MOMENTUM 2016

HACK
-A-
THON

INFOARCHIVE @ DOCKER

Table of Contents

Overview

Welcome to the InfoArchive@ Docker Lab

This DellEMC lab will walk you through using InfoArchive 4.1 and Docker; this lab includes an overview of Docker, building images, and creating containers.

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 Lab

In this lab, we will cover the following concepts:-

1. Running Docker on Windows
2. Docker ToolBox
3. Installing Docker
4. Verifying Docker is installed
5. Pulling Docker images
6. Working with Docker Images
7. Working with Docker Containers
8. Mapping local volumes to Docker Containers
9. Creating and using the InfoArchive Docker images

Overview

In this lab, we will be using InfoArchive 4.1, and the latest release of Docker Toolbox. We will also be using a centos 7 Docker image as a base when building the InfoArchive image. Docker Toolbox is an installer utility to quickly setup a Docker environment on your Windows or Mac Computer.

Assumptions

The labs will assume that you have some InfoArchive experience and are familiar with InfoArchive concepts and architecture; some Docker and Linux skills are helpful but not required. We will assume you are new to Docker and Linux.

Step 1 - Accessing Your Windows Image

For the lab, we are using a Windows image (Windows Server 2012) hosted on a cloud environment. You will access the Windows image via Browser. To start please open the browser of your choice and enter the following url:-

https://mmtm.getecdcontent.com/emcrdp/#/

NOTE: The keyboard on the windows image is English (Untied States)

NOTE: Your lab instructor will provide you with the IPs-address, username and password need to connect to the image.

Step 2 – Getting Familiar with Docker Toolbox

NOTE: Docker Toolbox is already installed on your Windows Image. The software is available for Windows and Mac at https://docker.com/products/docker-toolbox

Docker Toolbox includes the following components bundled together for simple setup of your Docker environment:-

1. Docker
2. Docker Machine: A tool that lets you install Docker engine on virtual host and manage the hosts with docker-machine command.[1]
3. Docker Compose: A tool used for defining and running multi-container Docker applications.[2]
    a. Dockerfile is used to define the application environment

[1] https://docs.docker.com/machine/overview/

[2] https://docs.docker.com/compose/overview/

   b. docker-compose.yml defines the services inside your app, in order for them to run in an isolated manner.
   c. docker-compose up will start and run the entire app.
4. Kitematic: is a GUI that simplifies the management and provision of Docker containers.[3]
5. Boot2Docker: a lightweight Linux distribution made specifically to run Docker containers. It runs completely from RAM, is a small ~38MB download and boots in ~5s (YMMV).[4]

Confirm Docker Installation

Before we begin let's verify the Docker installation on the image. Launch the Docker quick start terminal located on your desktop.

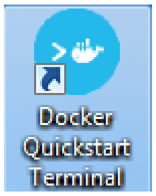
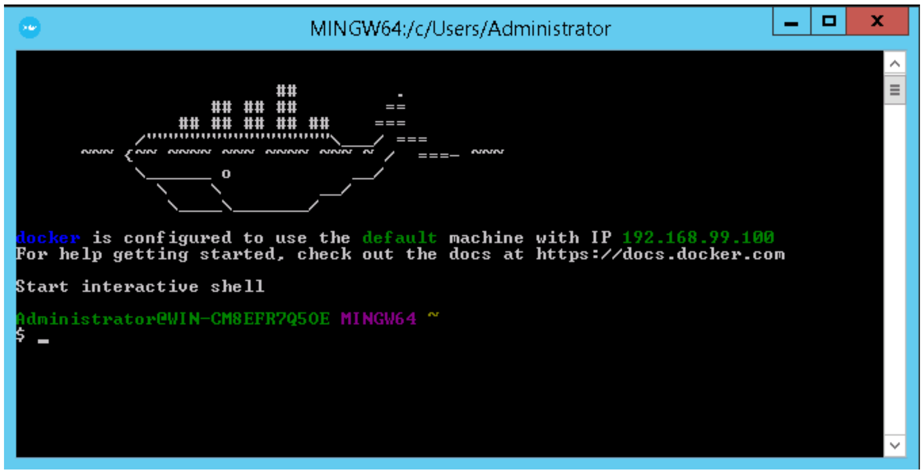


Figure 1 Docker QuickStart Terminal Icon

The terminal will start the boot2docker image running on VirtualBox, and bootstrap the environment to be ready for all Docker related activities.



```
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \______ o           __/
             \    \         __/
              \____\_______/

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@WIN-CM8EFR7Q50E MINGW64 ~
$ _
```

Figure 2 Docker Quick Start Terminal Output

[3] https://docs.docker.com/kitematic/userguide/

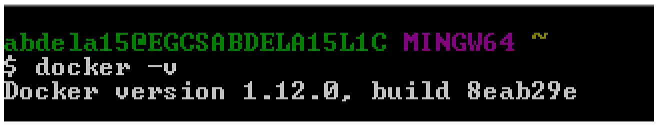
[4] https://github.com/boot2docker/boot2docker

Now since we have the Docker client running, we can run some Docker commands to check out the environment.

Identify Docker Version

In the terminal, enter the following command to see the Docker version.

