CSE 546 — Project 1 Report

Project Team Members
Aditya Nagesh Govardhan (1215374199),
Jay Shah (1215102837),
Kunal Vinay Kumar Suthar (1215112535)

1. Architecture

The available AWS services used that we have used in building this service are:

1. **Amazon EC2**
   EC2 allows us to create instances of virtual machines on the AWS infrastructure and also allows us to scale-in and scale-out to process a large number of requests in the same amount of time.
2. **Amazon SQS**
   We use SQS queues for sending and storing messages between the *WebTier* and *AppTier* instances. It enables the WebTier instances to send a message to the queue and the *AppTier* instance or any of the *AppTier_Terminator* instances can pick it up for processing and hence support the task for running in a loosely coupled manner.
3. **Amazon S3**
   We are using S3 because it is a scalable storage device where we store the results of the video detection in the form [key, value] received from the *AppTier and AppTier_Terminator* instances.
4. **Amazon VPC**
   Amazon VPC is used for defining the secure private network of the project. It is also configured along with internet gateway, subnet and security group. This helps in configuring the incoming and outgoing traffic of the network in which the infrastructure is deployed.
5. **AWS IAM**
   IAM is used to create *service-roles* for EC2 instances which help them access other AWS resources like SQS, S3 and EC2 itself. This minimizes the maintenance of AWS credentials on developer side since the temporary credentials are auto-configured.

[Figure 1] below shows the basic outline of the architecture we have implemented here for video detection as a service application.

