COMSM0010 Cloud Computing Project

George Church - 17137
Department of Computer Science
University of Bristol
Bristol, England
gc14768@my.bristol.ac.uk

***Abstract*—In this report I describe my cloud computing based solution for a simple photo sharing website. This application was developed as as part of the COMSM0010 Cloud Computing unit at the University of Bristol. The application can be run online at http://www.geochu.com/.**

***Index Terms*—cloud, computing**

I. Application

The application that I developed is a simple image sharing website. The home page of the website displays all of the images that have been shared by users. Each image has a title along with the name of the person that shared the image. A screenshot of the homepage of the website is shown in Fig. 1.

A user can share their own image by navigating to the upload page. They can then choose an image from their file system that they want to upload, provide a title and their name and then hit the submit button. The image will then be displayed on the homepage along with all of the other uploaded images. A screenshot of the upload page of the website can be seen in Fig. 2.

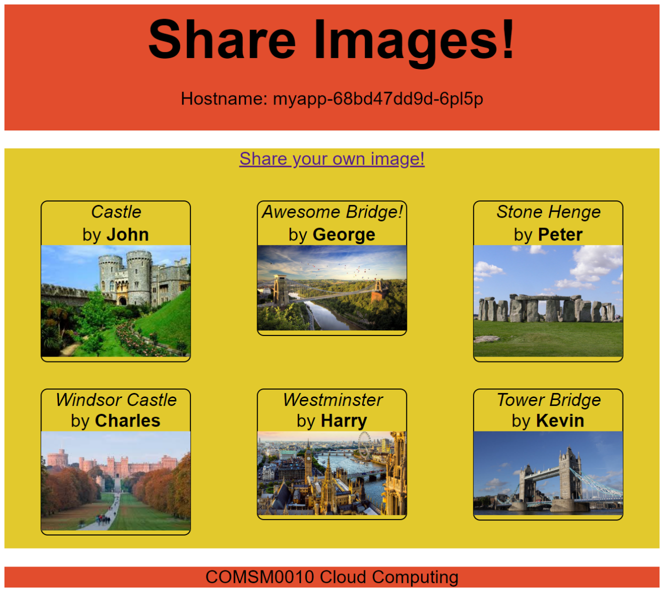


Fig. 1: A screenshot of the homepage of the website. This page shows images that have been uploaded by users. Above each image there is a title and the name of the user that uploaded the image. This screenshot was taken on Google Chrome, it may look different in other web browsers.

II. System Architecture

The application architecture has three main components: the web server, image storage and the database. Each of these components are described in the following subsections.

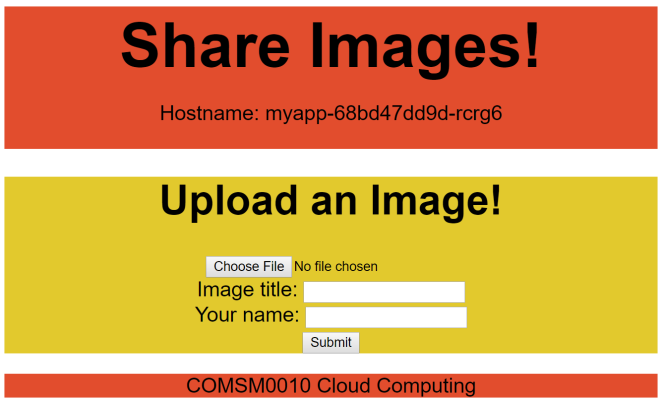
A. Web Server

The web server is implemented using the Express web application framework [1] for Node.js. The mustache.js template engine [2] is used in order to render the HTML pages that are being served. The Node.js package Multer [3] is used to handle the multipart/form-data which is sent to the server when a user uploads an image. The web server also makes use of the AWS Node.js SDK [4] in order to interact with the AWS services that are being used. The full source code of the application can be found at https://github.com/gchurch/Cloud_Computing.

