Running LogicalDOC with Docker

Recently, I have discovered LogicalDOC, (http://www.logicaldoc.com), and because Document Management Systems plays an important role in organizations, I wanted to see how well runs in Docker. Fortunately, LogicalDOC has a community edition, and that was the version used for Docker integration.

Here is a feature comparison, between Enterprise, Business and Community editions:

http://www.logicaldoc.com/product/features.html

LogicalDOC CE is hosted at SourceForge, https://sourceforge.net/projects/logicaldoc/ and it comes in different bundles, with an installer, bundled with Tomcat or as a .war file. The easiest way to get LogicalDOC installed is to get the bundle with installer. I have tried on my local computer, and at the end the installer asked if I want to create an auto-install.xml file for automatic installation. This opened the way for an easy install in Docker.

Short introduction to Docker

For those who are not familiar with Docker, I should say that Docker is derived from linux containers, an operating-system-level method for running multiple isolated Linux systems (containers) on a control host using a single Linux kernel. (https://en.wikipedia.org/wiki/LXC).

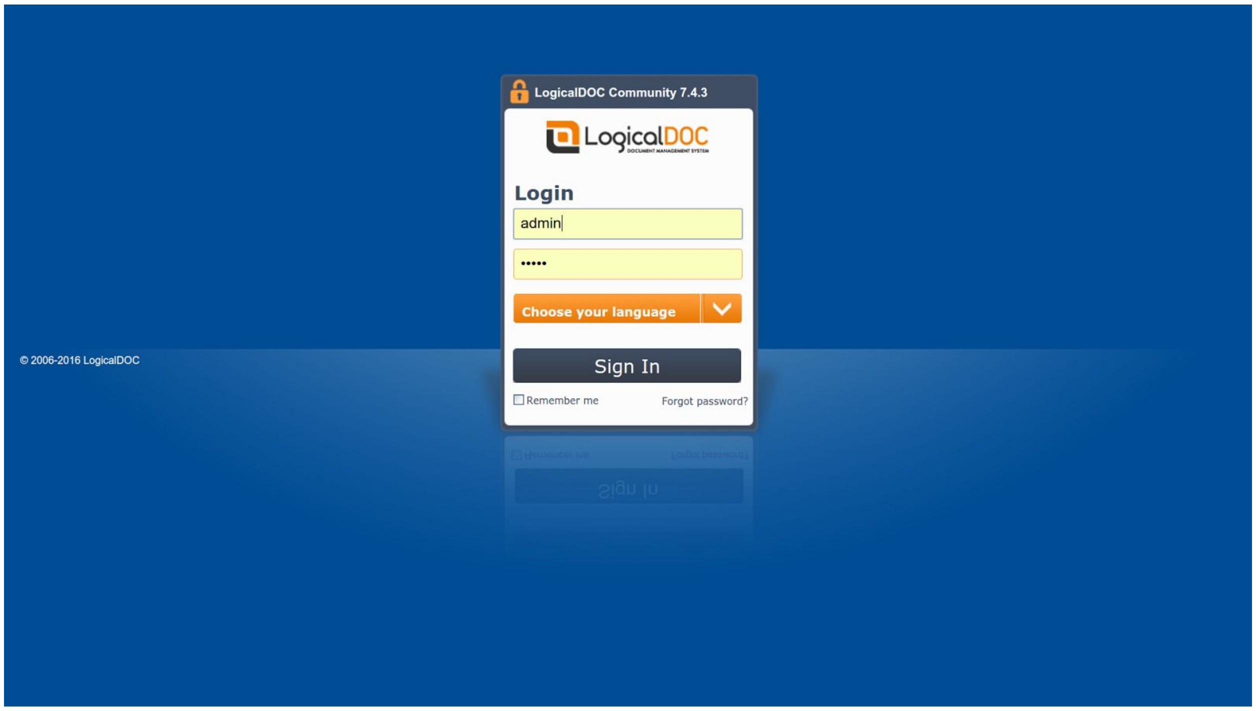
Docker uses LXC, and is a way to package and ship applications in containers. Once the application is packaged, it is very easy to run on every linux machine where Docker is installed (it runs on Windows and Mac also, but with virtualization, because Linux kernel is needed).

