COMPX527 Assignment 1 Report

Glenn Cumming
Department of Computer Science
University of Waikato
Hamilton, New Zealand
glenn@hif.nz

Mitchell Grout
Department of Computer Science
University of Waikato
Hamilton, New Zealand
mjg44@students.waikato.ac.nz

Shufen Li
Department of Computer Science
University of Waikato
Hamilton, New Zealand
sl302@students.waikato.ac.nz

Sunny Chikara
Department of Computer Science
University of Waikato
Hamilton, New Zealand
sc420@students.waikato.ac.nz

YingJun Huang
Department of Computer Science
University of Waikato
Hamilton, New Zealand
yh320@students.waikato.ac.nz

***Abstract*—We designed an Object Detection web service on the AWS Cloud Provider, using Terraform for programmable infrastrucure, Ansible for deployment, and Python for the web service and object detection.**

***Index Terms*—AWS, COMPX527, Terraform, Ansible, Flask, Python, Object Detection, Retinanet**

I. Solution Summary

The Flask Object Detection service is a load balanced HTTP service deployed in AWS EC2, using the AWS ELB load balancer. The service can accept a JPG image and run object detection on it, and returns the resulting image with identified objects highlighted.

II. Motivation

We decided on object detection in the cloud in order to learn and appricate the challenges of providing a stateless, scalbale service that could be used for a variey of other projects. Examples included

- Contexutal threat analysis, such as humans detected in an area which should only include livestock.
- Numbercial anyalisis, such as the number of a species of wildlife in an area over time.
- Absence detection, indetifying objects that should be in an image but are not

III. Proposed Solution

An Amazon Web Services based cluster of web servers taking images from clients and returning images with bionding boxed indetified objects plus json formatted data, behind an Eleastic Load Balancer. The cluster will be horizontally scalable by adding web servers, and accessable via curl or any other HTTP client that can perform HTTP POST requests. It was trained used the AWS COCO data set [1], and also used the set to for testing and demonstrations.

IV. Solution Architecture

We decided to use a load balanced web service architecture, a very common and well understood model with common solutions to scalability and uptime. We have the following components and features:

A. The Image Processing Server

The project uses the Retinanet based Object Detection developed by Keras [2]. The web service was developed in Python using a Flask server [3].

The server recives via HTTP JPG formatted image, and returns the image with bounding boxes around identified objects, plus JSON data of the images.

B. The Load Balancer

The object detection being inhernetly slow forces the use of laod balancing and scaling techniques. Using the AWS Elastic Load Balancer allowed us balance the HTTP requests over two or more EC2 servers running the Flask servers. We could create as many Flask servers as we wished, although of course this lent itself to waisted compute resources. Though we had great sucess early on using AWS AutoScaling Groups, which allows a lower and upper limit of EC2 instances running our Flask server to be defined, this functionality was not avalible in the student accounts.

C. The Web User Interface

In order to have a better demo, it was decided to go ahead and produce a web site interface that would allow uploading

D. The Web API

The intended standard way to interact with the Flask Object Detection Cluster is to access it via HTTP calls using the url http://<load_balancer_fqdn>/detect. As the system is inteneded for the use of non-private images no ssl was implemented on the load balancer, though it is supported.

[1] https://registry.opendata.aws/fast-ai-coco/

[2] https://github.com/fizyr/keras-retinanet

[3] https://pypi.org/project/Flask/

E. Amazon Web Services Cloud

The service running on the Amazon Web Services Cloud, the required choice of the assignment. Though later in lectures there it wa smentioned we could look in to using other Cloud Service Providers, we decied we had already made enough progress to commit ourselves to AWS for this project. Though we did use terraform, which supports IaaS deployments with mulitple CSP, in case we had a need to effect a change quickly.

V. Technology

The following are the significant technologies we used in this assignment.

A. Retinanet

Retinanet is an open source object detection neural net. This technology was chosen for the following reasons [4].

