilization falls below 5 percent. The autoscaling part was tested using the test scripts provided in the class.