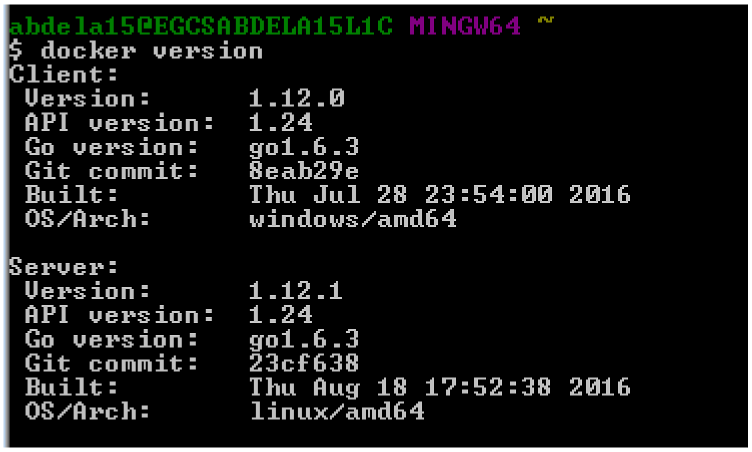
docker -v



```
abdela15@EGCSABDELA15L1C MINGW64 ~
$ docker -v
Docker version 1.12.0, build 8eab29e
```

Figure 3 docker -v

docker version



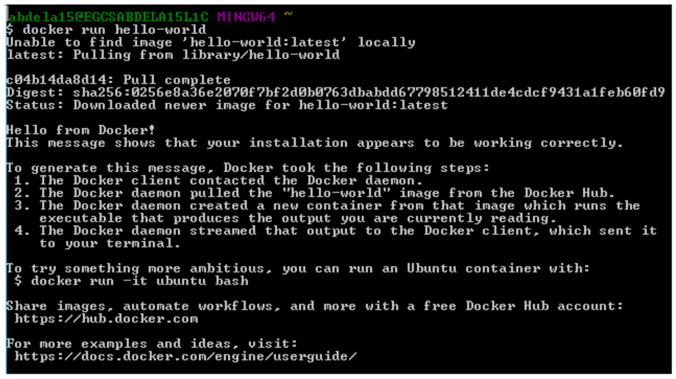
```
abdela15@EGCSABDELA15L1C MINGW64 ~
$ docker version
Client:
 Version:      1.12.0
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   8eab29e
 Built:        Thu Jul 28 23:54:00 2016
 OS/Arch:      windows/amd64

Server:
 Version:      1.12.1
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   23cf638
 Built:        Thu Aug 18 17:52:38 2016
 OS/Arch:      linux/amd64
```

Figure 4 docker version

Run a Docker Image

In the terminal, enter the following command:

docker run hello-world



```
abdela15@EGCSABDELA15L1C 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/
```

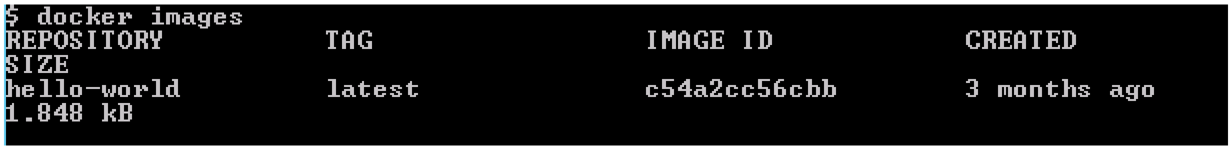
Figure 5 docker run hello-world

The docker run command pulled the latest version of the hello-world image from the Docker Hub, which is a public docker registry. The run command also launched the image and we had ourselves a running container.

Listing docker images

In the terminal, enter the following command:

docker images



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

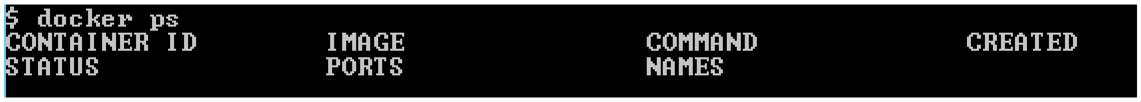
Figure 6 docker images

The command lists the currently available images; here we only have the hello-world image we just downloaded.

View Running Processes

In the terminal, enter the following command:

docker ps



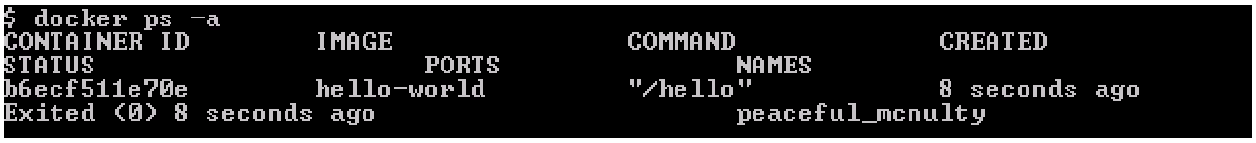
```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

Figure 7 docker ps

The docker ps command only shows container which are currently running, that is why the hello-world container we just launched is not showing.

To see all containers, use the following command "docker ps –a"

docker ps -a