- Free Open Source project obviously beneficial for a project with a limited budget.
- Python 3 for rapid development and deploymewnt. With a mix of experinced and less experinced team members this was particulary useful.
- Specific instructions using it with the dataset avaliable on the Registry of Open Data on AWS [5]

B. Terraform

Terraform was given in as an example of the provisioning service to use in this assignment. AWS CloudFormation was another option suggested. The decision to go with terraform rather than CloudFormation was based on the following

- CloudFormation is prioritory and is specific to AWS cloud offierings. Though this assigmnet is on a small and temporty scale, we still not want to use a technology that would create vendor lockin
- Terraform supports provisioning many different platforms, both open standards and priopritary, such as Azure, Goolge Cloud, Kubernetes and OpenStack [6]. Developng expericne in deployments with terraform therefore was demied to be more useful.

C. Ansible

Though Terraform is capable of running commands post-install in order to install services and other software needed for our cluster, our cluster used Ansible playbooks instead. Though it meant learning another technology, we demeed a good use of our time since:

- Terraform can only run scripts, which would have to be created.
- Ansible's langauge is very flxible and is created specifally for the purpose of delpoymnet.
- This is a recommended approach by HashiCorp, the devlopers of Terraform. [7]

D. Github

We used github for source control of all documents and code. This was chosen over other solutions suc h as SVN and Mecurial due to the familiarity of some of the team members with git. Others in our team have leanrt cloning, pulling and branching. [8]

E. Slack

As we needed to communicate effectively over a period of weeks without seeing each other often, we decided to use the colloberation software Slack [9]. We used a free account, and created a channel to allow us to privately communicate about our work. It was used extensively and allowed constant effecteive communication.

F. Trello

In ordr to break up and manage tasks we created a Kanban Board in Trello [10]. This got us started with basic learning and setup tasks early on, but as the project progressed we tended to move to the use of Slack to define and assign tasks and workloads.

G. LaTex

LaTex [11] is the accepted standard for scientific papers; therefore although the assignment pointed us to the IEEE A4 standard for reporting, we opted to use the LaTex standard. This gave us very useful experince in using LaTex for proper report formatting. This report was created in LaTex using the IEEEtran document class provided by IEEE template avaliable at [12] , and generated into PDF by Gnome LaTeX [13]

H. Flask

The decision to develop and depoly a service using the keras-retinanet Python library immediately suggested to us we should again use Python to provide the web service. Our team had develpoers with experince in both Tornado and Flask; Flask was used as the web service development fell on the person with Flask experince. Both solutions would have been suitable.

VI. Existing object detection soltutions

There is a wide range of cloud and non cloud solutions. A comparison of these to our own solution is beyound the scope of this document. Those provided by the large Cloud Service Providers includes:

A. Rekognition

Amazon's object detection solution [14].

[4]

[5] https://registry.opendata.aws/fast-ai-coco/

[6] https://aws.amazon.com/cloudformation/

[7] https://www.hashicorp.com/resources/ansible-terraform-better-together

[8] https://github.com/

[9] https://slack.com/

[10] https://trello.com/

[11] https://www.latex-project.org/

[12] https://www.ieee.org/conferences/publishing/templates.html

[13] https://wiki.gnome.org/Apps/GNOME-LaTeX

[14] https://aws.amazon.com/rekognition/

B. Vision AI

Google's object detection solution [15].

C. Computer Vision

Microsofts object detection solution [16].

VII. Security and Vunlerability Assessment

A. Data Security

The Flask Object Detection Cluster was specifically created to be stateless and public. Interception of HTTP requests and results can be easily interceptied by third parties with access to the networks between the client and the service. Users of the service should be aware that any images sent or data recived from the service could be illictly obtained, mallicously modified, or corrupted in transit.

Migtigation could be acheived by adding SSL encryption and offloading it at the load balancer.

B. Access Security

a) Identity and Access Management: Management access of the resources residing on AWS is protected via the AWS Identity and Access Management (IAM) system [17]. We used individual accounts provided by the course. If an account was compromised the it would allow malicous actors to shut down services, or replace the current services with malacoius alternatives. For example, they cloud replace an instance with one that will return malware embeded in an image.

Mitigation can be using good protection practices for authentication information, Two Factor Auth (2FA), regular monitoring of the service with a set of well known HTTP requests checked against expected results.

b) Server Access: The Flask servers are publically accessable via SSH, needed for deployment. If the key pair is compromised or the keys obtained then root access to the Flask servers cloud be obtained. This would grant the attacker full acccess and the ability to use the server's resources for there own purposes.

