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

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

***Abstract*—Abstract Text**
***Index Terms*—AWS, COMPX527**

I. Solution Summary

Solution Summary

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

In order to

IV. Solution Architecture

Solution Architecture

A. The Image Processing Server

The project uses the Retinanet based Object Detection developed by Keras [1]. The web service was developed in Pythion using a Flask server [2].

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 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. The code submitted for this project includes *simple_upload.py* and *forked_upload.py*, which are Python 3 examples for using the HTTP API.

V. Development

A. Technology

a) Retinanet: Retinanet is an open source object detection neural net. This technology was chosen for the following reasons.

- Open Source
- Python 3 for rapid development
- Specific instructions using it with the dataset avaliable on the Registry of Open Data on AWS [3]

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 TODOREF. Though this assigmnet is on a small and temporty scale, we still not want to use a technology that would create vendor lockin

[1] https://github.com/fizyr/keras-retinanet
[2] https://pypi.org/project/Flask/
[3] https://registry.opendata.aws/fast-ai-coco/

- Terraform supports provisioning many different platforms, both open source stand priopritary, such as Azure, Goolge Cloud, Kubernetes and OpenStack [4]. 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. [5]

d) 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.

B. Compairsion with existing object detection soltutions

Amazon Rekognition

VI. Security Assessment

Security 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.

B. Access Security

Access Security

C. Network Security

Network Security

D. Vulnerability Assessment

E. Monitoring

AWS EC2 instance monitoring was enabled during the terraform creation of the cluster. 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 Icinga, Promethues and Munin. Uptime moniroing would of nessescity be run etxernally, such as on internal machines or another cloud providers offering.

VII. Actual AWS Expenditure

TODO

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

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

VIII. 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 [6] for packaging for the Ubuntu servers, and potentially set up a Personal Package Archive [7]. 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 [8].

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 [9]. The simple reason for not using this was cost: we would have to pay for extra services when our budget was tight.

IX. Team Members Contributions

X. Assignment Requirements Completion

XI. Standards

This report was created in LaTex using the IEEEtran document class provided by IEEE template avaliable at [10] , and generated into PDF by Gnome LaTeX [11] .

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

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

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

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

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

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