Software Developer challenge

Version 0.1

March 27, 2017

Presented by: Sreejithu Panicker

72 Mccandless crt, milton, on

# Software Developer challenge

## Challenge 1

### Overview

**Q.** Read through the rest of the assignment, decide which technologies to use and explain why you chose what you chose.

This section describes about the technology choices that are used to execute these challenges. Also, describes the rationale behind each decision and their use. Following technologies are used,

* + - Docker
    - Docker registry
    - Java
    - Jetty embedded http server
    - Jax-rs Jersey (JSR 311 & JSR 339)
    - Maven
    - JSON (JSR 353 ), java api for JSON processing
    - Docker networking
    - Couchbase

Why Docker?

Container packages software with its dependency in a single shipment unit (self-contained application). I am fully utilizing it capabilities to package my application and distributes to paytm, so I don’t have to worry about the dependencies and runtime environment.

Why Docker registry?

I can push my docker images into the docker public registry, so that paytm can easily pull and run containers from these images.

Why Java?

I have more than 12 years of experience in writing programs in java, so it is my natural choice for the challenges. I could have used many other platforms such as node.js, but it may take little more time (bit of learning).

Why Jetty embedded http server?

Traditional Http/Servlet containers such as Tomcat are overkill for service based applications. Embedded http servers are quite a natural choice for development due its flexibility and programmability.

Jetty’s Solgan “Don’t deploy your application in Jetty, deploy Jetty in your application!" “

I could have used other http Server; such as, JDK HTTP Server, Grizzly HTTP Server or Netty HTTP Server , Jetty is quite popular and very stable .

Why JAX-RS Jersey?

It is one of the quite popular reference implementation for Restful architectural style. Also, it has its own packages to integrate with Jetty.

Why Maven?

I have used Maven for dependency resolution and building the “uber” jar file to deploy in the containerized solution.

Why Java api for JSON processing?

“Java api for Json processing “, provides natural way to interact with JSON object. We don’t have to build the unnecessary POJO’s for marshaling json object.

Why Docker networking?

Challenge 4 and 5 has specific requirement to persist the data, it is easy to create a Docker network and access the persistent layer through dns name. I could have used docker swarm, but docker networking can easily address the requirement.

Why Couchbase ?

Couchbase is vertically scalable and “*share nothing*” architecture allow us to scale the application vertically. It is also an ideal candidate to store profile information.

**Scalability**

Challenge 2 and 3 can be vertically scaled using any container schedulers such as , Kubernetes, cloudfountry , docker swarm etc. because app doesn’t contain any state i.e, stateless.

Challenge 4 and 5; currently app retains its session information in the server level so if we use “sticky” load balancers we can scale the application. If we externalize the session using the solutions such as redis cluster, hazlecast, memcached etc., we can easily scale the solution.

## 2 Challenge 2

### 2.1 PROBLEM:

Q. Create a deployable “Hello World” Server exposing simple REST “Hello World” API. It is going to be a base for your application for this assignment.

Solution approach:

* Create a “REST” service using JAX-RS implementation (Jersey).
* Create an embedded Http Server (Jetty) and deploy the service as part of the solution.
* Create an “uber” jar file using “Maven” and run as java program.
* Deploy java app into docker container.
* Run and expose the service to external consumption.

Code snippet:

REST service:

@Path("/helloworld")

public class HelloWorld {

@GET

@Path("greeting")

@Produces(MediaType.TEXT\_PLAIN)

public String display(){

return "Hello world";

} }

**A separate instruction file is created and uploaded in the github to setup the entire solution. Please check docker folder.**

**Running the application:**

Docker pull sreepanicker/paytmhello

Docker run –d –p 80:80 --name paytmhello sreepanicker/paytmhello

Curl –X GET <http://localhost/helloworld/greeting>



## 3. Challenge (3)

Q. Pick one of the available online API’s such as Twitter (<https://dev.twitter.com/overview/api>), LCBO (<https://lcboapi.com/>) or Weather (<https://openweathermap.org/api>), create and implement a flow involving that API and user of your application.

\* Scalability, please refer challenge 1’s scalability section.

Solution approach:

I have picked the LCBO API for creating the this solution.

