

14CS341:Cloud Computing

Class Project Report

ConPaaS- Container Platform as a Service

Cloud Computing | 15-May-2017

|  |  |  |  |
| --- | --- | --- | --- |
| **SNo** | **Name** | **USN** | **Class/Section** |
| 1 | Keerthi P | 1PI14CS050 | 6th Sem ‘G’ |
| 2 | Prajwal Anand | 1PI14CS071 | 6th Sem ‘G’ |
| 3 | Priyanka Narendar | 01FB14ECS166 | 6th Sem ‘C’ |
| 4 | Shilpa S Nair | 01FB14ECS221 | 6th Sem ‘D’ |

# Introduction

# This project is about building and deploying a mini-PaaS cloud using docker containers. When a user sends a request to search for a book from a particular book store, it’s sent to a container which is scheduled by the round-robin load balancer. A database service exists, to which the search requests are sent from the containers. This service communicates with the database and sends back the results acquired from the database to the containers. The results are then shown to the user.

## Related Work

**Docker**: Containers are used to provide isolation at the application level and are designed to virtualize a single application. They, unlike VMs, do not package the whole OS, just the libraries and other dependencies the application requires. Docker is an open source software container platform that automates the deployment of applications. Docker uses resource isolation features like cgroups and namespaces to allow independent containers to run, avoiding the overhead of running and maintaining VMs.

**NodeJs**: NodeJs is an asynchronous event-driven Javascript runtime that is designed to build scalable network applications. Its advantage over traditional thread based networking is that Node does not perform any IO directly, so it does not block, which allows developing scalable systems easier.

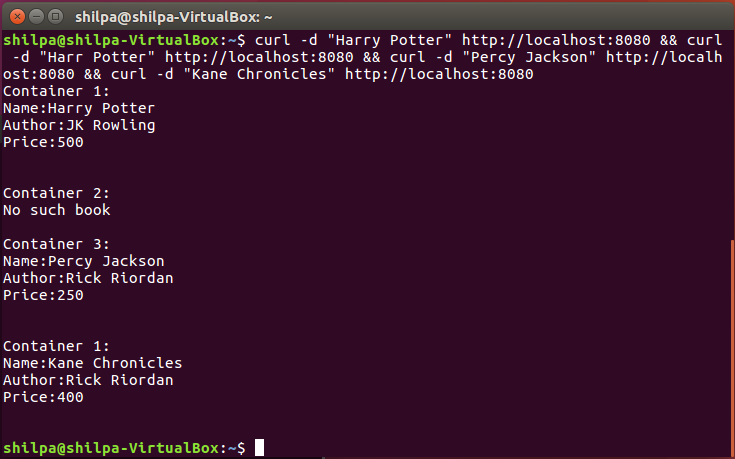
**Shared Database**: The shared database approach for multi-tenancy involves the lowest hardware and backup costs. In this approach the data of multiple tenants are stored in the same table. In the name-value pair variation for customization,there are 4 tables. The first is the main table which contains the data records of all the tenants and a name-value pair record column for the customized fields. The second is the pivot table,which contains the list of all name value pairs. The third which is the metadata table1 contains the name of the fields and their types. The last, metadata table2, contains the mapping between tenants and their id. This technique provides greater space-efficiency, but to do the query translation, a lot more joins may be required.

## ALGORITHM/DESIGN

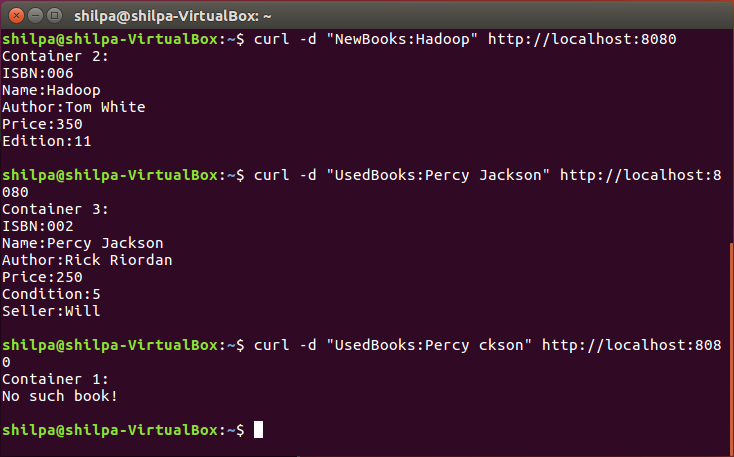
The architecture of our project consists of the following components:

1. **Load Balancer:** The load balancer receives incoming user requests, and forwards them to different server side containers. The selection of the container to which the request is forwarded, is done using a simple Round Robin scheduling algorithm. All client requests are made to the load balancer’s address, thus ensuring transparency in terms of which container is actually servicing the request. The containers have been created in such a manner as to share the network space of the host, and hence each container is accessed simply by the port number at which the server NodeJS program is listening on. The load balancer is written in NodeJS.
2. **Containers:** The server hosts 3 independent containers. Each container independently runs a NodeJS program, that takes the user’s search key (book name in this case) as input, and invokes a database service with this key. The response of the DB service is echoed to the user. The 3 containers, while otherwise completely independent of each other, connect to a shared database service. We have made use of Docker to manage these containers.
3. **Database Service:** The database service is shared by all 3 containers. There are no direct database calls supported; it is a microservice that only takes a search key as input, and the shared database is queried using this key. The search results are appropriately sent back to the container that invoked the DB service: all book details on success, and a failure message in case the book does not exist/internal server or database errors.
4. **Shared Database:** The database management system used is MySQL.  
   For Stage 3, this part is quite straightforward. The database resides in the shared memory space of the server; hence all containers will access the same data, via the database service. This ensures consistency of the database. However, there is the drawback that the database becomes a bottleneck in the application.  
   For Stage 4, this component is slightly more complex. We support different schemas for different users: UsedBooks users see 2 additional (and different) fields, whereas NewBooks users see 1 different field. The corresponding query mapping has been implemented using the Name-Value Pairs concept.

## EXPERIMENTAL RESULTS



Stage 3: Multiple requests are being sent by the client. The load balancer handles this and different servers process each request. Each server uses a common database service which handles all interaction with the MySQL database.



Stage 4: Different users search for books. The users have different schemas so their search results differ.

## FUTURE ENHANCEMENTS

1. The load balancing algorithm may be enhanced to take into account network traffic, execution time and actual server loads.
2. More microservices can be included to provide features such as inserting into and deleting from the database.
3. The database itself can be replicated on all containers, and appropriate consistency enforcing algorithms can be implemented in order to speedup the performance.

## REFERENCES

1. Using Dockers:

https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04

1. Using nodejs with mysql:

https://codeforgeek.com/2015/01/nodejs-mysql-tutorial/

http://www.w3resource.com/node.js/nodejs-mysql.php

1. Class PPTs

## 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 | Recorded video of demo |  |
| 4 | Instructions for building and running the code. Your code must be usable out of the box. |  |
| 5 | Dataset used for project uploaded. Please include a description of the dataset format. This includes input file format. |  |