Table of Contents

Overview 2
- Welcome to the Documentum and Docker Lab 2
- The Labs 2
- Overview 2
- Assumptions 2

Step 1 - Accessing Your Windows Image 3

Step 2 - Checking the Windows Server 4
- Using the Quickstart Terminal 6
- More Docker Commands 10

Step 4 - Pulling a Docker Image 12
- Where do I get information about a container? 13
- Accessing A Container 14

Step 5 - Loading the Documentum Images 15
- Documentum 15
- Documentum Administrator 16

Step 6 - Running the Documentum Images 16
- Documenum Content Server 16

Step 7 - Deploying DA 17

Step 8 - Cleaning Things Up 18
- Remove Containers 18
- Remove Images 19

Step 9 - Creating a custom Docker DA Image 19
- What is a Dockerfile? 19
- Create a new Dockerfile 20
- Create the entrypoint file 21

Overview

Welcome to the Documentum and Docker Lab

This EMC lab will walk you through using Documentum 7.3 (Content Server and DA) and Docker; this lab includes a Windows server image that has Docker pre-installed. During the lab, we will load and run Documentum Docker images and create a custom Dockerfile. Along the way, we will learn some basic Docker commands.

Docker is an open platform for developers to build, ship, and run distributed applications. Docker can be described as a container technology consisting of the Docker Engine, a portable, lightweight runtime and packaging tool. Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments. As a result, IT can ship faster and run the same app, unchanged, on laptops, data center VMs, and any cloud.

The Labs

In these labs, we will cover several concepts, which include –

1. Logging into the Windows environment
2. Pulling Docker images
3. Working with Docker Images
4. Working with Docker Containers
5. Using the Documentum Docker images
6. Working with a custom Dockerfile

Overview

In these labs, we will be using Documentum 7.3 (Content Server and Documentum Administrator[1]) and the latest release of Docker. We will also be using a Windows server which serves as the Docker host.

Assumptions

The labs will assume that you have some Documentum experience and are familiar with basic Documentum concepts (DA, DQL, Content Server, et al); some Docker skills are helpful but not required. We will assume you are new to Docker.

[1] Also called 'DA' throughout this lab

Step 1 - Accessing Your Windows Image

Docker can run on many platforms, including Linux, Mac and Windows. For this lab, we are using a Windows based image (Windows Server 2012) hosted on vCloud Air.

You will access the Windows image via a web browser (similar to RDP).

Note: Your Hackathon lab instructor will provide you with the URL and login details to access your Windows image.

Once you have your login credentials, open the URL in your web browser. You will see the following prompt -

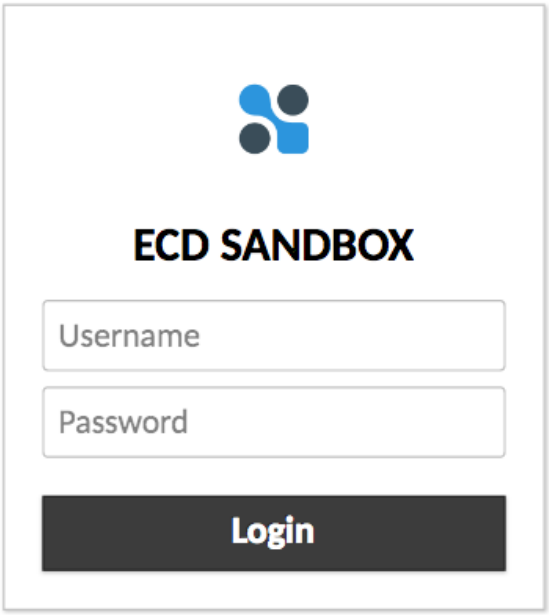

Figure 1 - Login to the Cloud Environment

After you enter in your vCloud Air credentials, you will be prompted to access the Windows server (shown below). At this point, enter in your Windows server password -