* Create an account with “LCBO API” server and requested for Key/Token for accessing the service.
* Create a Web Application, which is used for user interaction.
* Create a REST API call to the LCBO API with authorization information in the header (request.header("Authorization", "Token XXXXXXXX”) )
* Use JSON parser (JSR 353) for parsing the JSON Object.
* Deploy the “WAR” file in the embedded Jetty server. Enable jetty configuration for JSP/Servlet processing.
* Create container service for the application.

Code snippet:

private JsonObject getProductDetails(String productId) {

JsonObject jsonObject = null;

//lcbo api URL

String URL ="http://lcboapi.com/products/"+productId;

//lcbo tokem

String token ="MDo1MmFmNDU2OC0........o";

Client client = ClientBuilder.newClient();

WebTarget webTarget = client.target(URL);

Builder request = webTarget.request();

request.header("Content-type", MediaType.APPLICATION\_JSON);

request.header("Authorization", "Token "+token);

Response response = request.get();

if (response.getStatus()==200){

JsonReader jsonReader = Json.createReader(new StringReader(response.readEntity(String.class)));

jsonObject = jsonReader.readObject();

}

client.close();

return jsonObject;

}

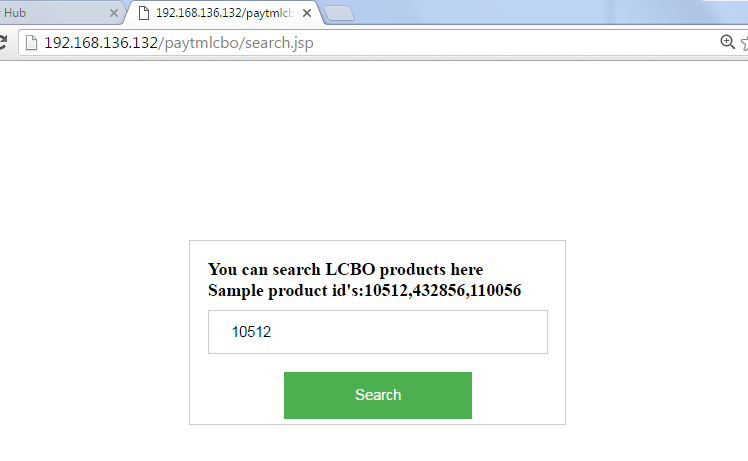
**Running the Application:**

Docker pull sreepanicker/paytmlcbo

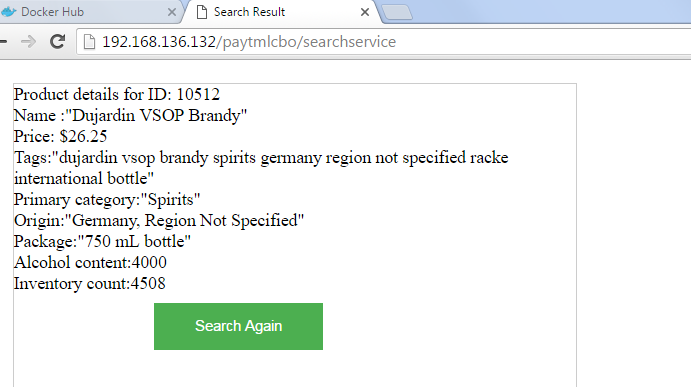
Docker run –d –p 80:80 --name paytmlcbo sreepanicker/paytmlcbo



Search screen (url) : <http://192.168.136.132/paytmlcbo/search.jsp> **(please use your(docker) host IP)**



**Search result:**



## 4. challenge 4

Q. Make your application secure and personalized by making people to have to sign-up / login. Bonus points, if users will be able to reset their passwords.

\* Security, in the current implementation user name and password are transmitted in plain text during the submit operation. This behavior can be changed by implementing channel level encryption using SSL/TLS certificate.

I have used java because of familiarity; node.js, express etc. are alternate solutions. Node.js solution could have taken more time (little bit of learning curve) so avoided.

**Solution Approach**

1. Create a web application using java servlet and jsp.
2. Store user information in the servlet context itself, data won’t persist between restarts.
3. Deploy war file in embedded Jetty server.
4. Create a container for web application.