Docker is not only a great way to ship applications, is also a great way to run applications at a scale, on cloud, on VM or bare metal.

If you already have Docker, because I have packaged LogicalDOC as a Docker container, you can simply run:

```
$sudo docker run -p 8080:8080 mcsaky/logicaldoc-mysql
```

This will fetch the image from Docker hub and start it locally, running the required scripts to setup MySQL and LogicalDOC, will publish the port 8080 to localhost:8080, and when is ready, you can use the browser to connect to http://localhost:8080 with user admin, password admin to use the application. Very easy, very simple.



Let’s have a look at how an all-inclusive image is built to allow LogicalDOC to run in one container. The project is on github (https://github.com/mihaics/logicaldoc-mysql), and Docker Hub automatically builds the image from this project, so the image you get to run is a trusted build, with no hidden features.

Dockerfile is the building block for the image:

```
FROM phusion/baseimage:0.9.18
```

Every Docker image starts with a base image. This can be an official operating system base image (like Ubuntu) or an already tweaked image. I have choosed phusion/baseimage (based on Ubuntu) for an all-inclusive installation, because it fixes some important problems with the official images:

- has a proper init system for Docker, which correctly stops the daemons, see https://blog.phusion.nl/2015/01/20/docker-and-the-pid-1-zombie-reaping-problem/
- has cron and logging
- It is easy to configure and set up

In the next steps we will have a look at how the required software (MySQL server, utils and libraries like Java, LogicalDOC installer) is brought together in a new image. The Dockerfile is: https://github.com/mihaics/logicaldoc-mysql/blob/master/Dockerfile

For example, this snippet installs Java from Oracle:

```
 RUN echo debconf shared/accepted-oracle-license-v1-1 select true |
debconf-set-selections && \
 echo debconf shared/accepted-oracle-license-v1-1 seen true | debconf-set-selections
&& \
 add-apt-repository -y ppa:webupd8team/java && \
 apt-get update && \
 apt-get install -y oracle-java7-installer
```

And this part is downloading LogicalDOC installer:

```
RUN mkdir /opt/logicaldoc
RUN curl -L
http://netcologne.dl.sourceforge.net/project/LogicalDOC/distribution/LogicalDOC%20CE%207.4/logi
caldoc-community-installer-7.4.3.zip \
   -o /opt/logicaldoc/logicaldoc-community-installer-7.4.3.zip  && \
   unzip /opt/logicaldoc/logicaldoc-community-installer-7.4.3.zip -d /opt/logicaldoc && \
   rm /opt/logicaldoc/logicaldoc-community-installer-7.4.3.zip
```

But the “real magic” is done by the scripts in /etc/my_init:

```
ADD 01_mysql.sh /etc/my_init.d/
ADD 02_logicaldoc.sh /etc/my_init.d/
```

These two scripts will configure MySQL (root password, database creation, user creation) and LogicalDOC by running the automatic install process:

```
java -jar /opt/logicaldoc/logicaldoc-installer.jar /opt/logicaldoc/auto-install.xml
```

So, the important file here is auto-install.xml (which can be found on my github repository: https://github.com/mihaics/logicaldoc-mysql/blob/master/auto-install.xml) and can be generated manually running the installer (you will be asked at the end of installation if you want to generate an autoinstall file for other identical installs)

We have two more important files:

```
#mysql service setup
RUN mkdir /etc/service/mysqld/
ADD mysqld.sh /etc/service/mysqld/run

#LogicalDOC service setup
RUN mkdir /etc/service/logicaldoc/
ADD logicaldoc.sh /etc/service/logicaldoc/run
```

Here we define two services, for MySQL and LogicalDOC Tomcat instance. The start and if necessary, automatically restart, of the processes is done by the init system in phusion baseimage, allowing us to keep everything simple:

```
/opt/logicaldoc/tomcat/bin/catalina.sh run
```