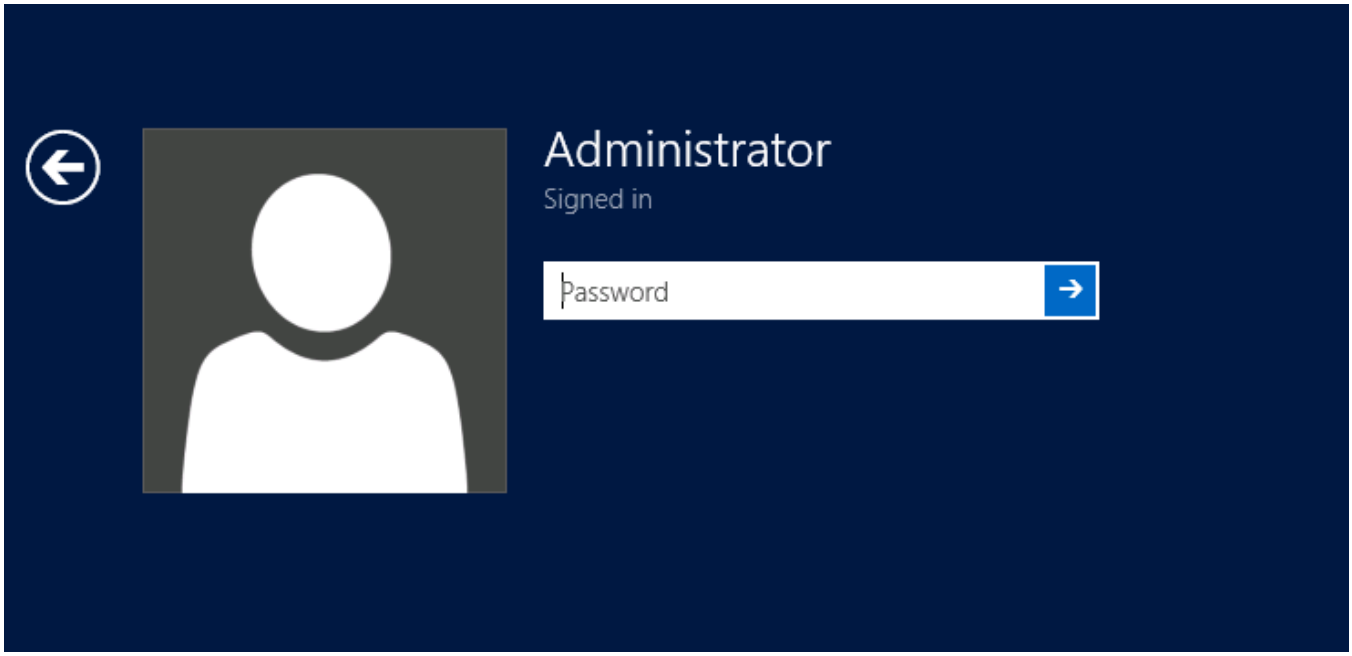

Figure 2 - Log into Windows

Step 2 - Checking the Windows Server

Once you are logged into your Windows image, let's take a tour to get you familiar with the pre-installed software. You will notice some new icons that you may not be familiar with (as shown below). For starters, you will see an icon called 'Docker Quickstart Terminal'. You will also see an icon for VirtualBox and Kitematic. All three of these icons are provided by Docker when installed.

For this lab, we will be using the Docker Quickstart Terminal quite a bit.

