In recently years, I have developed some proof-of-concepts or POCs for various clients using the Cloud Native Architecture, to help enterprises move their current Java or .NET applications to the Cloud. In this article, I am going to use “docker-compose” to demonstrate a typical full-stack Cloud application. Before going into technical details, here is a quick summary of the business context:

If an organization wants to work with illicit drugs in its productions, it has to apply for a permit or licence for the drugs. To do that, a complex application form has to be filled out and submitted to the Drug Authority, which will review the application and determines if a licence can be granted to the applicant.

The business objective of the POCs is to capture that complex application form effectively with Cloud friendly technologies. The POCs are developed using Angular, Angular Material, NodeJS, Express, Vertx and MongoDB. I have dockerized and pushed them to the Docker Hub. The following are the description of the dockerized application and services:

* [licence-application-web](https://hub.docker.com/r/weifang993/licence-application-web) – An Angular SPA application that captures sections of a licence application form.
* [licence-application-server](https://hub.docker.com/r/weifang993/licence-application-server) – A NodeJS and Express REST service that supports the above Web application, so that the licence application forms can be saved and retrieved. The submitted applications are saved as JSON documents in a MongoDB collection.
* [dpd-server](https://hub.docker.com/r/weifang993/dpd-server) – a Vertx Java REST service that provide drug searches over MongoDB active drugs collection. When filling out the form, the applicant needs to look up the specific drugs they would like to refer to in the application. The type-ahead fields in the Web GUI make calls to this service for drug matches.

With all the pieces in Docker Hub, all we need now is a docker-compose script document. When executed on your PC, the full-stack application will be pulled automatically from the Docker Hub and executed locally in their docker containers. You will be able to call up the licence Web application via your browser and it knows where its backend services live. The services, in turn, will know where to get data from and save data to. To perform that orchestration magic, we need a docker-compose file as follows:

# This is a licence application demo. File name: docker-compose.yml

version: "2"

services:

licence-web:

container\_name: licence-web

image: weifang993/licence-application-web

restart: always

ports:

- "4500:80"

dpd-service:

container\_name: dpd-service

image: weifang993/dpd-server

restart: always

ports:

- "8080:8080"

links:

- mongo

licence-service:

container\_name: licence-service

image: weifang993/licence-application-server

restart: always

ports:

- "3000:3000"

links:

- mongo

mongo:

container\_name: mongo

image: mongo:latest

volumes:

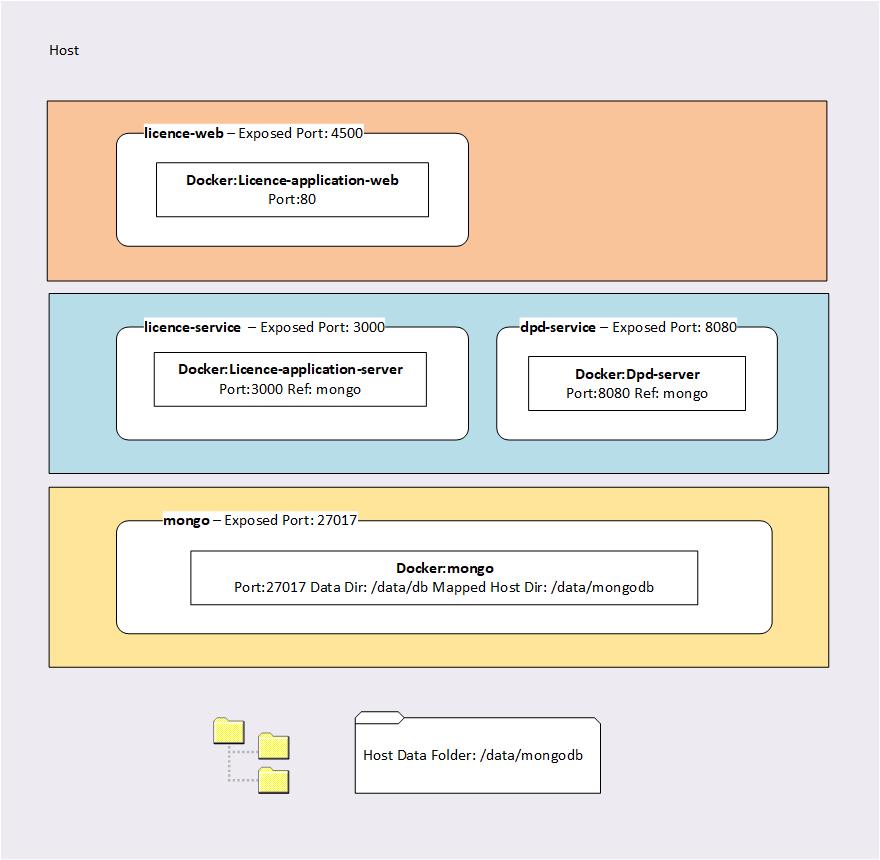
- /data/mongodb:/data/db

ports:

- "27017:27017"

Docker-compose YAML is easy to understand. Docker maintains good documentation on the subject. So far, docker-compose has evolved to version 3. I am using version 2 to keep things simple for this demonstration. For the gist of it, everything in docker-compose is defined as a “service”. Each service is provisioned by a container and each container is based on a docker image. In this demonstration, the images are pulled or downloaded from Docker Hub.

The following gives you a visual representation of the docker-compose definition:



As described in the diagram, docker-compose has wired the docker-compose “services” into a multi-tiered full-stack application. The exposed ports are accessible from the host. For that reason, you can use a browser to access the Angular application and REST services. You can also use a Mongo GUI or client to access the MongoDB. Due to the persistence mapping in the docker-compose script, all the mongo docker data will be saved to the host directory. This configuration will make sure the data you work with will survive docker-compose sessions. You can restart the docker-compose containers and continue to work with data from the previous sessions. That mapping will also allow you to copy data from the host file system to the mongo docker container very easily. We will come back to that later.

Now it is show time. Save the