

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, Terraform, Ansible, Flask, Python, Object Detection, Retinanet

I. SOLUTION SUMMARY

Solution Summary

II. MOTIVATION

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

- Contextual threat analysis, such as humans detected in an area which should only include livestock.
- Numerical analysis, such as the number of a species of wildlife in an area over time.
- Absence detection, identifying 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 bounding boxes identified objects plus json formatted data, behind an Elastic Load Balancer. The cluster will be horizontally scalable by adding web servers, and accessible via curl or any other HTTP client that can perform HTTP POST requests.

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 ¹. The web service was developed in Python using a Flask server ².

¹<https://github.com/fizyr/keras-retinanet>

²<https://pypi.org/project/Flask/>

B. The Load Balancer

The object detection being inherently slow forces the use of load balancing and scaling techniques. Using the AWS Elastic Load Balancer allowed us to 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 wasted compute resources. Though we had great success 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 available 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 intended for the use of non-private images no ssl was implemented on the load balancer, though it is supported.

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 was mentioned we could look in to using other Cloud Service Providers, we decided we had already made enough progress to commit ourselves to AWS for this project. Though we did use terraform, which supports IaaS deployments with multiple CSP, in case we had a need to effect a change quickly.

V. DEVELOPMENT

A. 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 ³.

- Free Open Source project obviously beneficial for a project with a limited budget.
- Python 3 for rapid development and deployment. With a mix of experienced and less experienced team members this was particularly useful.
- Specific instructions using it with the dataset available on the Registry of Open Data on AWS ⁴

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 proprietary and is specific to AWS cloud offerings. Though this assignment is on a small and temporary scale, we still not want to use a technology that would create vendor lockin
- Terraform supports provisioning many different platforms, both open standards and proprietary, such as Azure, Google Cloud, Kubernetes and OpenStack ⁵. Developing experience in deployments with terraform therefore was deemed 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 deemed a good use of our time since:

- Terraform can only run scripts, which would have to be created.
- Ansible's language is very flexible and is created specifically for the purpose of deployment.
- This is a recommended approach by HashiCorp, the developers of Terraform. ⁶

d) *Github*: We used github for source control of all documents and code. This was chosen over other solutions such as SVN and Mercurial due to the familiarity of some of the team members with git. Others in our team have learnt cloning, pulling and branching. ⁷

e) *Slack*: As we needed to communicate effectively over a period of weeks without seeing each other often, we decided to use the collaboration software Slack ⁸. 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 effective communication.

f) *LaTeX*: LaTeX ⁹ 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 experience in using LaTeX for proper report formatting. This report was created in LaTeX using the IEEEtran document class provided by IEEE template available at ¹⁰, and generated into PDF by Gnome LaTeX ¹¹

g) *Flask*: The decision to develop and deploy 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 developers with experience in both Tornado and Flask; Flask was used as the web service development fell on the person with Flask experience. Both solutions would have been suitable.

B. Existing object detection solutions

There is a wide range of cloud and non cloud solutions. Those provided by the large Cloud Providers includes:

- a) *Rekognition*: Amazon's object detection solution ¹².
- b) *Vision AI*: Google's object detection solution ¹³.
- c) *Computer Vision*: Microsoft's object detection solution ¹⁴.

VI. SECURITY AND VULNERABILITY 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 intercepted 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 received from the service could be illicitly obtained, maliciously modified, or corrupted in transit.

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 ¹⁵. We used individual accounts provided by the course. If an account was compromised then it would allow malicious actors to shut down services, or replace the current services with malicious alternatives. For example, they could replace an instance with one that will return malware embedded in an image.

b) *Mitigation*: 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.

c) *Server Access*: The Flask servers are publicly accessible via SSH, needed for deployment. If the key pair is compromised or the keys obtained then root access to the Flask servers could be obtained. This would grant the attacker full access and the ability to use the server's resources for their own purposes.

³

⁴<https://registry.opendata.aws/fast-ai-coco/>

⁵<https://aws.amazon.com/cloudformation/>

⁶<https://www.hashicorp.com/resources/ansible-terraform-better-together>

⁷<https://github.com/>

⁸<https://slack.com/>

⁹<https://www.latex-project.org/>

¹⁰<https://www.ieee.org/conferences/publishing/templates.html>

¹¹<https://wiki.gnome.org/Apps/GNOME-LaTeX>

¹²<https://aws.amazon.com/rekognition/>

¹³<https://cloud.google.com/vision/>

¹⁴<https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/>

¹⁵<https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>

C. Network Security

Network Security

D. Monitoring

AWS EC2 instance monitoring was enabled during the terraform creation of the cluster. ELB monitors the availability of the Flask HTTP servers.

VII. ACTUAL AWS EXPENDITURE

TODO

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 be going to use `stdeb`¹⁶ for packaging for the Ubuntu servers, and potentially set up a Personal Package Archive¹⁷. This would ensure further 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 management¹⁸.

c) *HTTP*: The service is stateless, public 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 Manager¹⁹. 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 currently 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 content url presented to the end consumer. The fact that the service is stateless makes it very suitable to this form of scale out²⁰

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 Icinga, Prometheus and Munin. Uptime monitoring would of necessity be run externally, such as on internal machines or another cloud providers offering.

IX. TEAM MEMBERS CONTRIBUTIONS

X. ASSIGNMENT REQUIREMENTS COMPLETION

XI. STANDARDS

¹⁶<https://pypi.org/project/stdeb/>

¹⁷<https://help.ubuntu.com/community/PPA>

¹⁸<https://wiki.debian.org/systemd>

¹⁹<https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/ssl-server-cert.html>

²⁰<https://www.cloudflare.com/load-balancing/>