**Running the Application:**

Docker pull sreepanicker/ paytmnoplogin

Docker run –d –p 80:80 --name paytmlcbo sreepanicker/paytmnoplogin

## 4. CHALLENGE (4 & 5)

Q. Make your application secure and personalized by making people to have to sign-up / login. Bonus points, if users will be able to reset their passwords. Challenge 5: Make your application persistent. Whatever functionality your application has, after restart, make it possible to view a history of user activity or anything else you deem necessary.

\* Scalability, please refer to challenge 1’s scalability section.

\* Security, in the current implementation user name and password are transmitted in plain text during the submit operation. This behavior can be changed by implementing channel level encryption using SSL/TLS certificate.

**Solution Approach**

1. Create a web application using java servlet and jsp
2. Couchbase provides the persistence
3. Create a data model for storing the data
4. Deploy war file in embedded Jetty server
5. Create a docker network , called “paytm”
6. Create a container for web application
7. Create a container for couchbase

Running the application

***IMPORTANT***

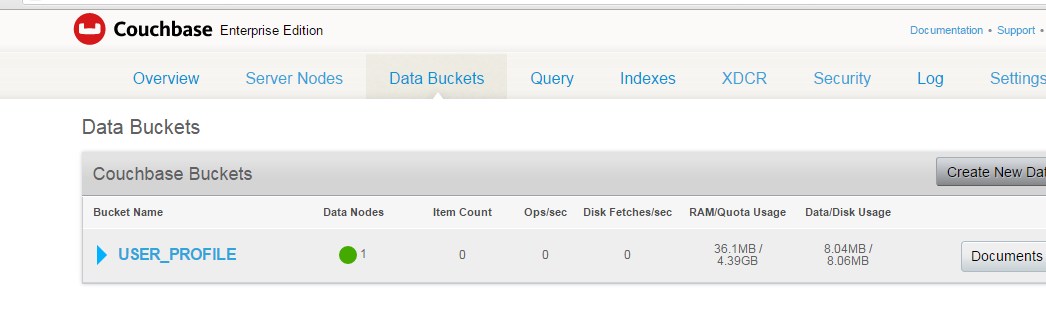
Application uses “USER\_PROFILE” bucket, and container name “paytmcouchbase” must match to run the application.

**A separate instruction file is created and uploaded in the github to setup the entire solution. Please check docker folder.**