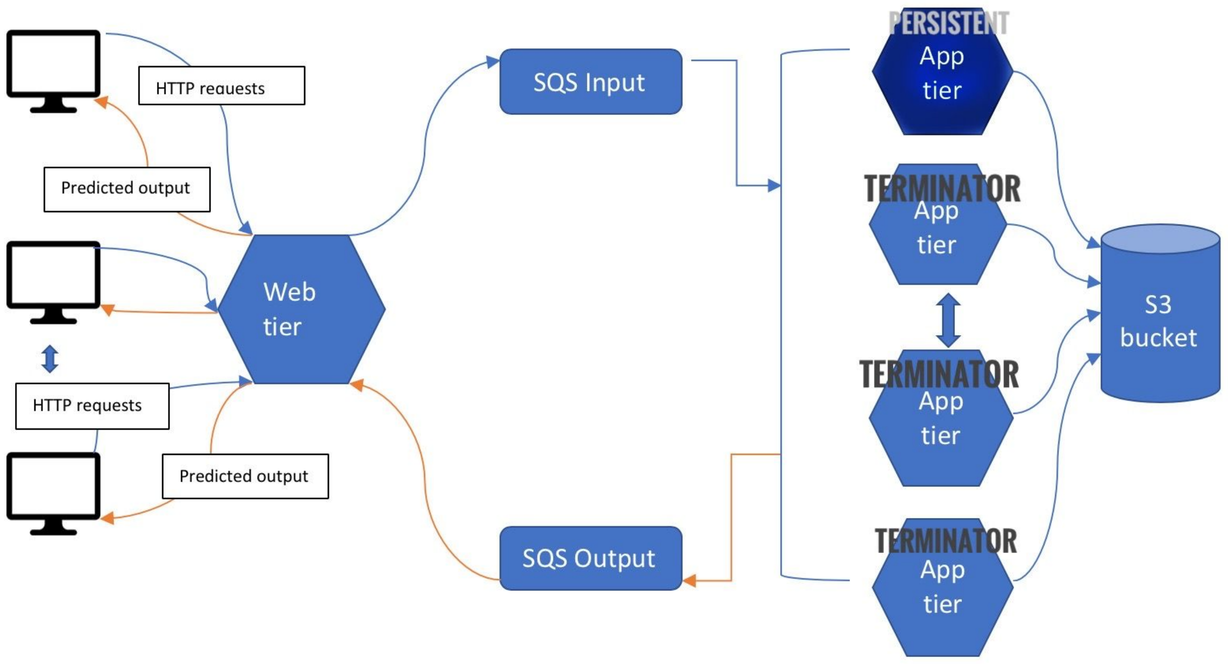


Figure 1: Overall architecture of the cloud service

Initially, we have only WebTier and AppTier running in order to receive & handle requests from the user. These instances have been initialized using a script that executes all the required instructions to set up the required infrastructure. The requests from the user would be handled by the WebTier instance which puts the received request in the INPUT QUEUE to be sent to the AppTier and AppTier_Terminator instances. The AppTier and AppTier_Terminator instances will run the given Object Detection Model in order to predict the objects from the video which is downloaded in the very instance using the trigger received from the WebTier via INPUT QUEUE. Once the labels of the video are predicted, the AppTier and AppTier_Terminator instances push the labels into S3 and also in the OUTPUT QUEUE.

Framework Used: The framework which we have used for the RESTful application is the Java Spring Boot framework.

Components:

(i) Web-Tier:

Our Web-Tier is an EC2 instance which takes in an HTTP GET request from the user when the user clicks on a request button on an HTML page, which is hosted on a Tomcat Server using the SpringBoot framework. Whenever it receives a user request, they are sent in the SQS-input queue(which is being constantly heard by the Persistent-App-Tier or the Terminating-App-Tiers). The Web-Tier is also responsible for performing the scaling-out functionality of the Auto-Scaling feature, i.e. it clones Terminating-App-Tier instances whenever the number of user requests increases, to provide fast service to the users. The Web-Tier is also responsible for listening to the SQS-Output queue(which contains the output string) constantly. Whenever there is an output in the SQS-Output queue, it will parse the output and return it on the user's browser screen(done using SpringBoot framework).

(ii) SQS-Input queue:

The SQS-Input queue stores the user request for object detection. The user request is a string message with content as "*Sending a request for object detection*". This message acts as a trigger which asks the Persistent-App-Tier/Terminating-App-Tier instances to download the video and run the object detection module.

(iii) Persistent-App-Tier:

The Persistent-App-Tier instance constantly listens for trigger messages in the SQS-Input queue. The Persistent-App-Tier EC2 instance takes in the message from the SQS-Input queue, which is sent by the Web-Tier instance, downloads the video from the Raspberry Pi cluster through an HTTP request and performs the object detection task using the deep learning model. This instance then pipelines the output to the SQS-Output queue and stores the output content in an S3 bucket. However, this particular instance does not terminate itself and is always running from the initiation of the application to allow a fast response to the user requests.

(iv) Terminating App-Tier:

These EC2 instances essentially provide the same functionalities as the Persistent-App-Tier instances with the additional capability of terminating itself after 10 seconds once the request is processed. Also, if once created and no message can be retrieved from the SQS-Input queue for 10 seconds, it will automatically terminate itself to avoid wastage of instances.

(v) SQS-Output queue:

The SQS-Output queue stores the output to the user request for object detection. The output is a string which is of the format ***output_filename***. The Web-Tier instance is constantly listening to the SQS-Output queue. It will extract the string from the queue, parse it and return the output on the user's browser screen.

(vi) S3 bucket:

The S3 bucket is used to store the content of the output with the name of the object stored as the filename. The content is uploaded to the S3 bucket by the Persistent-App-Tier/Terminating-App-Tier instances.

2. Autoscaling

Scaling-out is managed by the Web-Tier instance based on the capacity of the instance and the requests sent by the user. Scaling-in is managed by the Terminating-App-Tier instances itself as they terminate themselves immediately after they are processing the user request.

We implement here a very simple logic where we create more instances based on the number of requests remaining to be processed in the SQS-Input queue. We have built here our custom AMI image that is being used to create the Terminating-App-Tier instances which will process the request and self-terminate after 5 seconds once completed.

The idea for scaling-out is to match the number of instances to be created with the number of messages present in the queue as follows:

number of Terminating_App_Tier instances to created = [number of messages in the input queue
- the number of running app instances]

where the number of app tier instances = Persistent_App_Tier instance + Terminating_App_Tier instances

Scaling-in will be handled by the Terminating-App-Tier instances which self-terminate themselves after processing the request and waiting for 5 seconds post that. We are making the instances wait seconds in order to check for any unserved messages in the input queue in order to avoid the time taken to create a whole new instance.