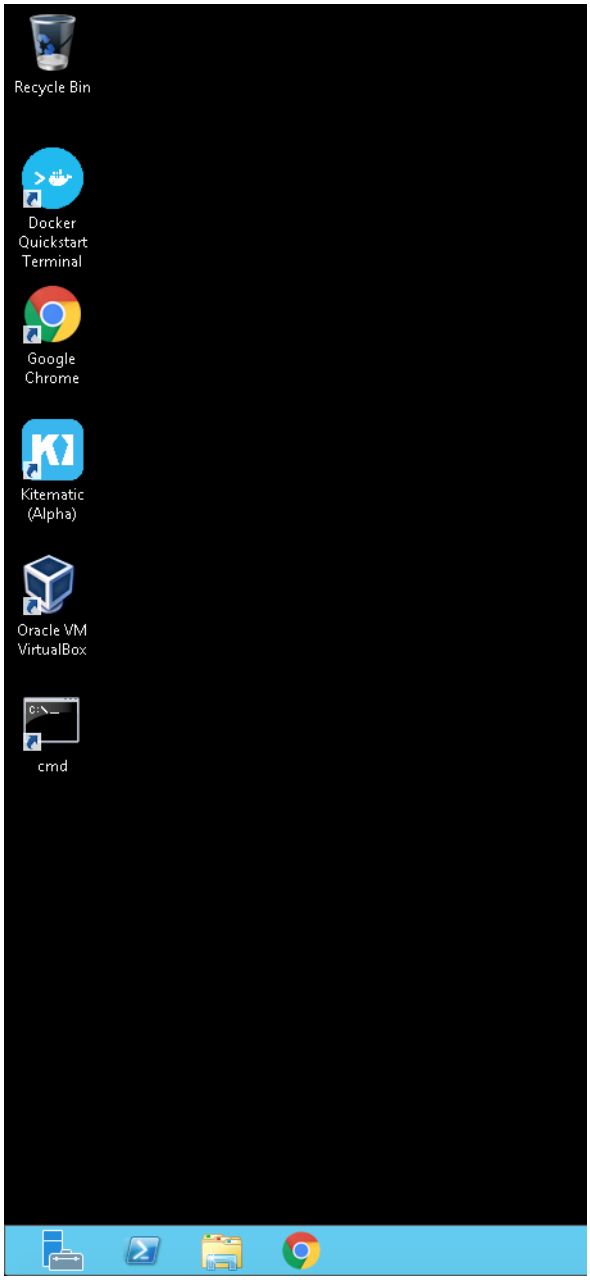

Figure 3 - Windows icons

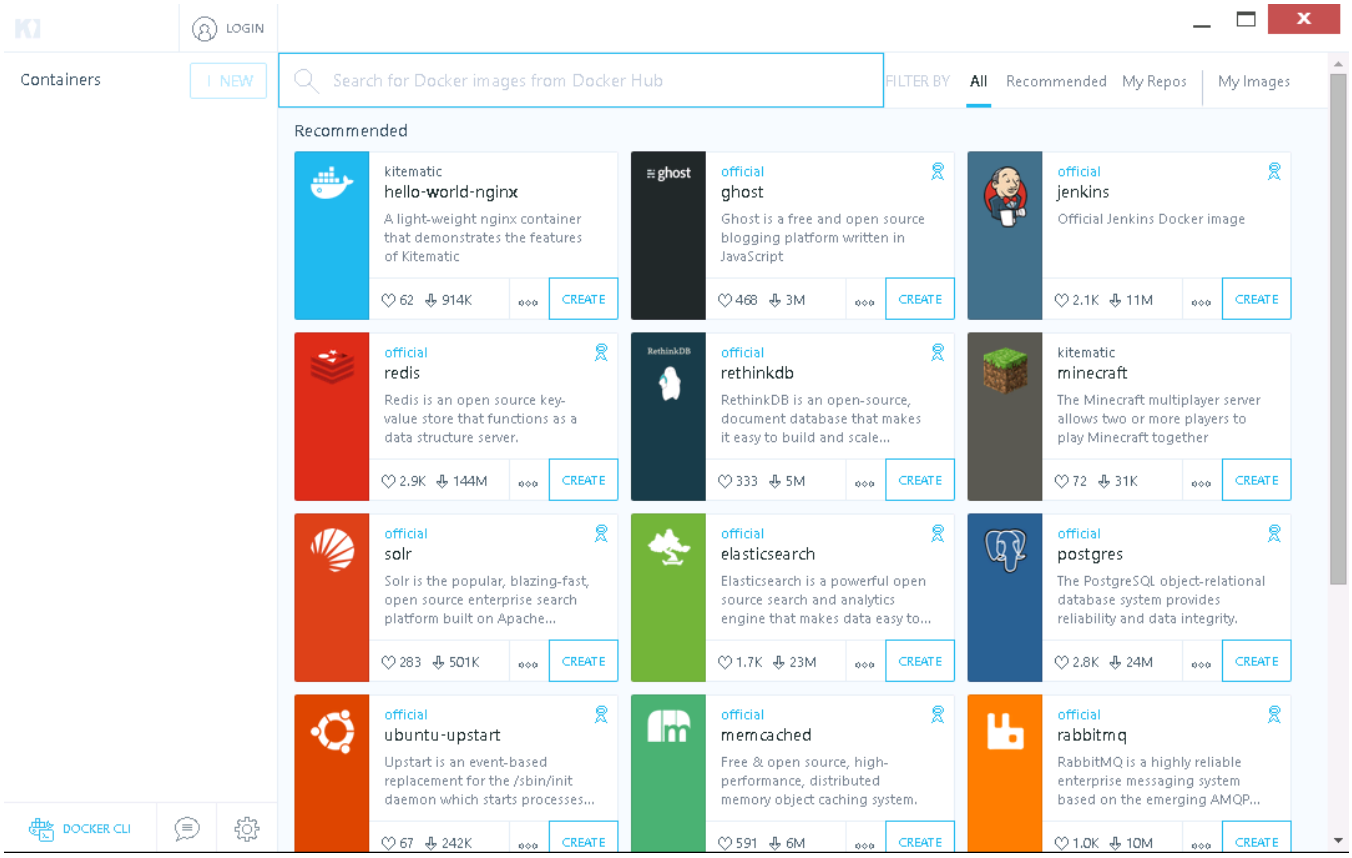

Figure 4 - Kitematic

Using the Quickstart Terminal

To start using Docker, let's open the Docker Quickstart Terminal program; to do this, click on the icon (shown below)-

Figure 5 - Docker Quickstart

This will open the Docker terminal (shown below) – the Docker Terminal is a command line tool to manage Docker (run images, start/stop containers, and much more).

Note: Take notice the IP address shown in the Docker Terminal. Please write this down as we will use it again.