```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                     PORTS               NAMES
b6ecf511e70e        hello-world         "/hello"            8 seconds ago
Exited (0) 8 seconds ago                       peaceful_mcnulty
```

Figure 8 docker ps –a

The name of the container 'peaceful_mcnulty' is randomly assigned, thus you may find a different name when running the command.

Step 3 – Building the Centos base image

In previous Labs, we used to load the product image directly into docker. In this lab we will follow some steps to create the image ourselves.

The first step to creating the InfoArchive image is to create an image with Java installed. Let us start by navigating to the InfoArchive Lab folder inside the Windows Image.

From the Docker quick start terminal enter the following command

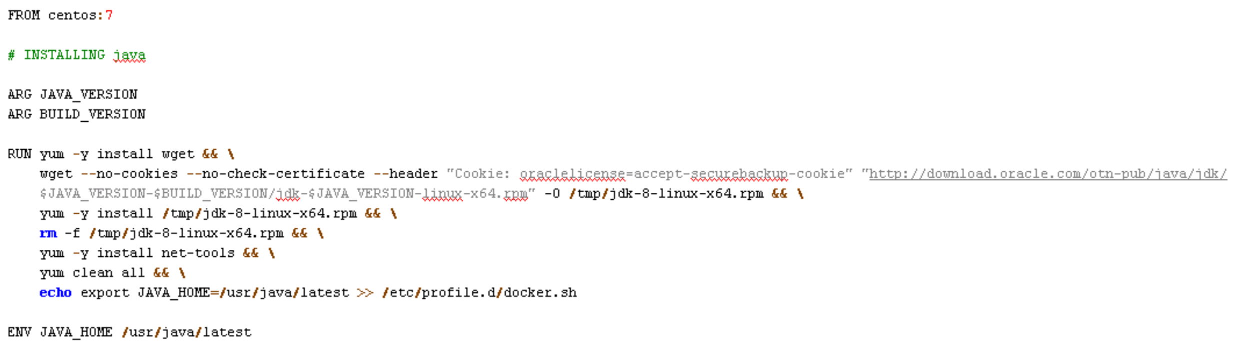
cd /c/Users/Administrator/InfoArchive/docker_files

```
$ cd /C/Users/Administrator/InfoArchive/docker_files
```

This folder contains all the required material for building and running the docker images.

Inside the docker_files directory, there are two directories that will be used to create the centos image and the InfoArchive image.

Inside the centos-java folder is a Dockerfile which describes to docker how to create the docker image.



```
FROM centos:7

# INSTALLING java

ARG JAVA_VERSION
ARG BUILD_VERSION

RUN yum -y install wget && \
    wget --no-cookies --no-check-certificate --header "Cookie: oraclelicense=accept-securebackup-cookie" "http://download.oracle.com/otn-pub/java/jdk/
    $JAVA_VERSION-$BUILD_VERSION/jdk-$JAVA_VERSION-linux-x64.rpm" -O /tmp/jdk-8-linux-x64.rpm && \
    yum -y install /tmp/jdk-8-linux-x64.rpm && \
    rm -f /tmp/jdk-8-linux-x64.rpm && \
    yum -y install net-tools && \
    yum clean all && \
    echo export JAVA_HOME=/usr/java/latest >> /etc/profile.d/docker.sh

ENV JAVA_HOME /usr/java/latest
```

Figure 9 centos-java Dockerfile

To build the image type the following command in to the command line

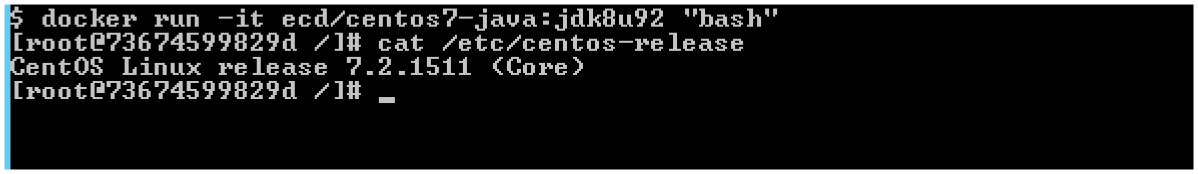
docker build --build-arg JAVA_VERSION=8u92 --build-arg BUILD_VERSION=b14 -t ecd/centos7-java:jdk8u92 centos-java

To test our machine, let us create a new container out of it.

Type the following command in to the command line:-

docker run -it ecd/centos7-java:jdk8u92 bash

Afterward type the following command to identify the os version
cat /etc/centos-release



```
$ docker run -it ecd/centos7-java:jdk8u92 "bash"
[root@73674599829d /]# cat /etc/centos-release
CentOS Linux release 7.2.1511 (Core)
[root@73674599829d /]# _
```

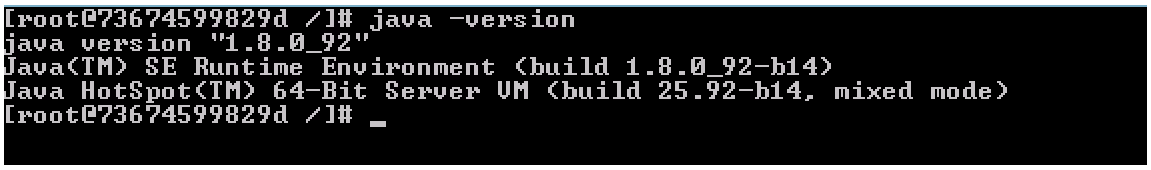
Figure 10 docker run -it ecd/centos7-java:jdk8u92 bash

- 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.

Check if Java has been installed correctly by typing the following command

java –version

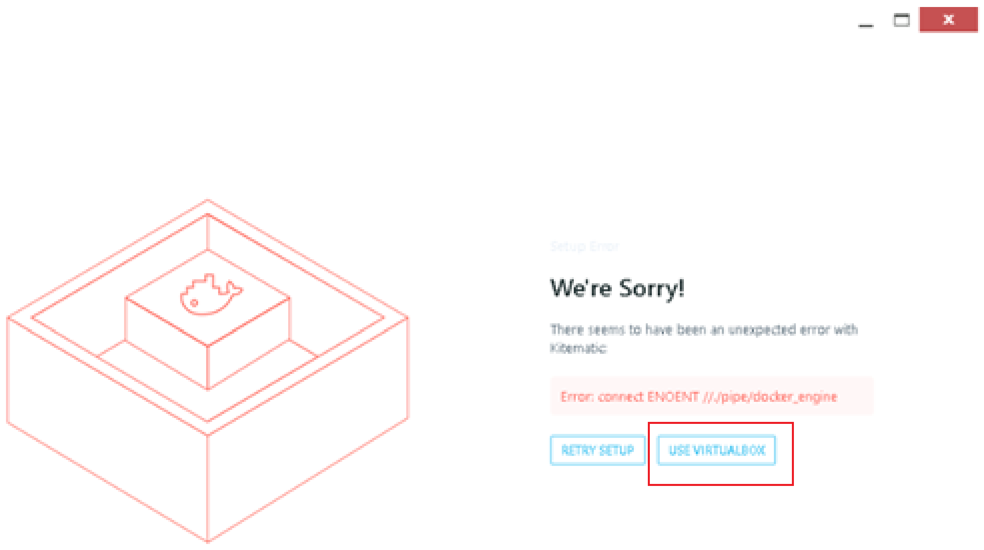