1. Docker start paytmlogin
2. Docker start paytmcouchbase

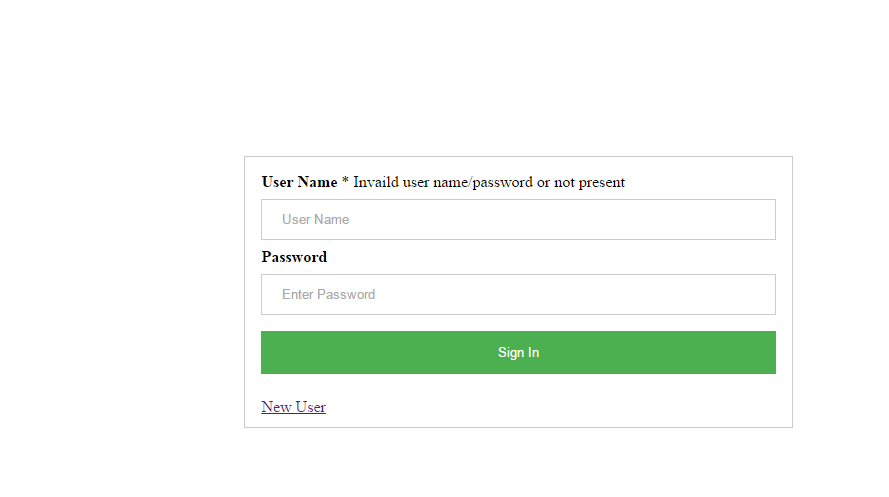


Couchbase

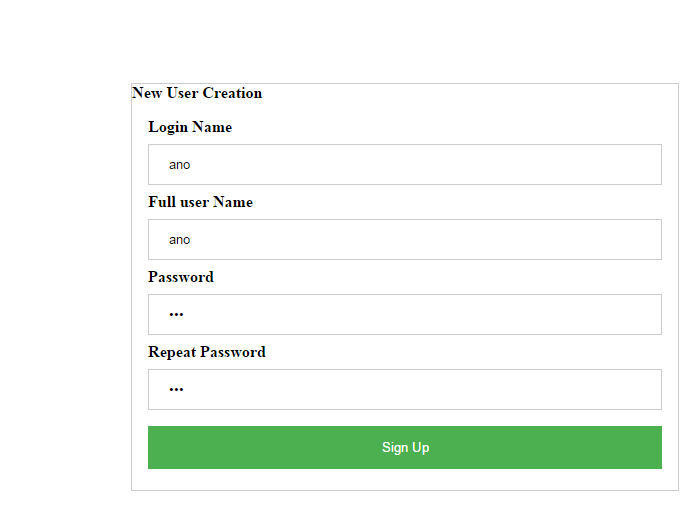


Application Screen

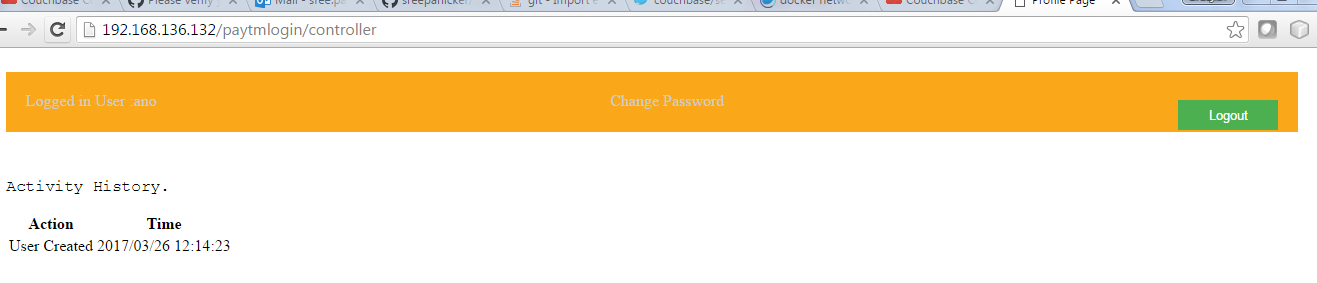
Accessing the application: <http://192.168.136.132/paytmlogin>



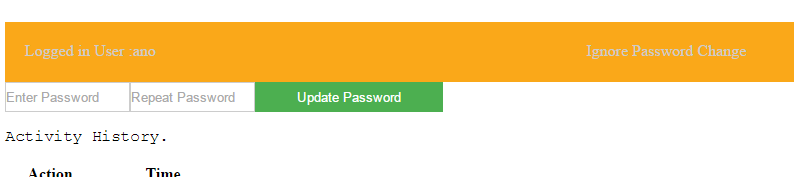
New User Screen



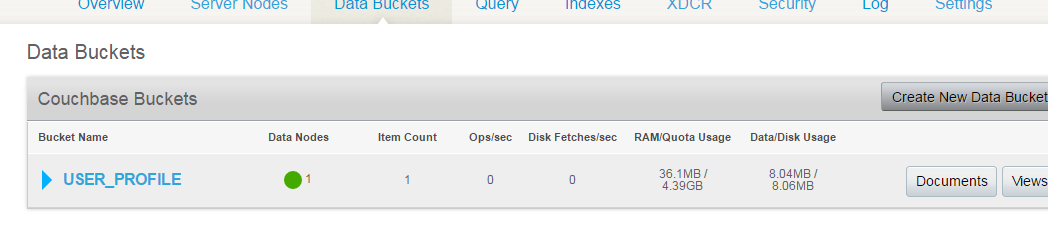
Activity Screen



Password rest screen



Couchbase Bucket count



## Challenge 6

Q. Test Application

Various screenshots are provided throughout the document. Also, unit test and integration test has been completed.

## Challenge 7

Q .Let us know how we can use it. You could either provide us with a zipped file containing your solution or a link to your Github repository containing one.

The entire solution is available in the github for download.

Github location: <https://github.com/sreepanicker/paytm-sourcecode.git>

Github structure,

* Folder case2 contains source code for REST service
* Folder case3 contains source code for API integration, LCBO
* Folder case4 contains source code for login app without persist capability.
* Folder case4&5 contains source code for login application with persist capability.
* Folder case\_server contains source code embedded jetty server to access the war file.
* Folder docker contains instructions for setting and running the various applications.
* Folder jars contains all the jars .