3. Code

I) Java Project: WebTier

- **Installation procedure:**

(Note: We have also written a shell script which automates the creation of the given infrastructure. The explanation for using the shell script is given in the final tab. The instructions mentioned below explain how to deploy the project manually)

i) Run **mvn clean package** in the terminal, while being in the directory of the project. Running the above command will create ***WebTier-1.0.0.jar*** file in the target directory of the project.

ii) We deploy this on an EC2 instance which will run the WebTier on an IP address. To copy the jar file on the ec2 instance, we use the command:

sudo scp -i pem_file path_of_the_jar_file ubuntu@ip_address_of_the_instance:~

iii) Then we use **sudo ssh -i pem_file ubuntu@ip_address_of_the_instance** to log into the ec2 instance.

iii) Run the command ***java -jar WebTier-1.0.0.jar &*** to start the SpringBoot application which will run the application on an embedded tomcat server listening to port 8080.

- **Functionalities of some the files involved:**

i) **WebTierInitializer.java:** This module initiates the WebTier instance and starts the SpringBoot application. This module is also responsible for the Scaling-out functionality of the Autoscaling feature. The application runs in a loop where it constantly looks for messages present in the SQS-Input queue. It scales out by cloning instances according to the algorithm mentioned above. The loop waits for one second before listening the SQS-Input queue again for efficient use of the AWS service.

ii) **WebTierController.java:** This module is a REST controller which maps the user HTTP request to generate a trigger message in the SQS-Input Queue. It also waits for the messages in the output queue and returns the filename and the corresponding output to the user's browser screen or request. This module handles the user requests.

II) Java Project: AppTier

- **Installation procedure:**

(Note: We have also written a shell script which automates the creation of the given infrastructure. The explanation for using the shell script is given in the final tab. The instructions mentioned below explain how to deploy the project manually)

i) Run **mvn clean package** in the terminal, while being in the directory of the project. Running the above command will create ***AppTier-1.0.0.jar*** file in the target directory of the project.

ii) We deploy this on an EC2 instance which will run the AppTier on an IP address. Run the command ***java -jar AppTier-1.0.0.jar &*** to start the SpringBoot application which will run the application on an embedded tomcat server listening to port 8080.

- **Functionalities of some the files involved:**

i) **AppInitializer.java:** This module initiates the always running AppTier instance and starts the SpringBoot application. It creates a Listener object and initiates listening to the SQS-Input queue.

ii) **Listener.java:** This module implements the attributes and functionalities of the Listener object. This object listens to the SQS-Input queue for trigger messages, downloads the video for every message it listens, performs object detection on it and uploads the results to the S3 bucket and send it to SQS-Output queue. Object Detection is performed by running the provided command ***./darknet detector demo cfg/coco.data cfg/yolov3-tiny.cfg + weight_path + video_path -dont_show > result.*** These commands are run using the ProcessBuilder object. The listener object would listen for trigger messages in the SQS-Input forever and will not terminate itself after performing object detection. We keep this AppTier running for achieving robust performance as terminating it would incur time in the creation of a new terminating AppTier instance, which would further cause delay in getting the output .

III) Java Project: AppTier_Terminator

- **Installation procedure:**

The AppTier_Terminator jar file is not deployed in the beginning. It is used for creating an AMI image which is used by the WebTier instance for scaling out. Before creating an AMI image, we deploy the AppTier_Terminator-1.0.0.jar (created in a similar manner to AppTier-1.0.0.jar) on an EC2 instance and install java, maven and Xvfb on that instance.

- **Functionalities of the files involved:**

It has the same attributes and functionalities as the AppTier above, with the additional functionality of terminating itself after object detection. After performing object detection, the AppTier_Terminator waits for 5 seconds to listen from the SQS-Input queue for any message request. If it does not retrieve any message in those 5 seconds, it will terminate itself. If it retrieves any message, it will perform object detection and continue the same process. Thus, scaling down feature of Auto-Scaling is achieved.