The web server is running in the Oracle Cloud. The web server is containerised using Docker and the image is stored in the Oracle Cloud Infrastructure Registry [7]. The Oracle Cloud Container Engine for Kubernetes is used to manage a Kubernetes cluster. Kubernetes is used to orchestrate many containers all running the web server. Kubernetes also manages automatic deployment and scaling of the application. Running our application on a Kubernetes cluster also means that our application is highly available and fault tolerant.

The Kubernetes cluster has 3 nodes, one in each of the availability domains of Oracle Cloud's London region. Each node is a Standard2.1 virtual machine running on a 2.0 GHz Intel Xeon Platinum 8167M processor with 8GB of of Memory [8]. I created a Kubernetes deployment that runs 4 pods with each pod running the web server container. The container image is also located at the London region which means that the image can be quickly and easily pulled when a pod is deployed. I also created a Kubernetes service which launches a 100Mbps load balancer with an external IP address. This IP address exposes the deployment to the outside world. The load balancer distributes incoming requests across each of the pods in the cluster.

In the header of each page of the website, the hostname of the pod that serves you the page is displayed. If you refresh the application a few times you should see that the hostname changes, demonstrating that a different pod has served you the page.


Share Images!

Hostname: myapp-68bd47dd9d-rcrg6

Upload an Image!

COMSM0010 Cloud Computing

Fig. 2: A screenshot of the upload page of the website. There is a file input to browse for the image that you want to upload. There are text fields for user to provide the image title and their name. There is a submit button to send the information that that the user provides to the web server. This screenshot was taken in Google Chrome, it may look different in other web browsers.

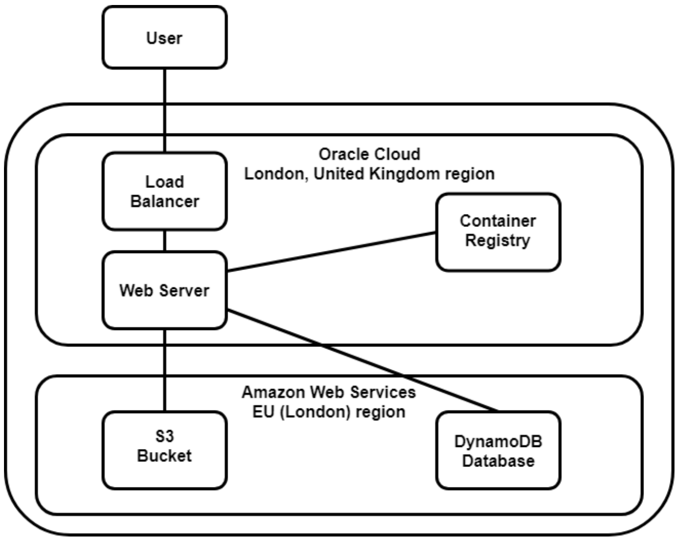


Fig. 3: A diagram of the overall architecture of the application. The diagram shows what components are located on the Oracle Cloud and what components are located on Amazon Web Services. Lines between components indicate that they interact with each other. The Web Server component represents the Kubernetes cluster running multiple pods containing the containerised web server.

B. *Image Storage*

The images that are uploaded to the website are stored using the Amazon Simple Storage Service (Amazon S3) [11]. The images are stored in an S3 bucket located in the EU (London) AWS region. S3 stores all of the images redundantly across the three availability zones in an AWS region in order to have high durability. The S3 bucket scales automatically allowing the application to store as many images as it needs.

The web server makes us of the AWS Node.js SDK in order to easily interact with the S3 bucket [4]. When a user presses the submit button to upload an image, a post request is sent to the web server with the image in the body of the request. When the web server receives this request, it takes the image and calls the PutObject function in the SDK in order to store the image in the S3 bucket. Similarly, when a get request is made to the web server for a particular image, the web server fetches that image from the S3 bucket by calling the GetObject function in the SDK and then sends the image in the response.

C. *Database*

The application uses Amazon DynamoDB [13] to store the information about the uploaded images. Amazon DynamoDB is a document store database built to be performant at scale. DynamoDB allows you to build application with virtually unlimited throughput and storage.