```
[root@73674599829d /]# java -version
java version "1.8.0_92"
Java(TM) SE Runtime Environment (build 1.8.0_92-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.92-b14, mixed mode)
[root@73674599829d /]# _
```

Figure 11 java –version

Launch the Kitematic application by double clicking the Kitematic (Alpha) icon from the Desktop.

If the following screen comes up when launching Kitematic, click on USE VIRTUALBOX, and skip the login page.



Setup Error

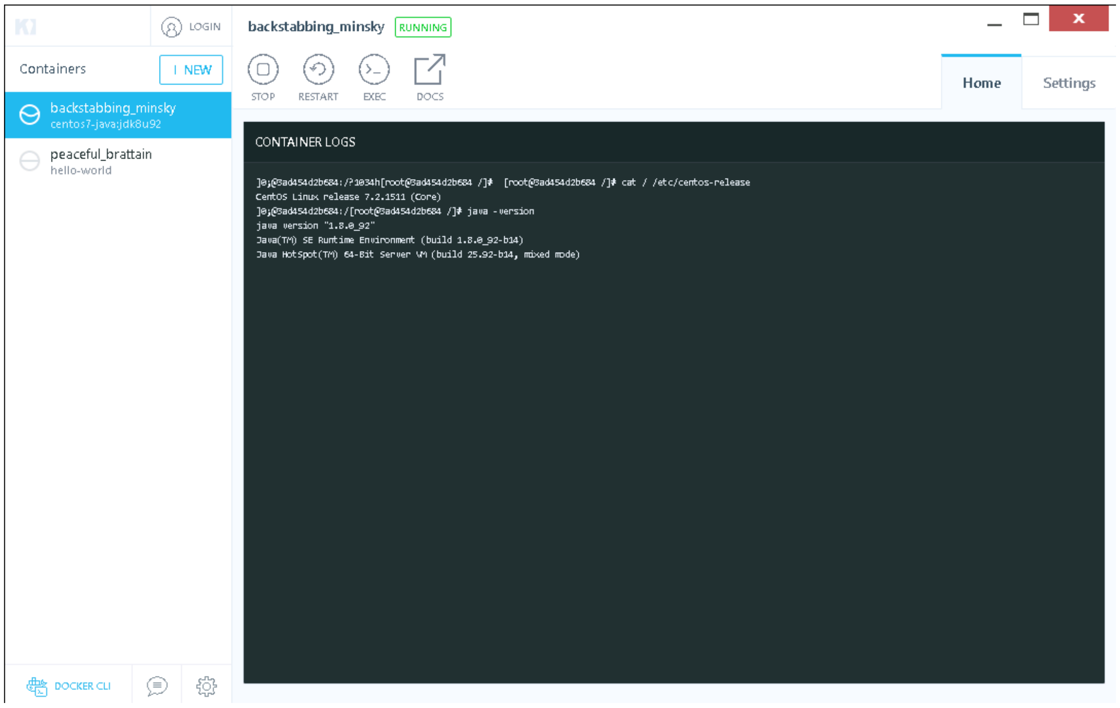
We're Sorry!

There seems to have been an unexpected error with Kitematic

Error: connect ENOENT //./pipe/docker_engine

In the Kitematic screen you will find the two containers we ran (hello-world) & (centos7-java:jdk8u92).

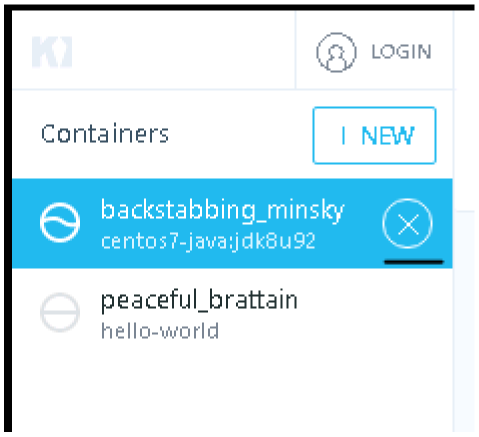
Docker creates random name to image if they were not named (i.e. thirsty_wilson). To assign a name to a container pass in the --name {{name}} argument when running the container.