IV) Scripts: infrastructure.sh, WebTier.sh, AppTier.sh

Bringing up the initial infrastructure, deploying the projects and taking down the infrastructure is carried out using *infrastructure.sh.* This helps in separating the concerns of application code from its deployment infrastructure as well as separation of responsibility. *The requirement for successful use of these scripts is that the project folders of WebTier, AppTier and AppTier_Terminator, deployment files WebTier.sh and AppTier.sh, project files darknet_test.py and yolov3-tiny.weights should be in the same folder as infrastructure.sh file.*

To bring up the initial infrastructure, following bash commands are executed:

```
$ cd  to_location_of_infrastructure.sh_file
$ bash  infrastructure.sh  create
```

To deploy the project, following bash commands are executed:

```
$ cd  to_location_of_infrastructure.sh_file
$ bash  infrastructure.sh  deploy-project
```

To bring down the infrastructure, following bash commands are executed:

```
$ cd  to_location_of_infrastructure.sh_file
$ bash  infrastructure.sh  destroy
```

When the initial infrastructure is created using script, it results in *aws-resources.properties* file which is used by the spring boot projects as properties file, from which it reads important infrastructure configuration parameters using dependency injection. This file is copied to respective spring boot projects and the projects are deployed to initial EC2 instances using *deploy-project* argument. Finally, for ease of bringing down the infrastructure, *destroy* argument is used.

For configuring packages like Java and Maven on EC2 instances as well as deploying and running spring boot projects on them, WebTier.sh and AppTier.sh scripts are used.

4. Project status

Our project application meets the following requirements:

1. The primary requirement of request response cycle - request from user to WebTier, alerting from WebTier to AppTier through input SQS queue, object detection by deep learning model, return of results to output SQS queue and response from WebTier to the user - is robustly and successfully achieved.

2. Scaling in and scaling out feature is implemented using algorithm for autoscaling mentioned previously, improving the throughput of application.

3. **The best performance which we got for dealing with 20 Concurrent requests in total 40 requests was 2 minutes 36 seconds.**

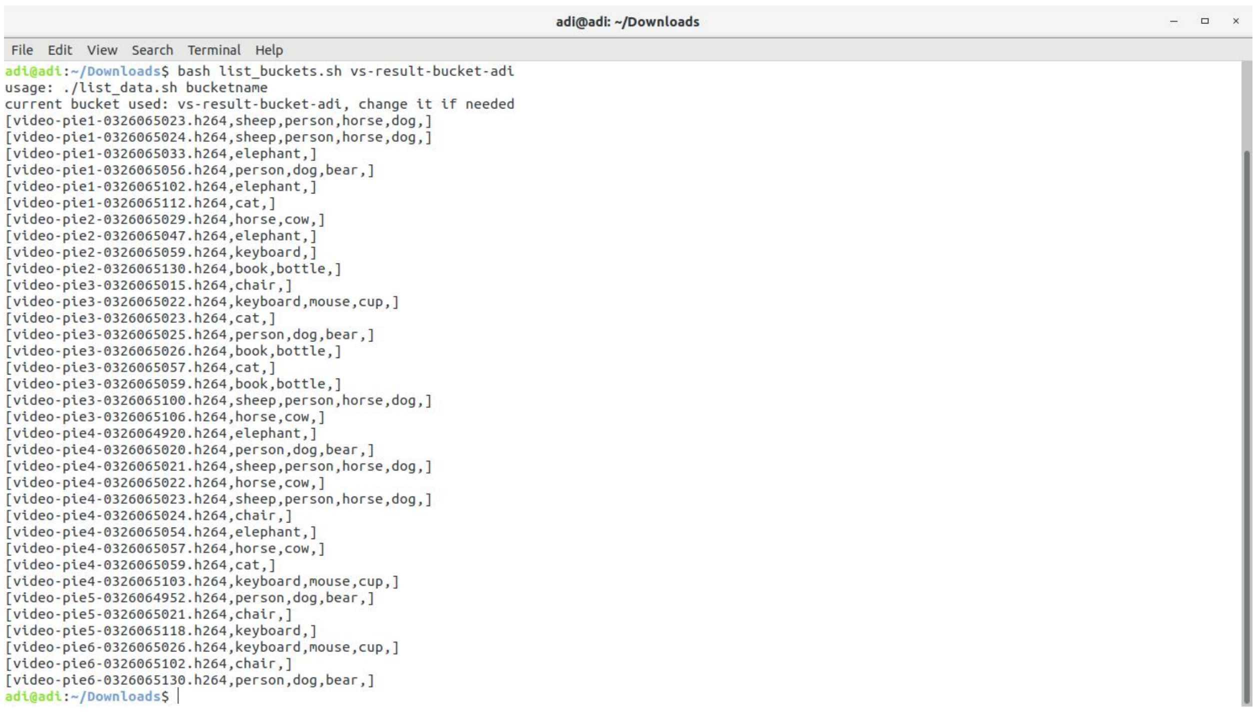
The best performance which we got for dealing with 100 Concurrent requests in total 100 requests was 11 minutes 20 seconds.

