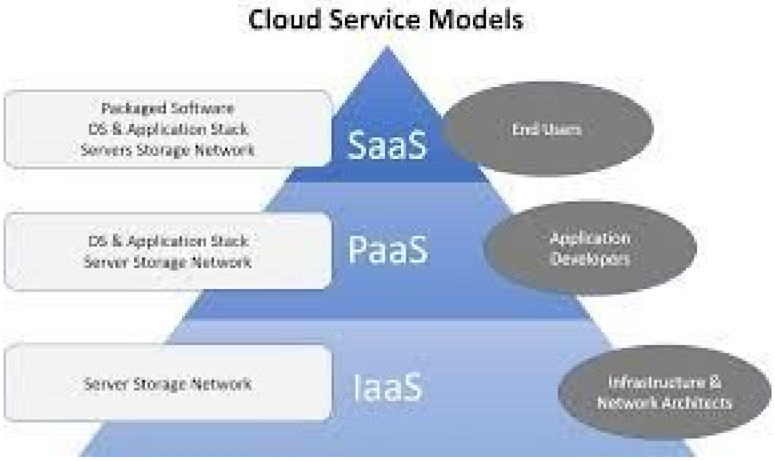
PES UNIVERSITY



CS352: Cloud Computing Jan-May 2019

Container Orchestration

Building a Fault tolerant Container Orchestrator with a load balancer, an auto scaling feature from scratch

UE16CS352 | Date- 27/04/19

SNo	Name	USN	Class/Section
1	Abhishek Narayanan	01FB16ECS016	6th sem/A
2	Abhishek Prasad	01FB16ECS017	6th sem/A
3	Abijna Rao	01FB16ECS019	6th sem/A
4	Adapa Shivani	01FB16ECS022	6th sem/A

Introduction

This project is aimed at building a container orchestrator that can perform load balancing, is highly fault tolerant, and can perform auto-scaling out or scaling in of container within an EC2 instance of Amazon AWS.

The Load balancer forwards all incoming HTTP requests equally between all running acts container. Fault tolerance is ensured by the Health Check API which monitors the health of each container by polling each container every one second, and recreates a new healthy container at the same port if a container running on a certain port was found to be unhealthy. Auto scaling api runs at every 2 minute interval and based on the no of http requests received it scales out or scales in respectively.

RELATED WORK

There are different existing literatures portraying different approaches to load balancing among multiple virtual machines or containers. The scheduling algorithm, which is the deciding factor in determining which request to the load balancer is to be routed to which node can vary from implementation to implementation. For example, requests could be routed by the load balancer based on various approaches such as Round Robin, Hashing, etc. to name a few. In our approach, for simplicity of implementation sake,we employ the round robin technique for load balancing or equally distributing incoming requests to all existent containers.

For convenience in implementation, our auto scaler feature just monitors the total number of requests every two minutes and scales out or scales in based on this. Existing auto scaling features usually are flexible and allow the user of the cloud platform to provide input rules in the form of monitor, triggers and actions wherein a user may instruct the cloud service provider to scale out or in based on selected parameters such as CPU Utilization, storage left on existing device, number of incoming requests, etc.

As an additional effort over and above the provided specifications, we try to implement in this project, a more generic auto scaler with user provided input parameters for monitors and triggers and actions.

EXPERIMENTS/DESIGN

Algorithms used for Task 1:

```
Algorithm Health_check:
// Checks if the server is functioning normally
        if ( crash == True)
                status_code ← 500
                 return status_code
        try block
                connect ← connection to database
                Insert data to database
                Select data from database
                Delete data from database
                status_code ← 200
                return status_code
         if any exception occured in try block
                status_code ← 500
                return status_code
```

```
Algorithm Crash_Server:
// Permanently disables an Acts container
        global crash ← True
        status_code ← 200
        return status_code
```

Algorithms used for Task 2:

```
Algorithm Run_Load_Balancer
    Listen on Port 80
    Accept requests from all hosts


Algorithm Load_Balancer ( request)
    global port_index, active_ports
    port_index ← (port_index + 1)%len(active_ports)
    new_host_url ← "localhost:"+ active_ports[port_index]
    url ← request.host_url replaced by new_host_url
    method ← request.method
    headers ← {key:value in request.headers such that key!="Host"}
    data ← request.get_data()
    response ← Send http request having method, headers, data to url
    return response.text, response.status_code, response.headers.items()
```