The primary partition key is the name of the user that shares the image and the primary sort key is the title of the image. Each item additionally contains the file name of the image that is uploaded and stored in the S3 bucket. The database is located in the EU (London) AWS region in order to have low latency for UK users.

The web server uses the scan operation in order to retrieve all of the items in the databases. It does this in order to display all of the images on the homepage. The scan operation returns a JSON file containing all of the data on the images. This JSON file is used to construct the necessary HTML code in order to display the images.

D. *Overall Architecture*

The application makes use of both the Oracle Cloud and Amazon Web Services (AWS), the reason for this decision is discussed in Section IV. All of the cloud services used are located in the London region so that the application will have low latency when accessed from within the UK. An overview of the components in the system architecture is shown in Fig. 3.

III. Scalability

The application was built to be scalable. S3 automatically scales the storage in order to meet our applications needs. DynamoDB allows the application to have virtually unlimited throughput and storage. Using Kubernetes also makes it possible to scale the application.

Kubernetes uses the Horizontal Pod Autoscaler in order to automatically scale the number of pods in a deployment based on the observed CPU utilisation. In order to obtain the CPU utilisations, the metrics-server needs to be deployed in the cluster [14]. Unfortunately, I ran into problems when trying to launch the metrics-server on the Kubernetes cluster, which prevented me from successfully using the Horizontal Pod Autoscaler. With the metrics-server working properly, making the application scale automatically is very simple.

IV. Cloud Services

I make use of both the Oracle Cloud and Amazon Web Services (AWS) to run my application. The decision for which services to use was made due to both economic and practicality reasons.

I used Oracle Cloud to run the Kubernetes cluster as I was provided with an Oracle Cloud account containing £3500 worth of credits. At the time of writing, running a 100Mbps Load Balancer costs $0.0213 per hour and each of the three VM.Standard2.1 instances running as the nodes costs $0.0638 per hour [9]. This adds up to $5.10 per day, so clearly I will be able to run the cluster for a long time without running out of credits. I could have easily run the cluster using AWS's Elastic Container Service for Kubernetes (EKS) [10] instead, however this would have meant taking money out of my own pocket. At the time of writing, Amazon's EKS service costs $0.20/hour plus the cost of the EC2 worker nodes. This service is not covered in the free tier of AWS.

I used AWS for the image storage and database of the web application, mainly because AWS provides an SDK for Node.js whereas Oracle Cloud does not. The SDK makes it a lot easier to interact with the cloud services programmatically. With Oracle Cloud I would have had to use the REST API to interact with the services, which would have been more difficult. The AWS free tier covers the usage of S3 and DynamoDB. The free tier allows 5GB of storage, 20,000 get requests and 2,000 put requests for S3 each month. It also allows 25GB of storage for DynamoDB each month. This current application is well within these bounds. If I go over any of these allowances then I have $150 worth of credits in my account that can cover any of the costs of these services.

V. Security

Since the application was not built with security in mind, I was wary about exposing the credentials that the web server uses to interact with the AWS services. The S3 bucket and the DynamoDB database are both set to not allow public access, so the web server has to use the security credentials of an IAM user in order to gain access to AWS services [12]. I took the precaution of making sure that the permissions that this IAM user has are limited in order to mitigate the damage that can be done if the security credentials are somehow exposed. So if a malicious actor manages to get their hands on these credentials via a security vulnerability in the website, they won't be able to freely use all of the available AWS services and leave me with a large bill.

References

[1] https://expressjs.com/
[2] https://www.npmjs.com/package/mustache
[3] https://www.npmjs.com/package/multer
[4] https://aws.amazon.com/sdk-for-node-js/
[5] https://www.docker.com/
[6] https://cloud.oracle.com/containers/registry
[7] https://cloud.oracle.com/containers/kubernetes-engine
[8] https://cloud.oracle.com/compute/virtual-machine/features
[9] https://cloud.oracle.com/iaas/pricing
[10] https://aws.amazon.com/eks/
[11] https://aws.amazon.com/s3/
[12] https://aws.amazon.com/iam/
[13] https://aws.amazon.com/dynamodb/
[14] https://github.com/kubernetes-incubator/metrics-server