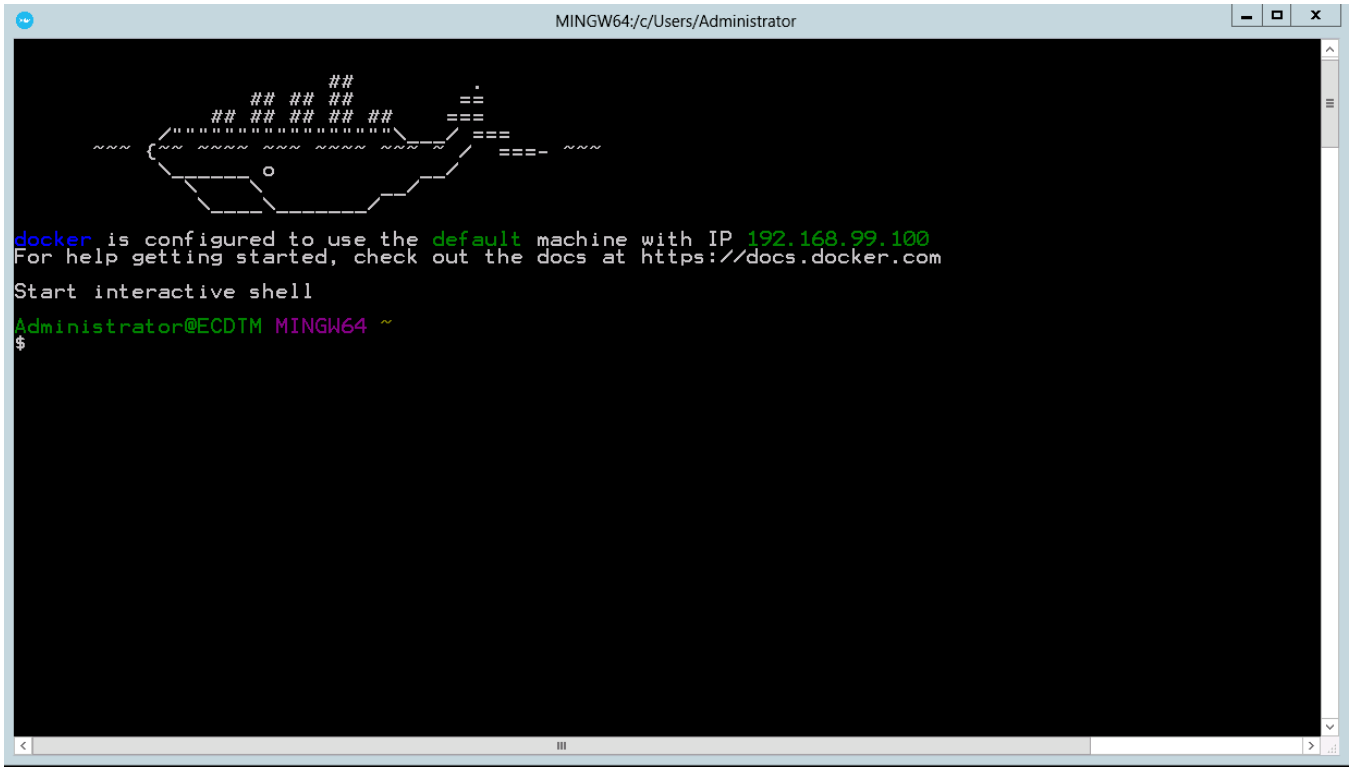

Figure 6 - Docker Terminal

Now that we are running Docker, we can run some Docker commands to check out the environment. Docker has three commands that provide some useful information. From the command prompt, enter in the following command to see the Docker version –

$ **docker version**

```
MINGW64:/c/Users/Administrator

docker is configured to use the default machine with IP 192.168.99.100
For help getting started, check out the docs at https://docs.docker.com

Start interactive shell

Administrator@ECDTM MINGW64 ~
$ pwd
/c/Users/Administrator

Administrator@ECDTM MINGW64 ~
$ docker version
Client:
 Version:      1.12.2
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   bb80604
 Built:        Tue Oct 11 17:00:50 2016
 OS/Arch:      windows/amd64

Server:
 Version:      1.12.2
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   bb80604
 Built:        Tue Oct 11 17:00:50 2016
 OS/Arch:      linux/amd64

Administrator@ECDTM MINGW64 ~
$ _
```

Figure 7 - Docker version

Let's try another command. From the command prompt, enter in the following command –

$ docker –v

```
Administrator@ECDTM MINGW64 ~
$ docker -v
Docker version 1.12.2, build bb80604

Administrator@ECDTM MINGW64 ~
$ _
```

Figure 8 - Docker v command

Figure 9 - Docker version command

Lastly, let's show the info command. From the command prompt, enter in the following command –

$ docker info

```
MINGW64:/c/Users/Administrator
Administrator@ECDTM MINGW64 ~
$ docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 1.12.2
Storage Driver: aufs
 Root Dir: /mnt/sda1/var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 0
 Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: host overlay null bridge
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Security Options: seccomp
Kernel Version: 4.4.24-boot2docker
Operating System: Boot2Docker 1.12.2 (TCL 7.2); HEAD : 9d8e41b - Tue Oct 11 23:40:08 UTC 2016
OSType: linux
Architecture: x86_64
CPUs: 1
Total Memory: 995.9 MiB
Name: default
ID: TUEI:7IPG:MYBO:2YGN:M34X:4KUX:66BK:PQAV:VR7E:EPFC:LJAI:YNKO
Docker Root Dir: /mnt/sda1/var/lib/docker
Debug Mode (client): false
Debug Mode (server): true
 File Descriptors: 13
 Goroutines: 23
 System Time: 2016-10-27T20:15:14.534702247Z
 EventsListeners: 0
Registry: https://index.docker.io/v1/
Labels:
```

Figure 10 - Docker info command

We just verified that the environment is working and saw some details on the configuration. Let's pull our first Docker image by running the Docker hello-world image; this command will use the Docker '*run*' command. Docker run is used to run Docker images; if the images does not exist, it will pull the image from Docker Hub (or a local repository). From the command prompt, enter in the following command –

$ docker run hello-world

```
MINGW64:/c/Users/Administrator

Administrator@ECDTM MINGW64 ~
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world

c04b14da8d14: Pull complete
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdcf9431a1feb60fd9
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
 https://hub.docker.com

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/


Administrator@ECDTM MINGW64 ~
$ _
```

Figure 11 - Docker hello-world example

So what just happened? We ran the Docker '*run*' command and our Docker client '*pulled*' the Docker image to our server. The Docker image came from Docker Hub, which is a public Docker registry. The '*run*' command also launched the image and we have a running container.

