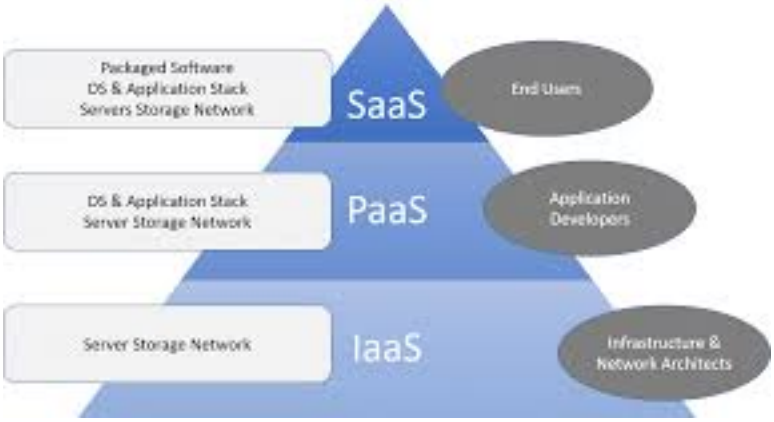
PES UNIVERSITY



CS352: Cloud Computing
Jan-May 2019

SelfieLessActs on AWS

PHOTO SHARING APPLICATION with the help of CONTAINER ORCHESTRATION

CC_308_313_331_352 | 04-**May**-2019

Member Name	USN	Section
Rohan R	01FB16ECS**308**	F
Rohit U Bogulla	01FB16ECS**313**	F
Sailesh Gaddalay	01FB16ECS**331**	F
Sharath N	01FB16ECS**352**	F

Introduction

- We have developed an Online Photo Sharing application that is based on Amazon's AWS.
- Application focuses on people sharing their good deeds which they encounter in day-to-day life.
- This App is auto orchestrated and auto containerized
- Application is partitioned into ACTS and USERS where the later handles all the user related processes and the former handles the acts uploaded by the users
- The functionality is achieved together with Python's FLASK Framework and AWS

Related work

Dockers were used to build, run, manage, and orchestrate the containerization of the containers in two components of the app's instances, users and acts. Python's FLASK Framework was used to build the backend of the application, and to handle with all the database related updates.

Elastic Load Balancing distributes incoming application or network traffic across multiple targets, such as Amazon EC2 instances, containers, and IP addresses, in multiple Availability Zones. Elastic Load Balancing scales your load balancer as traffic to your application changes over time, and can scale to the vast majority of workloads automatically

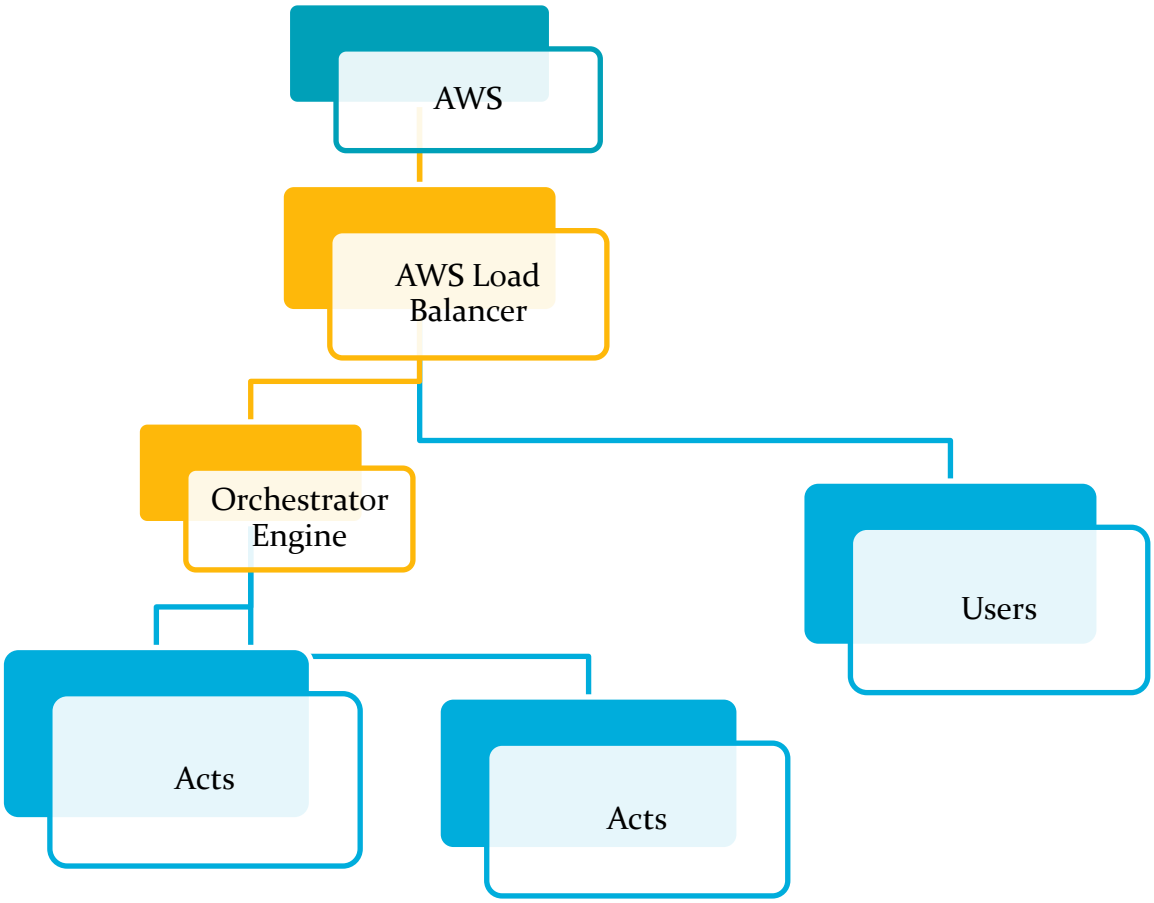
Since the SelfieLess Acts is a service-based applications, it must therefore employ fault handling techniques such as fault tolerance to cope with errors propagated by their constituent services, and thereby ensure an end-to-end Quality of Service (QoS).Fault tolerant computing accepts that faults are unavoidable and may lead to component failure