4. All the inputs-video names and outputs-detection results are stored in S3 for persistence in the form of (“video-pie3-0326091636.h264”, “chair”).

5. The application handles all the requests as fast as possible, does not miss any request and all object detections are correct.

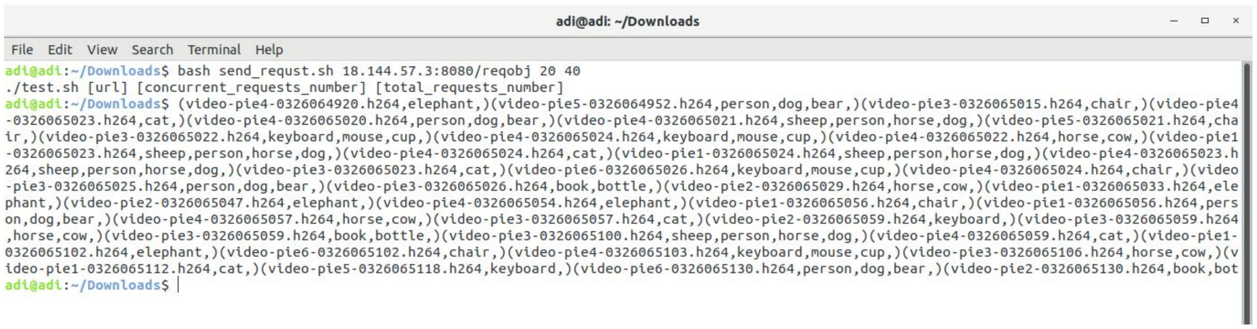
6. Performs Deep Learning - Detecting objects on videos and hence provides a video surveillance service to users in real time.