More Docker Commands

Let's take a look at the Docker images we have on this server by using some common Docker commands – you will use these commands quite often; From the command prompt, enter in the following command –

$ docker images

```
Administrator@ECDTM MINGW64 ~
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              c54a2cc56cbb        3 months ago        1.848 kB

Administrator@ECDTM MINGW64 ~
$ _
```

Figure 12 - Docker images command

You should only see one image, which is the hello-world image. You can see that it has an Image Id, when it was created and the size. In the screen shot above, you see more than one image, but this gives you an idea on what it looks like with more than one image.

Since we ran the Docker '*run*' command and launched a container, how do we see what images are running (aka containers)? Let's use the Docker '*ps*' command. From the command prompt, enter in the following command –

$ docker ps

What? There are no running containers! This is because the hello-world container ran and then exited - it did not have a long running process. But, there is a command we can use to see Docker containers and when they ran - and of course, re-start them. From the command prompt, enter in the following command –

$ docker ps -a

Figure 13 - Docker ps -a command

Now we can see that the hello-world image was launched and created a container id of '*00619815bf5b*' – this is important since we will use container ids in the future. To re-cap, '*docker ps*' will show only running containers by default. To see all containers, use the command '*docker ps –a*'.

Note: your container Id's will be different!

Let's start the hello-world container again. First, let's get the id for the container. From the command prompt, enter in the following command –

$ docker ps –a

Find the container id, which is the first column of the results.

From the command prompt, enter in the following command by passing in the container id (there is a shortcut to entering in all of the id pass in the first two letters of the container id) –

$ docker start <id>

The container will start and pass back the id. To check that the container ran, let's check the logs.

From the command prompt, enter in the following command –

$ docker logs –f <id>

You should see the hello-world message you saw earlier when you ran the Docker '*run*' command. If you run another Docker '*ps*' command, you will see that the container started and then exited.

Step 4 - Pulling a Docker Image

Earlier, we ran the Docker '*run*' command and pulled a hello-world image; we will now pull a CentOS Docker image. We will use the Docker '*run*' command but will include some additional commands.

From the command prompt, enter in the following command –

$ docker run -it centos /bin/bash

```
MINGW64:/c/Users/Administrator

Administrator@ECDTM MINGW64 ~
$ docker run centos /bin/bash
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos

8d30e94188e7: Pull complete
Digest: sha256:2ae0d2c881c7123870114fb9cc7afabd1e31f9888dac8286884f6cf59373ed9b
Status: Downloaded newer image for centos:latest

Administrator@ECDTM MINGW64 ~
$ _
```

Figure 14 - Pulling a CentOS image

What just happened?

Note: Notice that the prompt changed, you are now inside of the container!

```
Administrator@ECDTM MINGW64 ~
$ docker run -it centos /bin/bash
[root@b744091f1bce /]# _
```

Figure 15 - Inside of the new CentOS image

We pulled the CentOS image from Docker Hub and it downloaded to our server. As you can see from the output, it could not find the image locally (if it did exist, this download would be much faster). When we inspect the Docker '*run*' command, we can see some new parameters. Let's examine them.

- The –i flag specifies to keep STDIN open even if not attached
- The –t flag specifies to allocate a pseudo-tty
    - In short, we want an interactive session with the container.
- The /bin/bash parameter specifies we want to actually bash into the container

If you run a '*ls*' command, you will see the CentOS container, not the local Ubuntu server! This is an actual CentOS Linux server running in Docker.

From the command prompt, enter in the following command –

$ ls

You can now see the files inside of the container.

From the command prompt, enter in the following command –

$ exit

When you exit from a container, you will stop it by default. So if you do a Docker '*ps*' command, you won't see the running image. Keep in mind there are ways to run images in the background.

From the command prompt, enter in the following command –

$ docker ps –a

Find the container id for the CentOS image

From the command prompt, enter in the following command –

$ docker start <container_id>

The container is now running, which you can verify by running a Docker '*ps*'.

Where do I get information about a container?

Let's get some information about the running container by using a new Docker command called 'inspect'. All we need is the container id.

NOTE: Remember that we don't need the full container id, you just need the first two letters of the container!