```
]0;@3ad454d2b684:/?1034h[root@3ad454d2b684 /]#  [root@3ad454d2b684 /]# cat / /etc/centos-release
CentOS Linux release 7.2.1511 (Core)
]0;@3ad454d2b684:/[root@3ad454d2b684 /]# java -version
java version "1.8.0_92"
Java(TM) SE Runtime Environment (build 1.8.0_92-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.92-b14, mixed mode)
```

Figure 12 Kitematic Screen

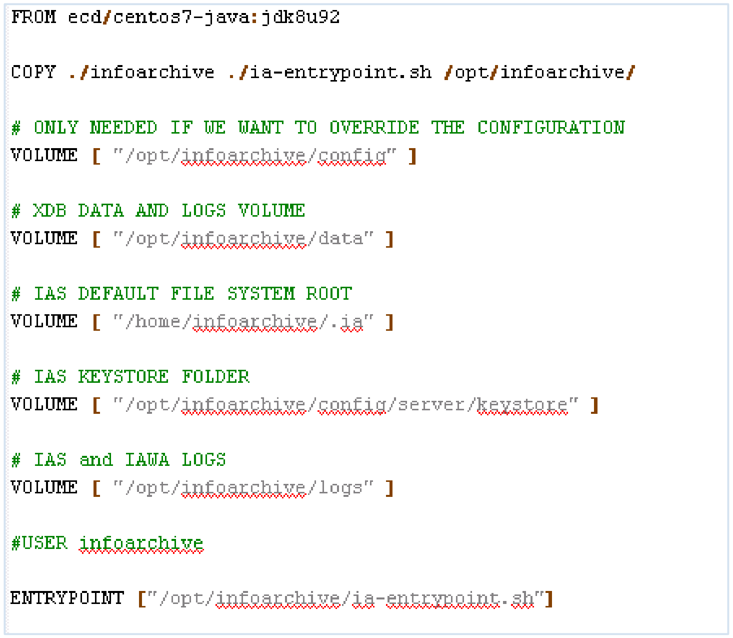
NOTE: To delete a container from the Kitematic screen, hover over the container name on the side bar and click on the X button that appears.



Step 4 – Building the Info Archive Image

After building the centos base image containing java we can now being building the InfoArchive 4.1 Image.

Inside the ia41 folder (**C:\Users\Administrator\InfoArchive\docker_files\ia41**) you can find the InfoArchive Binaries, and a Dockerfile describing the image.



```
FROM ecd/centos7-java:jdk8u92

COPY ./infoarchive ./ia-entrypoint.sh /opt/infoarchive/

# ONLY NEEDED IF WE WANT TO OVERRIDE THE CONFIGURATION
VOLUME [ "/opt/infoarchive/config" ]

# XDB DATA AND LOGS VOLUME
VOLUME [ "/opt/infoarchive/data" ]

# IAS DEFAULT FILE SYSTEM ROOT
VOLUME [ "/home/infoarchive/.ia" ]

# IAS KEYSTORE FOLDER
VOLUME [ "/opt/infoarchive/config/server/keystore" ]

# IAS and IAWA LOGS
VOLUME [ "/opt/infoarchive/logs" ]

#USER infoarchive

ENTRYPOINT ["/opt/infoarchive/ia-entrypoint.sh"]
```

The Dockerfile contains the tasks needed to build the InfoArchive image (copying the binaries, defining volumes and specifying an Entry point for the docker container, which will be executed once the container is created.

Firstly, let's enumerate the different volumes that are configured in this docker image.

- **/opt/infoarchive/data :** Where the xDB database files are written to.
- **/home/infoarchive/.ia :** Where the default File System Root will be created.
- **/opt/infoarchive/config/server/keystore :** Where the JCE keystore file is written to.
- **/opt/infoarchive/logs :** Where the IAS and IAWA logs are written to

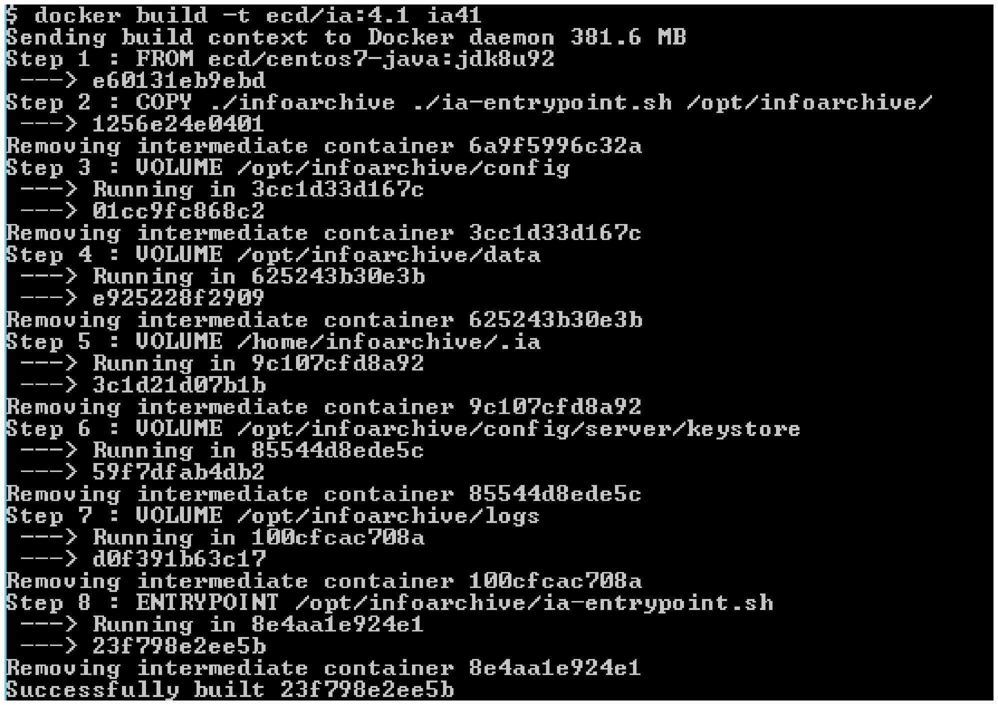
In this lab, we are creating one Image for all InfoArchive services and each container will act as an xdb_node, InfoArchive Server or InfoArchive WebApp depending on flags passed to the container.

Inside the Command line, type the following command to exit the **ecd/centos7-java:jdk8u92** container we created in Step 3

exit

Inside the Command line, type the following command

docker build -t ecd/ia:4.1 ia41

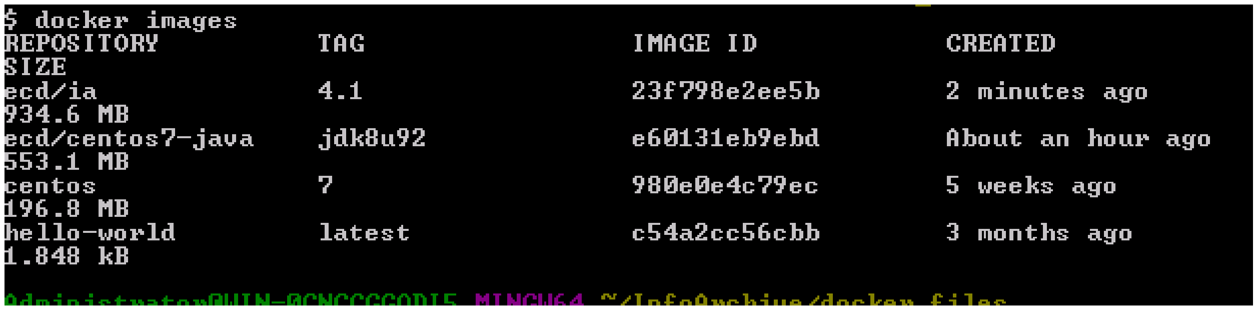