7. Security features are implemented using AWS services of VPC and IAM.



```
adi@adi:~/Downloads$ bash list_buckets.sh vs-result-bucket-adi
usage: ./list_data.sh bucketname
current bucket used: vs-result-bucket-adi, change it if needed
[video-pie1-0326065023.h264,sheep,person,horse,dog,]
[video-pie1-0326065024.h264,sheep,person,horse,dog,]
[video-pie1-0326065033.h264,elephant,]
[video-pie1-0326065056.h264,person,dog,bear,]
[video-pie1-0326065102.h264,elephant,]
[video-pie1-0326065112.h264,cat,]
[video-pie2-0326065029.h264,horse,cow,]
[video-pie2-0326065047.h264,elephant,]
[video-pie2-0326065059.h264,keyboard,]
[video-pie2-0326065130.h264,book,bottle,]
[video-pie3-0326065015.h264,chair,]
[video-pie3-0326065022.h264,keyboard,mouse,cup,]
[video-pie3-0326065023.h264,cat,]
[video-pie3-0326065025.h264,person,dog,bear,]
[video-pie3-0326065026.h264,book,bottle,]
[video-pie3-0326065057.h264,cat,]
[video-pie3-0326065059.h264,book,bottle,]
[video-pie3-0326065100.h264,sheep,person,horse,dog,]
[video-pie3-0326065106.h264,horse,cow,]
[video-pie4-0326064920.h264,elephant,]
[video-pie4-0326065020.h264,person,dog,bear,]
[video-pie4-0326065021.h264,sheep,person,horse,dog,]
[video-pie4-0326065022.h264,horse,cow,]
[video-pie4-0326065023.h264,sheep,person,horse,dog,]
[video-pie4-0326065024.h264,chair,]
[video-pie4-0326065054.h264,elephant,]
[video-pie4-0326065057.h264,horse,cow,]
[video-pie4-0326065059.h264,cat,]
[video-pie4-0326065103.h264,keyboard,mouse,cup,]
[video-pie5-0326064952.h264,person,dog,bear,]
[video-pie5-0326065021.h264,chair,]
[video-pie5-0326065118.h264,keyboard,]
[video-pie6-0326065026.h264,keyboard,mouse,cup,]
[video-pie6-0326065102.h264,chair,]
[video-pie6-0326065130.h264,person,dog,bear,]
adi@adi:~/Downloads$
```

Figure 2: S3 Bucket Content



```
adi@adi:~/Downloads$ bash send_requst.sh 18.144.57.3:8080/reqobj 20 40
./test.sh [url] [concurrent_requests_number] [total_requests_number]
adi@adi:~/Downloads$ (video-pie4-0326064920.h264,elephant,)(video-pie5-0326064952.h264,person,dog,bear,)(video-pie3-0326065015.h264,chair,)(video-pie4
-0326065023.h264,cat,)(video-pie4-0326065020.h264,person,dog,bear,)(video-pie4-0326065021.h264,sheep,person,horse,dog,)(video-pie5-0326065021.h264,cha
ir,)(video-pie3-0326065022.h264,keyboard,mouse,cup,)(video-pie4-0326065024.h264,keyboard,mouse,cup,)(video-pie4-0326065022.h264,horse,cow,)(video-pie1
-0326065023.h264,sheep,person,horse,dog,)(video-pie4-0326065024.h264,cat,)(video-pie1-0326065024.h264,sheep,person,horse,dog,)(video-pie4-0326065023.h
264,sheep,person,horse,dog,)(video-pie3-0326065023.h264,cat,)(video-pie6-0326065026.h264,keyboard,mouse,cup,)(video-pie4-0326065024.h264,chair,)(video
-pie3-0326065025.h264,person,dog,bear,)(video-pie3-0326065026.h264,book,bottle,)(video-pie2-0326065029.h264,horse,cow,)(video-pie1-0326065033.h264,ele
phant,)(video-pie2-0326065047.h264,elephant,)(video-pie4-0326065054.h264,elephant,)(video-pie1-0326065056.h264,chair,)(video-pie1-0326065056.h264,pers
on,dog,bear,)(video-pie4-0326065057.h264,horse,cow,)(video-pie3-0326065057.h264,cat,)(video-pie2-0326065059.h264,keyboard,)(video-pie3-0326065059.h264
,horse,cow,)(video-pie3-0326065059.h264,book,bottle,)(video-pie3-0326065100.h264,sheep,person,horse,dog,)(video-pie4-0326065059.h264,cat,)(video-pie1-
0326065102.h264,elephant,)(video-pie6-0326065102.h264,chair,)(video-pie4-0326065103.h264,keyboard,mouse,cup,)(video-pie3-0326065106.h264,horse,cow,)(v
ideo-pie1-0326065112.h264,cat,)(video-pie5-0326065118.h264,keyboard,)(video-pie6-0326065130.h264,person,dog,bear,)(video-pie2-0326065130.h264,book,bot
adi@adi:~/Downloads$
```

Figure 3: Test Script Output

5. Individual contributions (optional)

Kunal Vinay Kumar Suthar (ASU ID: 1215112535)