EXPERIMENTS/DESIGN

The heart of the application is the AWS which provides the instances for the partitions(as mentioned above). It has a AWS load balancer which forwards requests by users based on the paths to users and acts instance.

There is a Orchestrator Engine in acts instance which redirects requests to containers running in the instance in a Round-Robin order to balance the load. It also keeps checking the health of the containers and replaces the unhealthy containers with new ones. It does the part of auto scaling where it increases or decreases the number of containers running based on rules defined automatically. The users and acts instances are in sync with each other about the databases.

The container creation and management is done using the docker container manager. Each of the containers run their own versions of Flask web application updating the single database keeping the data in synchronization. One of the most important tasks was to implement database synchronization. The acts and users instances were communicating with each other to synchronize their data over REST API, but the containers in acts instance got connected to one single database on instance and kept updating their changes and migrating their modifications.



TESTING/RESULTS

All the testing of the REST API implementations was performed using Postman application. All the container orchestration was performed using a python script checking for the balancing, fault tolerance, and auto scaling. Although the containers were checked for manually, we tried to automate the process as much as possible.

16CS352: Cloud Computing – Final Project

Evaluation for `CC_308_313_331_352`
SelfieLess IP address: `http://imonfire-876680989.us-east-1.elb.amazonaws.com`
Acts-only IP address: `http://35.171.29.151`
Acts-only EC2 Username: `ubuntu`

1. One Acts container running with ID 376af17b848b
2. Acts container 376af17b848b is listening on port 8000
3. Acts container 376af17b848b is healthy
4. 20 successful API requests made to orchestrator
5. Script slept for 2 minutes
6. Second Acts container now running with ID ebbba55540c7
7. Acts container ebbba55540c7 is listening on port 8001
8. 6 successful API requests made to orchestrator
9. API requests evenly distributed b/w 2 Acts containers in round-robin manner
10. Successfully crashed container 376af17b848b with /api/v1/_crash
11. Replacement Acts container running with ID 2d1ec7a114c1
12. Replacement Acts container 2d1ec7a114c1 is listening on port 8000
13. Replacement Acts container 2d1ec7a114c1 is healthy
14. Script slept for 2 minutes
15. Orchestrator successfully scaled down containers. Only 1 Acts container running with ID 2d1ec7a114c1

REFERENCES

1. Documentation of Docker : https://docs.docker.com/
2. Documentation of Flask : http://flask.pocoo.org/docs/1.0/
3. Documentation of Amazon's AWS : https://docs.aws.amazon.com/
4. Error handles and recoveries : StackOverflow, UbuntuForums, etc
5. Lecture Slides formulated by Dr.K V Subramaniam

EVALUATIONS (Leave this for the faculty)

Date	Evaluator	Comments	Score

CHECKLIST

SNo	Item	Status
1.	Source code documented	
2	Source code uploaded to CCBD server	
3	Source code in GitLab. Please do not upload your source code to github where it can be seen by everyone.	