From the command prompt, enter in the following command –

$ docker inspect <id>

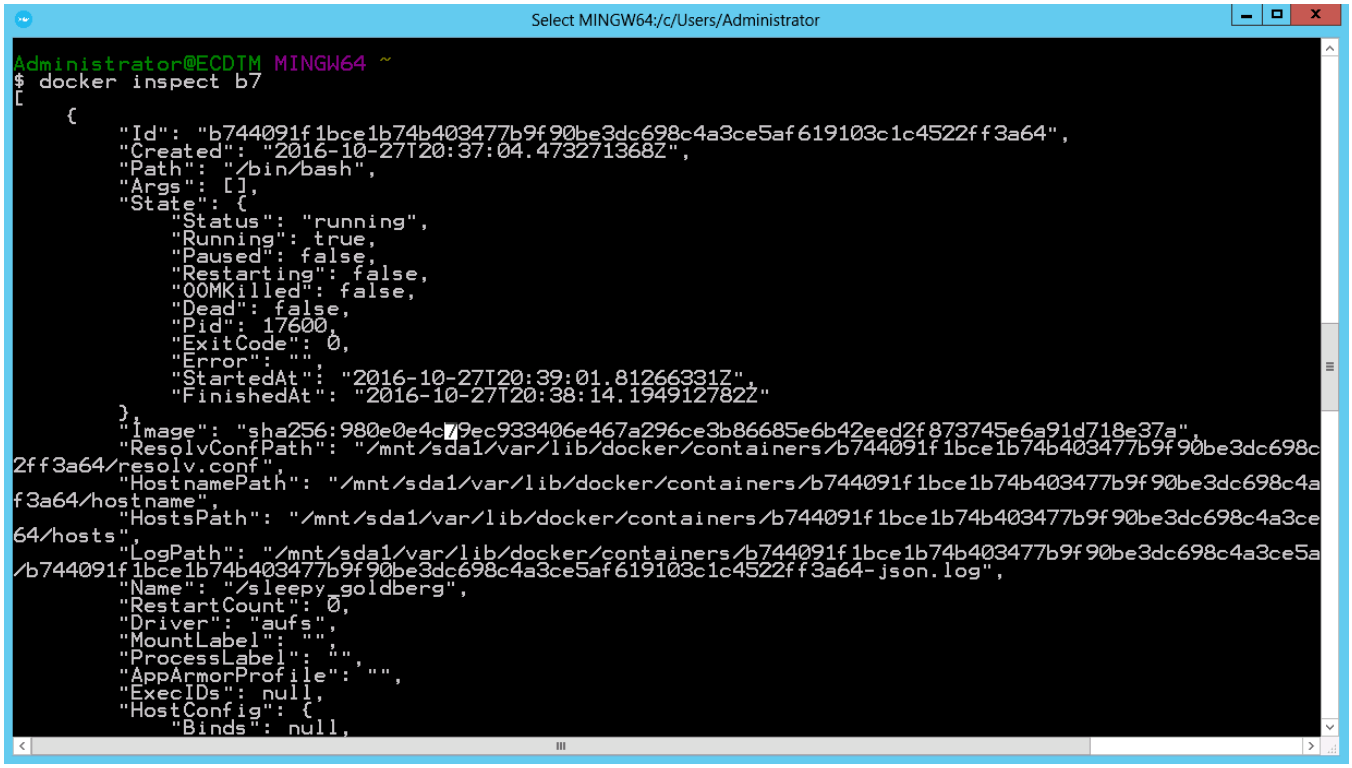

Figure 16 - Docker Inspect command

The result is a long JSON message which has all of the details about the running container; you will find the local IP of the container, networking setting, links, volumes, host information and much more. Explore the JSON file to see more information about the running container.

Accessing A Container

As we can see from the Docker '*ps*' command, the CentOS container is running. How do we access it? We will use the Docker '*exec*' command. Use the Docker '*ps*' command to get the id of the running container.

From the command prompt, enter in the following command –

$ docker exec –it "<container_id>" bash

From here, you can run a 'ls' command to explore the CentOS container; to exit the running container, just enter in the following command –

$ exit

You will see that the container is still running; how do we stop this container? Let's use the 'stop' command.

From the command prompt, enter in the following command –

$ docker stop <container_id >

If you run a Docker '*ps*' command, you can see that the container has stopped because it is no longer visible; running a Docker '*–ps –a*' command will show the container stopped.

Step 5 - Loading the Documentum Images

Now that we are familiar with some Docker commands, we are now ready to load our Documentum images. Because our Documentum Docker images are not stored in the cloud (on Docker Hub), we will import them from a local directory (we have saved them in the temp directory).

Documentum

From the Docker Quickstart Terminal, enter in the following command –

$ pwd

This will tell you your current working directory; we want to access the temp folder to interact with our Docker images.

From the command prompt, enter in the following command –

$ cd /c/temp

From the command prompt, enter in the following command –

$ ls

You will see two images and one folder; here are the Docker images for Documentum Content Server and DA. The folder called '*dctm-da*' has some code we will use in a later section of this lab. Let's jump right in and load the Content Server image into our Docker environment.

From the command prompt, enter in the following command –

$ docker load -i cs73.image

This command will import the Documentum image into Docker.

```
dmadmin@NKXCP21 MINGW64 ~
$ pwd
/c/Users/dmadmin

dmadmin@NKXCP21 MINGW64 ~
$ cd /c/temp

dmadmin@NKXCP21 MINGW64 /c/temp
$ ls
cs73.image  da73.image  dctm-da/  documentum/  webtop681.image

dmadmin@NKXCP21 MINGW64 /c/temp
$ docker load -i cs73.image
```

Figure 17 - Loading the Docker image

Note: This process might take several minutes to complete – good time for a coffee break!

Once it's done, let's check our work - let's run a Docker '*images*' command. From the command prompt, enter in the following command –

$ docker images

You should see a new image called '*cs73*' created.

Documentum Administrator

Let's import the DA image by using the same command.

$ docker load -i da73.image

This process might take a few minutes. Once these images are imported, you should see two new images.

Step 6 - Running the Documentum Images

Documenum Content Server

Let's run the Documentum Docker image; before we can do this, we need the IP address of the local server. You will need this to start the Content Server.

From the command prompt, enter in the following command –

Note: Double check the IP address, yours might be different!