Algorithms used for Task 3:

```
Algorithm Fault_Tolerance
    global active_ports
    while (True)
        for port in active_ports
            response ← send GET request to "http://localhost:port/api/v1/_health"
            if (response.status_code == 500)
                container_id ← fetch container id of container listening through port
                stop container_id
                new_container_id ← create new container with port_no ←  port
        sleep for 1s
```

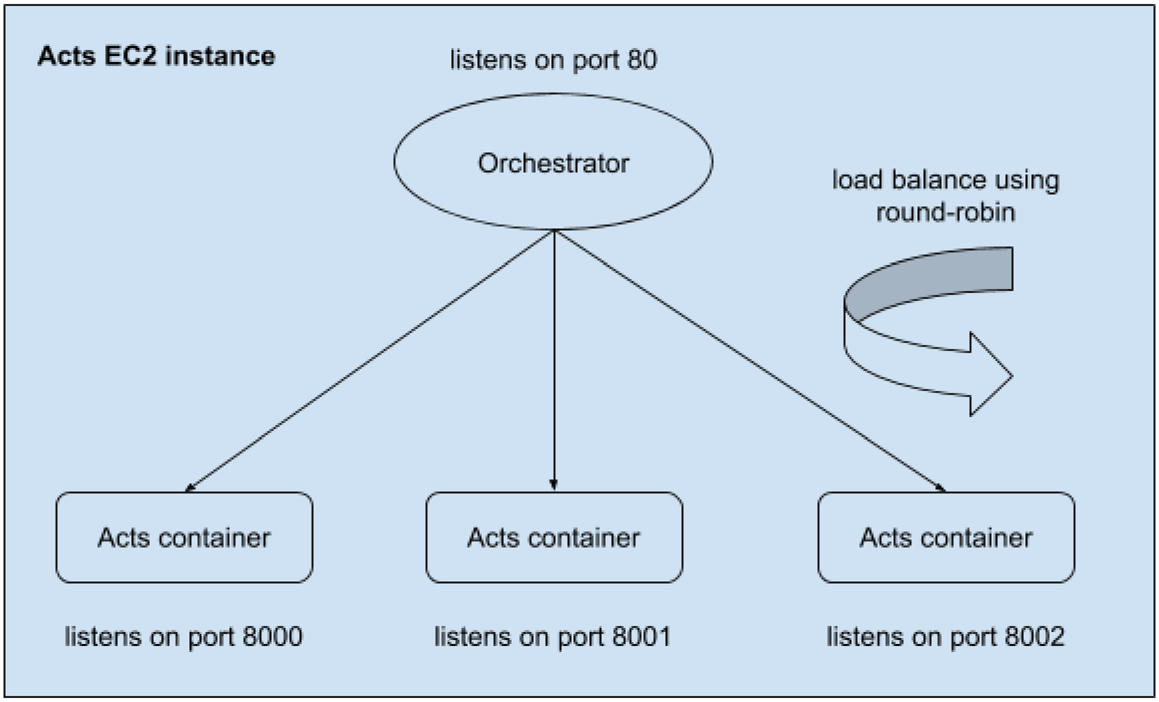
Algorithms used for Task 4:

```
Algorithm Auto_scaling
        global active_ports, total_no_of_counts
        Wait till first request is received
        Start timer
        end_request_count ← 0
        while (True):
            start_request_count ← total_no_of_counts
            no_of_required_containers=(start_request_count - end_request_count )//20 + 1
            required_container_ports ← [8000+i for i in range(no_of_required_containers)]
            if no_of_required_containers > len(active_ports)
                for each container_port in required_container_ports
                    if container_port not in active_ports
                        create new container with port ← container_port
            if no_of_containers < len(active_ports)
                for each container_port in active_ports
                    if container_port not in required_container_ports
                        remove container with port ← container_port
            end_request_count ← start_request_count
            sleep for 2 mins
```

ADDITIONAL EXTENSIONS ABOVE GIVEN SPECIFICATIONS :

We extend the auto scaling feature to be more generic, such that the scaling in and scaling out of containers occurs based on user defined rules specifying how long the number of requests parameter should be monitored, what the threshold limit for every t seconds interval should be to trigger an action (which is scale in or out) and also the scale factor by which we should scale out our containers or scale in.