```
$ docker build -t ecd/ia:4.1 ia41
Sending build context to Docker daemon 381.6 MB
Step 1 : FROM ecd/centos7-java:jdk8u92
 ---> e60131eb9ebd
Step 2 : COPY ./infoarchive ./ia-entrypoint.sh /opt/infoarchive/
 ---> 1256e24e0401
Removing intermediate container 6a9f5996c32a
Step 3 : VOLUME /opt/infoarchive/config
 ---> Running in 3cc1d33d167c
 ---> 01cc9fc868c2
Removing intermediate container 3cc1d33d167c
Step 4 : VOLUME /opt/infoarchive/data
 ---> Running in 625243b30e3b
 ---> e925228f2909
Removing intermediate container 625243b30e3b
Step 5 : VOLUME /home/infoarchive/.ia
 ---> Running in 9c107cfd8a92
 ---> 3c1d21d07b1b
Removing intermediate container 9c107cfd8a92
Step 6 : VOLUME /opt/infoarchive/config/server/keystore
 ---> Running in 85544d8ede5c
 ---> 59f7dfab4db2
Removing intermediate container 85544d8ede5c
Step 7 : VOLUME /opt/infoarchive/logs
 ---> Running in 100cfcac708a
 ---> d0f391b63c17
Removing intermediate container 100cfcac708a
Step 8 : ENTRYPOINT /opt/infoarchive/ia-entrypoint.sh
 ---> Running in 8e4aa1e924e1
 ---> 23f798e2ee5b
Removing intermediate container 8e4aa1e924e1
Successfully built 23f798e2ee5b
```

Figure 13 docker build -t ecd/ia:4.1 ia41

You have now created the InfoArchive docker image. Try checking out the list of available docker images using the "docker images" command:-

docker images



```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ecd/ia              4.1                 23f798e2ee5b        2 minutes ago
934.6 MB
ecd/centos7-java    jdk8u92             e60131eb9ebd        About an hour ago
553.1 MB
centos              7                   980e0e4c79ec        5 weeks ago
196.8 MB
hello-world         latest              c54a2cc56cbb        3 months ago
1.848 kB

Administrator@WIN-0CNCCGODI5 MINGW64 ~/InfoArchive/docker-files
```

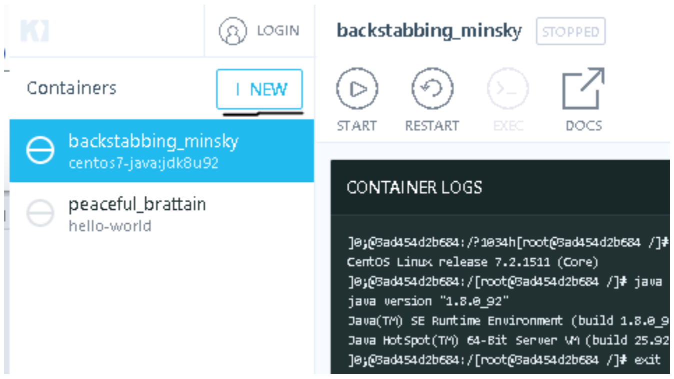
Figure 14 docker images

The docker images command now list all the image we created (ecd/centos7-java, ecd/ia), and he centos:7 image which was used to create the ecd/centos7-java image.

The same information can be viewed from the Kitematic images tab.

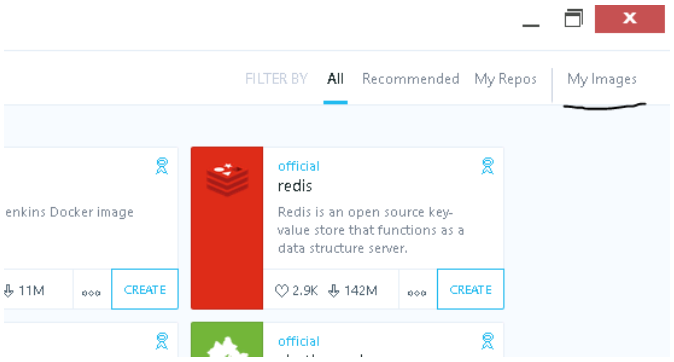
To view the images tab

- Click on | New