$ docker run --privileged=true -p 1489:1489 -p 50000:50000 -h cs -ti cs73 bash -c "/start-db 192.168.99.100"

As you can see from the Docker '*run*' command, there are more parameters being used to start the Content Server. One of things you will notice is the –p flag which sets the ports used by the Content Server. The Content Server will load (as shown below). At this stage, the Content Server is running, and you can open up a new terminal window to get DA up and running.

```
MINGW64:/c/temp
8.99.100"
remote ip 192.168.99.100
local ip 172.17.0.3
Before
[DOCBROKER_CONFIGURATION]
secure_connect_mode=dual

[TRANSLATION]
HOST="10.43.66.13=172.17.0.2"
after
[DOCBROKER_CONFIGURATION]
secure_connect_mode=dual

[TRANSLATION]
HOST="192.168.99.100=172.17.0.3"
pg_ctl: another server might be running; trying to start server anyway
server starting
< 2016-10-27 21:43:30.001 UTC >LOG:  redirecting log output to logging collector process
< 2016-10-27 21:43:30.001 UTC >HINT:  Future log output will appear in directory "pg_log".
starting connection broker on current host: [cs]
with connection broker log: [/home/testqa/dctm/dba/log/docbroker.cs.1489.log]
connection broker pid: 37
starting Documentum server for repository: [d73]
with server log: [/home/testqa/dctm/dba/log/d73.log]
server pid: 195
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 21:43 ?        00:00:00 bash -c /start-db 192.168.99.100
root        15     1  0 21:43 ?        00:00:00 rngd -b -r /dev/urandom -o /dev/random
postgres    19     1  0 21:43 ?        00:00:00 /usr/pgsql-9.4/bin/postgres -D /var/lib/pgsql/9.4/da
postgres    24    19  0 21:43 ?        00:00:00 postgres: logger process
testqa      37     1  1 21:43 ?        00:00:00 ./dmdocbroker -port 1489 -init_file /home/testqa/dct
testqa     195     1  5 21:43 ?        00:00:00 ./documentum -docbase_name d73 -security acl -init_f
postgres   198    19  0 21:43 ?        00:00:00 postgres: checkpointer process
postgres   199    19  0 21:43 ?        00:00:00 postgres: writer process
postgres   200    19  0 21:43 ?        00:00:00 postgres: wal writer process
postgres   201    19  0 21:43 ?        00:00:00 postgres: autovacuum launcher process
postgres   202    19  0 21:43 ?        00:00:00 postgres: stats collector process
root       203     1  0 21:43 ?        00:00:00 ps -ef
Waiting for ever
```

Figure 18- Running the content server

Step 7 - Deploying DA

Now that we have the Content Server running, we need to run Documentum Administrator (DA); we loaded the image in an earlier step. Now we need to run the Docker '*run*' command and create the container.

Open a new Docker Quickstart Terminal window.

From the command prompt, enter in the following command to see our images you should still see the two Documentum Docker images –

$ docker images

From the command prompt, enter in the following command –

$ docker run -ti --privileged=true -p 8080:8080 -h da da73 bash -c "/start-da 172.17.0.2"

This will start the DA container and you will see Tomcat starting; the DA webapp will also be deployed. You might see some error messages about the global registry (DFC errors) but you can ignore them for this lab.

Open another Docker Quickstart Terminal window (you should have three now).

From the command prompt, enter in the following command –

root@Docker-Hack-0: docker ps -a

When you do a Docker '*ps*' command, you will see two containers.

We now have two containers running; let's verify the DA instance by checking from a browser. Open up a web browser and type in –

http://<server_ip>:8080

You should see the welcome page for Tomcat; modify the URL –

http://<server_ip>:8080/da

Login to DA using these credentials –

Login: testqa

Password: password

You can create cabinets and folders, just like you would in a normal Documentum environment.

Let's stop the container for DA.

From the command prompt, enter in the following command –

$ docker stop <container_id>

Step 8 - Cleaning Things Up

In the earlier labs, we created two images – one being the Docker Hello World image and the other a CentOS image. Let's delete those images and containers from the system.

Remove Containers

First, let's remove any containers; if they are running, they need to be stopped.

From the command prompt, enter in the following command to stop the hello-world and CentOS containers - you will perform this twice for both containers –

$ docker stop <container_id>

From the command prompt, enter in the following command to delete the containers from storage - you will perform this twice for both containers –

$ docker rm –f <container_id>

From the command prompt, enter in the following command –

$ docker ps –a

Verify the containers are not there any longer

Remove Images

Now we will delete the images from the system.

From the command prompt, enter in the following command –

$ docker rmi -f hello-world

From the command prompt, enter in the following command –

$ docker rmi -f centos

From the command prompt, enter in the following command to see that the images are gone –

$ docker images

```
Administrator@ECDTM MINGW64 ~
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              latest              980e0e4c79ec        7 weeks ago         196.8 MB
hello-world         latest              c54a2cc56cbb        3 months ago        1.848 kB
da73                latest              ad7ac69c353b        6 months ago        1.202 GB
cs73                latest              aca3ed9a5a93        6 months ago        6.79 GB

Administrator@ECDTM MINGW64 ~
$ _
```

Figure 19 - Docker images

Step 9 - Creating a custom Docker DA Image

In the previous steps, we imported three Documentum Docker images and ran them; in this step, we will make a custom Docker DA image by using a Dockerfile.

What is a Dockerfile?

