

CS352: Cloud Computing Jan-May 2019

Container Orchestration

REPORT

CS352 | 04 April, 2019.

| SNo | Name | USN | Class/Section |
| --- | --- | --- | --- |
| 1 | Maanvi Nunna | 01FB16ECS187 | 6/D |
| 2 | Meghana Ganesh | 01FB16ECS205 | 6/D |
| 3 | Nandagopal NV | 01FB16ECS221 | 6/D |
| 4 | P Abhishikta Sai | 01FB16ECS242 | 6/D |

# Introduction

This goal of this project was to build a container orchestrator service that can perform load balancing, auto-scaling and ensures fault tolerance.

## Related work

We mostly picked up information from the internet on things we needed to lookup. This includes docker documentation pages, flask documentation and some examples online that were helpful.

## DESIGN

The orchestrator itself is composed of three main parts:

The load balancer, which in accordance to the specification, uses the Round-Robin policy for load balancing. In other words, whenever a new request comes in, the load balancer attempts to give equal turns to each running container and does this by distributing requests in a circular queue fashion.

The fault tolerance is ensured using a heartbeat mechanism, which polls each container at a fixed frequency(of 1 second according to the spec) and in the event that a particular container fails to respond, the orchestrator initiates a replacement of that container with a new healthy container.

Lastly, the auto scaler is present to dynamically scale the number of containers up or down depending on the number of incoming requests. As per the spec, we’ve set it such that the number of containers running at any point in time is given by the formula:

N=(K/20)+1 where K is the number of requests received in the last two minutes. This scaling only happens every two minutes and not continuously. The containers are running on ports 8000 and onwards, all based on the same docker image that we created with the required flask program that serves REST API calls.

Later, to generalize this, we isolated our program from various specifics like number of containers, starting port number, heartbeat, threshold after which scaling occurs, interval of auto-scaling and so on which need not have been part of the program itself as it introduces a level of unnecessary hard-coding. These specifics were read from a JSON file. The rest of the functionality of the orchestrator remains unchanged in that it continues to comply with the spec provided.

## TESTING

The orchestrator was tested for its three main functions :

Load balancing, auto scaling and fault tolerance -all in accordance to the spec. This was verified via manual inspection with various test cases and also using the test script that we were provided with.

The error messages from the test script were inadequate at best and we had to decompile the .pyc file provided to us into a .py file(using python module- uncompyle6) so we could look into what was happening under the hood. Soon enough, we fixed the problem and passed all test cases.

Other tests included connection with the app, ensuring shared DB container was accessible to all accessing containers. We also had to check for thread synchronisation by artificially introducing a race condition.

## REFERENCES

The round robin and auto scaling policies were pretty straightforward by themselves. Our references predominantly revolved around docker errors, redirection problems and misleading error messages from the test script. We also had to synchronise threads using semaphores, for which we referred to:

<https://stackoverflow.com/questions/2332765/lock-mutex-semaphore-whats-the-difference>

Another reference we used to remove debug output prompts from docker commands:

<https://stackoverflow.com/questions/41171791/how-to-suppress-the-output-of-subprocess-run>

We figured out how to use the subprocess module from here:

<https://docs.python.org/3.6/library/subprocess.html#subprocess.run>

We used this to debug errors of docker network and connecting the shared database to containers :

<https://docs.docker.com/v17.09/engine/userguide/networking/work-with-networks/>

Used these to tackle the problem of VM instantly disconnecting after SSH:

<https://stackoverflow.com/questions/51413410/understanding-raise-remotedisconnectedremote-end-closed-connection>

<https://stackoverflow.com/questions/35552065/ssh-connection-closed-by-remote-host-having-security-group-ssh-inbound-permiss>

## EVALUATIONS (Leave this for the faculty)

| Date | Evaluator | Comments | Score |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## CHECKLIST

| SNo | Item | Status |
| --- | --- | --- |
|  | 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. |  |