Migragtion can be using ssh key passwords, encrypted storage of the keys, checking server hashes, and keeping ssh software up to date with security patches.

C. Network Security

We used AWS Security Groups to restrict the Load Balancer to only ingress traffic port 80. The Image Processers restricted to port 22 and 80.

Allowing direct access to the service port is a known problem that could allow an attaacker to easily DDOS a single server, bypassing the Load Balancer. To mitigate this we need to work out how to define the CIDR range in the security group to match the internal IP of the Load Balancer. The service is load banced, but we where not able to use AutoScaling Groups due to our account restrictions. This results in having to manually create additional servers to cope with increased requests, and so a DDOS would be more effective since a human will have to respond to any such attack.

To mitigate this we should move to an account that allows the use of Auto Scaling Groups.

D. Monitoring and Security

AWS EC2 instance monitoring was enabled during the terraform creation of the cluster. ELB monitors the avalibility of the Flask HTTP servers. The lack of external monitoring means that we are totally dependent on AWS for security checks and uptime monitoring.

Mitigation of the lack of monitoring is obviousloy acheived by using both in house and monitoring services. Common examples of in house monioring is via Nagios, Icinga2, Munin and Promethues. Monitoring services include Uptime.com [18] for ping and HTTP monitoring, and Paessler [19], which provides addiotnal services such as port monitoring.

VIII. Actual AWS Expenditure

TODO

IX. Future Improvements

a) Packaging: Deployment scripts were chosen for deployment and running the services. However, if the service was to be developed further we would being to use stdeb [20] for packaging for the Ubuntu servers, and potentially set up a Personal Package Archive [21]. This would ensure futher development and upgrades by making it part of the standard apt package management system.

b) Service Management: Currently the starting and stopping of the service is done via control scripts. These would be integrated in to systemd service management [22].

c) HTTP: The service is stateless, plublic and insecure. If a layer of security is desired to protect the data in transit, then HTTPS can be added to the Elastic Load Balancer via the AWS Certificate Manager [23]. The simple reason for not using this was cost: we would have to pay for extra services when our budget was tight.

d) Cloudflare Multi Region Load Balancing: Redeployment of the service changes the FQDN in the URL. Also, the service is currenlty deployed on only one region. The Cloudflare Load Balancing would allow the use of many regions or in fact many Cloud Providers to be used with a single contant url presented to the send consumer. The fact that the service is stateless makes it very suitable to this form of scale out [24]

[15] https://cloud.google.com/vision/

[16] https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/

[17] https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

[18] https://uptime.com/uptime-monitoring

[19] https://www.paessler.com/port_monitoring

[20] https://pypi.org/project/stdeb/

[21] https://help.ubuntu.com/community/PPA

[22] https://wiki.debian.org/systemd

[23] https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/ssl-server-cert.html

[24] https://www.cloudflare.com/load-balancing/

e) Monitoring: Future development of the cluster would likely use additional monitoring of performance, costs and uptimes using dedicated EC2 instances, and such existing monitoring systems as Icinga2, Promethues and Munin. Uptime moniroing would of nessescity be run etxernally, such as on internal machines or another cloud providers offering.

We would also start to use third party external monitoring services such as Uptime.com [25] or Paessler [26]

X. Team Members Contributions

XI. Assignment Requirements Completion

The requirements for this Assignment were communicated in serveral ways, including the assignment documentation, messages in the Moodle channel, the lectures, and by asking the lectuerer and asistant directly. We treated these as customer reuirements, which typically start with a good understanding of the desired outcomes and often require feedback from the customer to ensure we are achiving them.

The following is what we have determine dto be the requirements, and how we have achived each one.

- Use of AWS to create a non-trivial service, demonstrating our understanding of cloud technologies, their use, and their challenges.
- Use of technologies such as Terraform, Ansible and Cloud Formation for programmably defined infrastructure and depoyments
- Use of a open dataset from AWS
- Work within a team with very diverse technical backgrounds and manage them to ensure fair distribution of the work.

.

[25] https://uptime.com/uptime-monitoring

[26] https://www.paessler.com/