Further extensions in this regard would be focussed at allowing more flexibility with the parameter to be monitored as well based on user input rather than us considering only number of requests for triggering a scaling action.

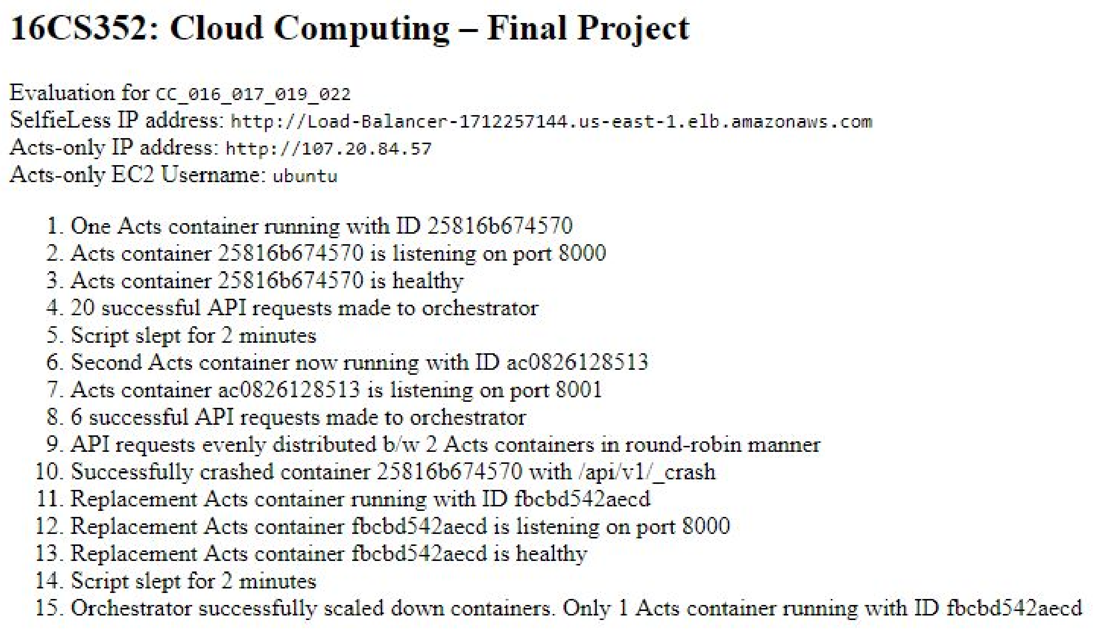


The above figure pictorially portrays the architecture of our Acts EC2 setup. In the users EC2 instance, we have only one container running with all requests related to /api/v1/users url being routed to it by the Amazon AWS Load Balancer.

TESTING/RESULTS

The features involved in the project, namely the load balancer, the health monitoring api for ensuring fault tolerance and auto scaler have been tested manually by writing a test script sending post requests for users and categories and 170 post requests for acts.

The test script for evaluation was also run locally and our cloud backend was robust enough to pass all required test cases, the screenshot of which is attached below :

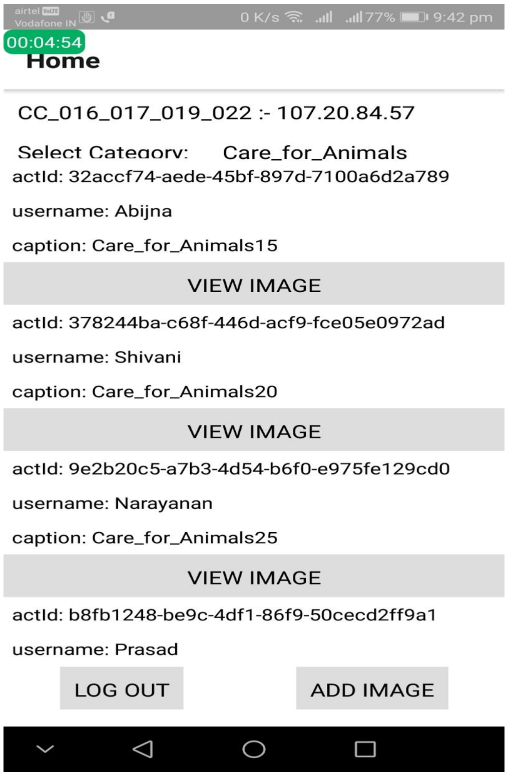


16CS352: Cloud Computing – Final Project

Evaluation for `CC_016_017_019_022`
SelfieLess IP address: `http://Load-Balancer-1712257144.us-east-1.elb.amazonaws.com`
Acts-only IP address: `http://107.20.84.57`
Acts-only EC2 Username: `ubuntu`

