**Roboshop Components Setup**

**Pre-Requisites:**

1. Create a folder for Roboshop project in VS workspace.
2. Create each folder separately for each of the components inside Roboshop folder because the Dockerfiles which we are going to write should be isolated.
3. Once the component is ready, we push to the git repository and from there we clone that into the server where is git is installed.
4. Once the component is ready we use docker commands to process it.
5. **MongoDB:**

* Create a folder for MongoDB under ronoshop project folder and keep 3 files inside it.
* 1. Dokerfile – to generate an image out of it.
* 2. User.js 3.catalogue.js – these .js files contain schema information(products info + users info) that we need to load in MongoDB as pre-requisite.
* We use **COPY \*js /docker-entrypoint-initdb.d**/ to load the schema by default into the MongoDB.
* The above command tells that the js files are in the current directory where our Docker file is exists and copying those js files to the specified(Mongo default dir) directory **/docker-entrypoint-initdb.d/**
* Push the MongoDB directory into the server through git and get inside into the mongoDB directory and check for the files.
* ***docker build -t mongodb:1 .*** *– builds the image*
* ***docker run -d --name MongoDB mongodb:1*** *– creates the container*
* ***docker logs mongodb –*** *logs*
* *references*

[*https://github.com/daws-76s/roboshop-documentation/blob/master/02-mongodb.MD*](https://github.com/daws-76s/roboshop-documentation/blob/master/02-mongodb.MD)

[*https://github.com/daws-76s/roboshop-documentation/blob/master/03-catalogue.MD*](https://github.com/daws-76s/roboshop-documentation/blob/master/03-catalogue.MD)

[*https://github.com/daws-76s/roboshop-documentation/blob/master/05-user.MD*](https://github.com/daws-76s/roboshop-documentation/blob/master/05-user.MD)

1. **Catalogue:**

* Here also we need 3 files 🡪 1.Dockerfile 2.package.json 3.server.js
* Set WORKDIR as /opt/server/
* Copy 2&3files into the WORKDIR
* Setup Environmental variables
* RUN npm install
* Start the service by default and always make it run with server.js
* ***docker build -t catalogue:1 .*** *– builds the image*
* ***docker run -d --name catalogue catalogue:1*** *– creates the container*
* ***docker logs catalogue –*** *logs*
* here you will get an error in logs saying that mongo network error – to resolve it we should have our own created project network.
* **docker network create <networkName>**
* ***remove the existing mongodb and catalogue container and create new.***
* ***docker run -d --name catalogue –network=<networkname> catalogue:1*** *– creates the container*
* ***now you see mongodb connected successfully to catalogue.***

1. **Web:**

* Here we need a few static files for the web page that we need to show by default as landing page.
* All those files are zipped in <https://roboshop-builds.s3.amazonaws.com/web.zip>
* Download,unzip and place the files in a folder called static under Web component folder.
* In dockerfile, remove the default configuration and update,place roboshop.conf.
* ***docker build -t web:1 .*** *– builds the image*
* ***docker run -d --name web –p 80:80 –network=mynetwork web:1*** *– creates the container*
* check the roboshop url – publicIP:80( we can ignore 80 port in url as by default it takes 80)

\*\* Please refer the Docker Compose Documentation before proceeding further components\*\*

1. **User:**

* Create a Dockerfile and place the unzipped files from<https://roboshop-builds.s3.amazonaws.com/web.zip>
* Add user service in compose file.
* Create an image for user ***docker build -t user:1***

1. **Cart:**

* Create a Dockerfile and place the unzipped files from<https://roboshop-builds.s3.amazonaws.com/cart.zip>
* Add cart service in compose file.
* Create an image for cart ***docker build -t cart:1***

1. **Redis:**

* There is no specific configuration for Redis to create customised Dockerfile.
* Hense we add redis service directly in compose file.

\*\* the APIs mapping(proxy info) we did earlier is only for catalogue servive in web component

Map the User and cart services as well in web proxy info\*\*

**Verify the project at this time**

1. **MySQL:**

* Create a Dockerfile and place the shipping sql script files from<https://roboshop-builds.s3.amazonaws.com/shipping.zip>
* Add cart service in compose file.
* Create an image for cart ***docker build -t mysql:1***

1. **Shipping:**

* Shipping is an Java project and not like previous node js projects where we ran server.js with node CMD [“node”, “server.js”].
* There are 2 steps for any java project. 1. Build 2. Run
* For Building the java project we need JDK and Maven to get package as jar file from the java code.
* For running we do not need JDK & Maven, just we need is JRE to execute the Jar file.
* In Dockerfile we mention these 2 steps as multi stage builds where the 1 step contains the building a jar and 2nd is running the jar.
* And the container only occupies the space created for 2nd step only.
* Place the Java source project and pom.xml at you host machine where dockerfile is exists to make out a jar from it. <https://roboshop-builds.s3.amazonaws.com/shipping.zip>
* Create an image for cart ***docker build -t shipping:1***

1. **RabbitMQ:**

* For RabbitMQ database we need the default username & pwd, we can mention them in environment variables of either Dockerfile or Docker-compose.
* There is no specific configuration for RabbitMQ to create customised Dockerfile.
* Hense we add There is no specific configuration for Redis to create customised Dockerfile.
* Hense we add redis service directly in compose file along with environment variables.
* Create an image for cart ***docker build -t rabbitmq:1***

1. **Payment:**

* Payment is a python project and we need python as image.
* Create a Dockerfile and place the shipping sql script files from [**https://roboshop-builds.s3.amazonaws.com/payment.zip**](https://roboshop-builds.s3.amazonaws.com/payment.zip)
* Add cart service in compose file.
* Create an image for cart ***docker build -t payment:1***