```
]0;@3ad454d2b684:/?1034h[root@3ad454d2b684 /]#
CentOS Linux release 7.2.1511 (Core)
]0;@3ad454d2b684:/[root@3ad454d2b684 /]# java
java version "1.8.0_92"
Java(TM) SE Runtime Environment (build 1.8.0_9
Java HotSpot(TM) 64-Bit Server VM (build 25.92
]0;@3ad454d2b684:/[root@3ad454d2b684 /]# exit
```

- Click on My Images on the Top Right side of the screen.



Inside the Kitematic screen; you will find all the images we created so far.

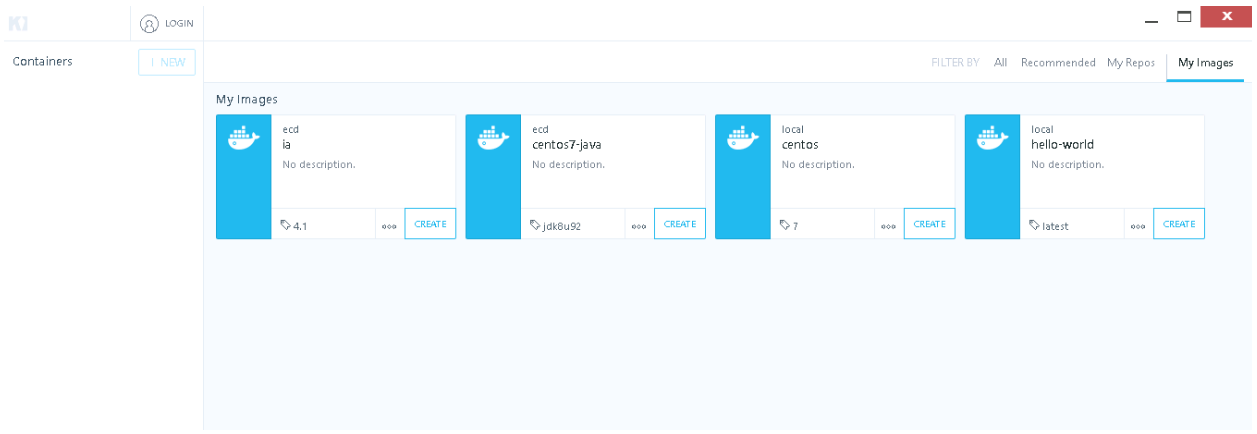
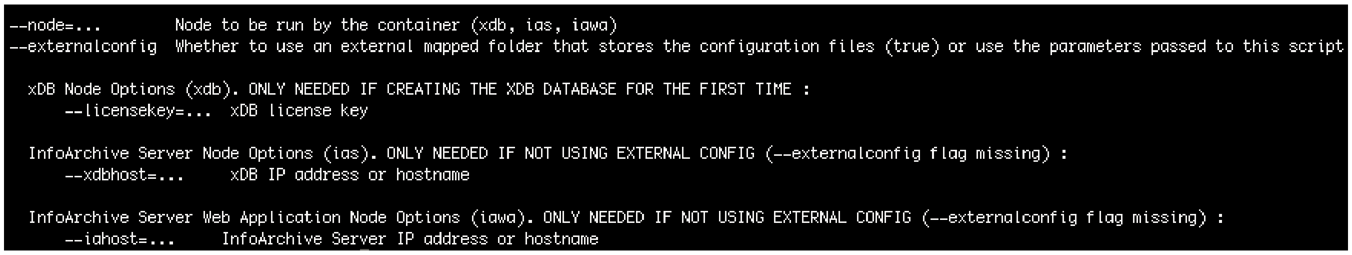


Figure 15 Kitematic's display of docker images

Step 5 – Running the InfoArchive Containers

In the previous step, we built the InfoArchive image and set a file (entrypoint.sh) as an entry point to the application. The file launches the specific InfoArchive service according to the given command line argument.

The following figure describes the entrypoint script options



```
--node=...        Node to be run by the container (xdb, ias, iawa)
--externalconfig  Whether to use an external mapped folder that stores the configuration files (true) or use the parameters passed to this script

  xDB Node Options (xdb). ONLY NEEDED IF CREATING THE XDB DATABASE FOR THE FIRST TIME :
      --licensekey=...  xDB license key

  InfoArchive Server Node Options (ias). ONLY NEEDED IF NOT USING EXTERNAL CONFIG (--externalconfig flag missing) :
      --xdbhost=...     xDB IP address or hostname