1. One Acts container running with ID 25816b674570
2. Acts container 25816b674570 is listening on port 8000
3. Acts container 25816b674570 is healthy
4. 20 successful API requests made to orchestrator
5. Script slept for 2 minutes
6. Second Acts container now running with ID ac0826128513
7. Acts container ac0826128513 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 25816b674570 with /api/v1/_crash
11. Replacement Acts container running with ID fbcbd542aecd
12. Replacement Acts container fbcbd542aecd is listening on port 8000
13. Replacement Acts container fbcbd542aecd is healthy
14. Script slept for 2 minutes
15. Orchestrator successfully scaled down containers. Only 1 Acts container running with ID fbcbd542aecd

Apart from this, we have also integrated the SelfieLessActs Mobile App with our cloud backend, the screenshots of which are portrayed below :



Home

CC_016_017_019_022 :- 107.20.84.57

Select Category: Care_for_Animals

actId: 32accf74-aede-45bf-897d-7100a6d2a789

username: Abijna

caption: Care_for_Animals15

actId: 378244ba-c68f-446d-acf9-fce05e0972ad

username: Shivani

caption: Care_for_Animals20

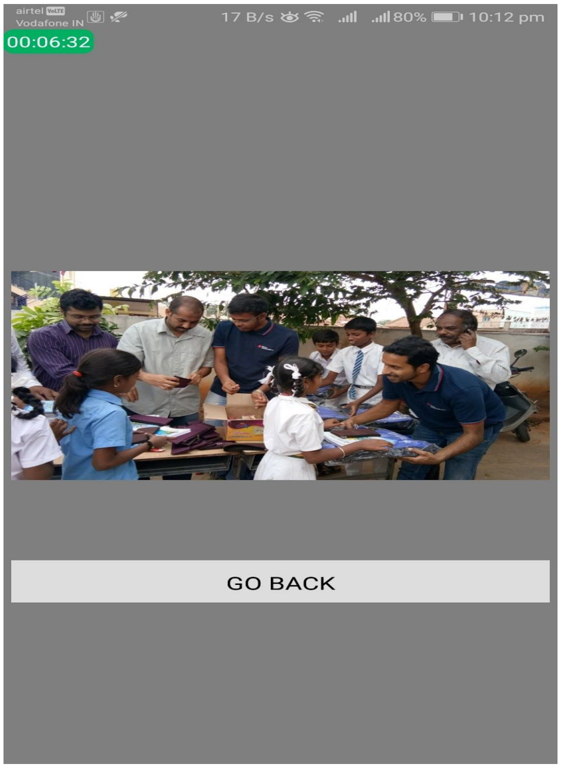
actId: 9e2b20c5-a7b3-4d54-b6f0-e975fe129cd0

username: Narayanan

caption: Care_for_Animals25

actId: b8fb1248-be9c-4df1-86f9-50cecd2ff9a1

username: Prasad



REFERENCES

The references referred to for achieving the tasks in the project are as follows ::

https://docs.docker.com/develop/sdk/examples/

https://runnable.com/docker/python/dockerize-your-python-application

http://containertutorials.com/py/docker-py.html

https://github.com/docker/docker-py

https://docker-py.readthedocs.io/en/stable/containers.html

https://www.bogotobogo.com/python/Multithread/python_multithreading_Synchronization_Lock_Objects_Acquire_Release.php

https://www.geeksforgeeks.org/multithreading-in-python-set-2-synchronization/

https://hackernoon.com/synchronization-primitives-in-python-564f89fee732

https://stackoverflow.com/questions/37144357/link-containers-with-the-docker-python-api

https://www.laurentluce.com/posts/python-threads-synchronization-locks-rlocks-semaphores-conditions-events-and-queues/

EVALUATIONS (Leave this for the faculty)

Date	Evaluator	Comments	Score

CHECKLIST

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