Persistence

Persistence with the all inclusive image is very easy, there have been already defined volumes for LogicalDOC repository, indexer and MySQL database in Dockerfile:

```
#volumes for persistent storage
VOLUME /var/lib/mysql
VOLUME /opt/logicaldoc/repository
```

Create your Docker volumes:

```
$sudo docker volume create --name repodata
$sudo docker volume create --name mysqldata
```

And here is a docker-compose.yml file as a running example:

https://github.com/mihaics/logicaldoc-mysql/blob/master/docker-compose.yml

Copy docker-compose.yml file and run $docker-compose up in the same directory.
Of course, there are many other ways, map the volumes to the local filesystem, use only docker command line parameters for mounting volumes etc.

How to extend the image

Installing LogicalDOC using auto-install.xml in an single all purpose image was very easy, but in a scalable environment, we will need to use separate instances for database server and application server. Even more, we may need to use more than one instance for LogicalDOC, deployed in cluster configuration, with load-balancing front ends. Some knowledge about Docker networks, bridging and container link is required, but at the end we will have a very flexible and scalable architecture.
A big role in setup is played by environment variables. When Docker links two images, environment variables are passed. Also, we can use environment variables to discover services.

Let's have a look at the mysql initialization script:
https://github.com/mihaics/logicaldoc-mysql/blob/master/01_mysql.sh

We can already set the MySQL root password, logicaldoc database, user and password, and there are some defaults when we don't want to change them:

```
if [ -z "$MYSQL_ROOT_PASSWORD" ]; then
   MYSQL_ROOT_PASSWORD=mysqlroot
 fi
 if [ -z "$MYSQL_DBNAME" ]; then
   MYSQL_DBNAME=logicaldoc
 fi
 if [ -z "$MYSQL_DBUSER" ]; then
   MYSQL_DBUSER=logicaldoc
 fi
 if [ -z "$MYSQL_DBPASS" ]; then
   MYSQL_DBPASS=logicaldoc
 fi
```

But how about the auto-install.xml file? Actually this is very easy, if we use templates. For example in Dockerfile we need to install jinja2 template engine and jinja2 client:

```
RUN apt-get -y install python-jinja2 python-pip
RUN pip install j2cli
```

and the auto-install.j2 file should contain:

```
<entry key="dbpassword" value="{{ MYSQL_DBPASS | default('logicaldoc') }}"/>
<entry key="dbhost" value="{{ MYSQL_DBHOST | default('localhost') }}"/>
<entry key="dbport" value="3306"/>
<entry key="dbdatabase" value="{{ MYSQL_DBNAME | default('logicaldoc') }}"/>
<entry key="dbusername" value="{{ MYSQL_DBUSER | default('logicaldoc') }}"/>
<entry key="dbinstance" value=""/>
```

And before installation, we can dynamically generate the auto-config.xml file from template:

```
j2 /opt/logicaldoc/auto-install.j2 > /opt/logicaldoc/auto-install.xml
```

Clustering with Docker

LogicalDOC can be configured as a cluster because the repository and the index can be accessed by many instances. The indexing process can be enabled on just one node of the cluster or on many nodes. There are solutions for database master-master clusters, so there is no problem to deploy LogicalDOC in a Docker cluster environment like swarm, kubernetes or rancher.

More on running LogicalDOC clusters can be found on the website:
http://docs.logicaldoc.com/en/clustering

To have a complete cloud running environment we will need: mysql images (percona-galera for master-master clusters) or an external mysql server, LogicalDOC images able to discover the environment (using jinja templates, as explained) and some frontend for load-balancing and caching. Because we have a web application, nginx image is recommended. And because we want encryption at transport level, we can get letsencrypt certificates for nginx.

Conclusions

LogicalDOC is a solution for archiving large amount of data, and because it is a web application and already has clustering capabilities, it is easy to deploy in Docker and in Docker clusters.