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using the command 'docker build' users can create an automated build that executes several command-line instructions in succession.

NOTE: Make sure your DA container is stopped since we will use port 8080

From the command prompt, enter in the following command to go into our working directory –

$ cd /c/temp/dctm-da

In this folder we will create our Dockerfile; you will notice a folder called 'bundles'. In this folder in the DA.war file, and some other files we will copy into our Tomcat server once it's deployed. The DA.war file is the standard WAR file from the EMC Documentum download site.

Create a new Dockerfile

From the command prompt, enter in the following command to go into our working directory –

root@Docker-Hack-0: vi Dockerfile

This will launch vi; enter in the following code into the vi screen and save it.

```
FROM tomcat:7

VOLUME /var/log/tomcat

ENV LOGDIR /var/log/tomcat

# copy DA war

COPY bundles/da.war ${CATALINA_HOME}/webapps/


# the entrypoint (wrap catalina script)

COPY bundles/entrypoint.sh ${CATALINA_HOME}/docker-entrypoint.sh

RUN chmod a+x ${CATALINA_HOME}/docker-entrypoint.sh

# tomcat tuning for DA

ENV CUSTOM_CATALINA_OPTS="-server -Xms512m -Xmx1024m -XX:MaxPermSize=256m -
XX:+UseParallelOldGC" \

   CUSTOM_JAVA_OPTS="-Ddfc.properties.file=${CATALINA_HOME}/conf/dfc.properties -
Dcatalina.logdir=${LOGDIR} -Dlog4j.properties=${CATALINA_HOME}/conf/log4j.xml"

RUN echo "org.apache.jasper.compiler.Parser.STRICT_WHITESPACE=false" >>
${CATALINA_HOME}/conf/catalina.properties

# over ride web.xml (disable jsp pooling)

COPY bundles/web.xml ${CATALINA_HOME}/conf/

ENV REPOSITORY_NAME d73

ENV REPOSITORY_USER testqa

ENV REPOSITORY_PWD password

WORKDIR ${CATALINA_HOME}

# FIXME: can't use variable in ENTRYPOINT directive

ENTRYPOINT [ "./docker-entrypoint.sh", "catalina.sh", "run" ]

CMD ["catalina.sh", "run"]
```

Please make sure you save your file; to make sure the file is correct, cat out the file like we did in previous steps.

Create the entrypoint file

The entrypoint file provides other instructions for your container when it gets built. This is a Docker best practice. From the command prompt, enter in the following command to go into our working directory –

root@Docker-Hack-0: cd bundles

From the command prompt, enter in the following command –

root@Docker-Hack-0: vi entrypoint.sh

Enter in the following text in the entrypoint file -

```
#!/bin/sh
CATALINA_OPTS="${CUSTOM_CATALINA_OPTS} ${CATALINA_OPTS}"
JAVA_OPTS="${CUSTOM_JAVA_OPTS} ${JAVA_OPTS}"
export CATALINA_OPTS JAVA_OPTS
# configure dfc
DFC_DATADIR=${CATALINA_HOME}/temp/dfc
[ -d ${DFC_DATADIR} ] || mkdir -p ${DFC_DATADIR}
cat << __EOF__ > ${CATALINA_HOME}/conf/dfc.properties
dfc.name=da
dfc.data.dir=${DFC_DATADIR}
dfc.tokenstorage.enable=false
dfc.docbroker.host[0]=<Your Host IP Address>
dfc.docbroker.port[0]=1489
dfc.session.secure_connect_default=try_native_first
dfc.session.allow_trusted_login = false
__EOF__
echo "DFC Config file:"
cat conf/dfc.properties
echo "Using CATALINA_OPTS:   ${CATALINA_OPTS}"
echo "Using JAVA_OPTS:       ${JAVA_OPTS}"
exec "$@"
```

Save the file once you are done; to check your work, you can cat out the file as we did in previous labs.

Let's CD to the folder with the Dockerfile.

From the command prompt, enter in the following command –

root@Docker-Hack-0: cd ..

From the command prompt, enter in the following command to build this image –

root@Docker-Hack-0: docker build -t "myda" .

```
dmadmin@Docker-Hack-0:~/Downloads/dctm-da$ docker build -t "myda" .
Sending build context to Docker daemon 263.6 MB
Step 1 : FROM tomcat:7
7: Pulling from library/tomcat
efd26ecc9548: Pull complete
a3ed95caeb02: Pull complete
d1784d73276e: Pull complete
52a884c93bb2: Pull complete
c35f0a4a3a31: Pull complete
d01b68d70ee0: Pull complete
73faba584c67: Pull complete
a24bef42bc18: Pull complete
85d3cd970604: Pull complete
```

Figure 20 - Building a Dockerfile

Once the image is done downloading and loading, from the command prompt, enter in the following command –

root@Docker-Hack-0: docker images

You should see a new image called 'myda'

From the command prompt, enter in the following command to run this new image –

root@Docker-Hack-0: docker run -d -it -p 8080:8080 --link <cs_container_name>:cs73 myda

You will notice that the logs for Tomcat did not show – why? Because we passed in a new parameter (the –d flag) which runs the container in the background. You will also notice another new parameter called 'link'. This is a way to link containers that need to communicate with each other. At this point you can check the URL and hit the new container!