- **Design**

  I collaborated with my teammates in coming up with the infrastructure used in our project. I contributed to the discussions of having a persistent app-tier which does not terminate itself and listens forever, which is allowing us to serve all the requests and thus ensuring reliable performance. This was not the case, when we had only terminating app tiers, where we were missing some requests. I also contributed to the design decision of keeping the Auto-scaling module independent from the module which deals with the user requests. This was done by creating the WebTierInitializer(handles Autoscaling) and the WebTierController(handles the user requests) which run parallelly and independently. Due to this feature, our application will not break down even if autoscaling fails, as we have a persistent AppTier instance which is always running.

- **Implementation**

  I was responsible for developing the pipeline/infrastructure of the project which involved from ensuring that the user request is sent to receiving the output for the request. I worked on creating the front-end for the WebTier instance in HTML and implementing the SpringBoot application for the WebTier, persistent App-Tier and the terminating App-Tier instances, which involves tasks like creating the instances, ensuring that the messages are being sent in the queue and is received by the instances, and the answer is getting stored in the S3 bucket. I also worked on the implementation of WebTier, Persistent-AppTier and Terminating App-Tier instances in an object oriented manner(which included deciding the attributes and functions for the classes of tier-instances, EC2, SQS and S3). I also worked on handling the HTTP requests from the 1) user to the cloud application 2) the cloud application to the raspberry-pi cluster 3) the raspberry-pi cluster to the cloud application 4) the cloud application to the user.

- **Testing**

  I was responsible for testing different modules of the project which involved communication between different AWS components(Web-Tier instances, SQS-Input queue, SQS-Output queue, Persistent App-Tier instance, Terminating App-Tier instances, S3). I was also involved in the final application's performance testing, where my contribution was introducing delays within the code at crucial junctures to ensure robust performance and efficient use of resources.

Jay Shah (ASU ID: 1215102837)

Introduction
The aim of this project was to build an elastic video surveillance application that can automatically scale out and in on-demand and cost-effectively by using cloud resources. Because this application requires a variety of computing, storage, and message cloud services that are also easily scalable, we used the Amazon Web Services (AWS) to build this application. This application takes in an input URL that makes the request for a video and returns the objects detected in that video to the user using an AMI image that was provided to us. It makes use of a deep learning model and AWS cloud resources such as computing, queues, and storage. The major parts of this project can be divided as (a)Designing the infrastructure of the application (b)Implementing the REST application for using the AWS services (c)Writing scripts for easy use of commands for setting up the architecture & (d) Testing the application manually and using the scripts provided to us.

1. Designing the infrastructure of the application
We needed to design the infrastructure for this application so that it is robust, easy to scale in and out and uses the resources very efficiently.

- I came up with the basic idea of implementing the auto-scaling of instances here (formula is shown in Section-2: Auto-scaling) to handle multiple demands and helped in integrating it within the code.
- I collaborated with teammates & after giving enough thoughts to the infrastructure design to it and weighing the parameters, we were able to reach a consensus on the final infrastructure as shown in the Figure-1 above.

2. Implementing the overall infrastructure for using the AWS services

- I was responsible for integrating the Deep Learning (darknet) model within the framework we were building parallelly. I tried those commands by manually creating instances and testing the python script given to us and later on helped to integrate those commands into the App_Tier instances. I was able to debug all deep learning model related application failures while testing.
- I was responsible for integration of the Auto-Scaling module in the Web_Tier application and making sure it is working smoothly in the case of handling multiple users-requests and concurrent requests.

3. Testing the application - manually and using the scripts
We also did manual testing of the application that was finally built to check its functionalities and it proved to be critical as it helped us fine-tune certain parts of the application in order to make it handle concurrent requests and checking that it does terminate abruptly. We also did this part using the scripts provided to us in order to better evaluate our application.

- It was an iterative process where all of us tested the application individually, found some minor changes causing messages to be lost, not handled alongside maintaining time; we made those changes parallely by discussing with one another.