  InfoArchive Server Web Application Node Options (iawa). ONLY NEEDED IF NOT USING EXTERNAL CONFIG (--externalconfig flag missing) :
      --iahost=...     InfoArchive Server IP address or hostname
```

Figure 16 InfoArchive entrypoint script options

We will run three instances for the following services:-

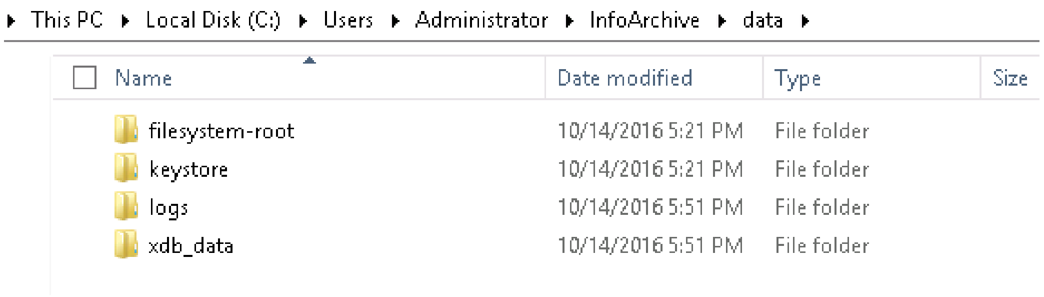
- IA XDB
- IA Server
- IA Web App

Let us first create some directories that will be mapped to the InfoArchive containers for easier access to logs.

Navigate to C:\Users\Administrator\InfoArchive\data\

Create the following folders:-

1. xdb_data
2. logs
3. filesystem-root
4. keystore



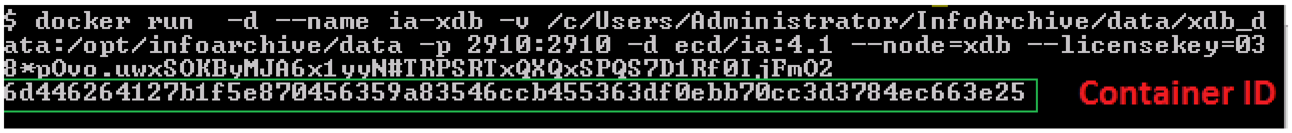
▸ This PC ▸ Local Disk (C:) ▸ Users ▸ Administrator ▸ InfoArchive ▸ data ▸

Name	Date modified	Type	Size
filesystem-root	10/14/2016 5:21 PM	File folder	
keystore	10/14/2016 5:21 PM	File folder	
logs	10/14/2016 5:51 PM	File folder	
xdb_data	10/14/2016 5:51 PM	File folder	

IA xDB Container

To start the InfoArchive xDB container, type the following into the command line:

docker run -d --name ia-xdb -v /c/Users/Administrator/InfoArchive/data/xdb_data:/opt/infoarchive/data -p 2910:2910 -d ecd/ia:4.1 --node=xdb --licensekey=038*pOvo.uwxSOKByMJA6x1yyN#TRPSRTxQXQxSPQS7D1Rf0IjFmO2



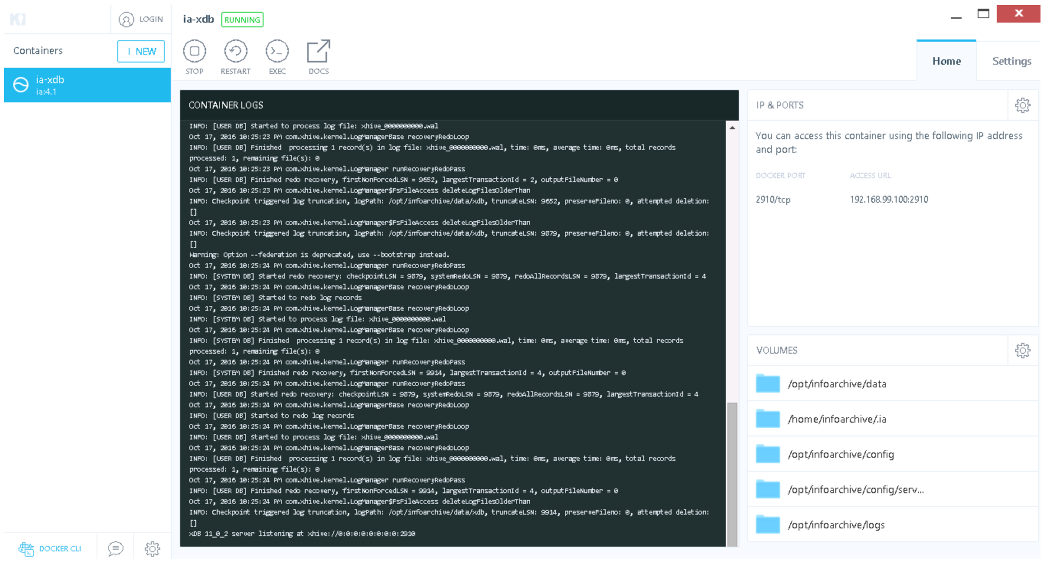
```
$ docker run  -d --name ia-xdb -v /c/Users/Administrator/InfoArchive/data/xdb_d
ata:/opt/infoarchive/data -p 2910:2910 -d ecd/ia:4.1 --node=xdb --licensekey=03
8*pOvo.uwxSOKByMJA6x1yyN#TRPSRTxQXQxSPQS7D1Rf0IjFmO2
6d446264127b1f5e870456359a83546ccb455363df0ebb70cc3d3784ec663e25
```

Let us go through the different parameters inside the previous command

Parameter	Definition
--name ia-xdb	The parameter specifies a name for container that is created and run by this command in this case (ia-xdb). This makes it easier for us to locate this container instance
-v /c/users..:/opt/infoarchive/data	The –v parameter informs Docker to map the following volume (/opt/infoarchive/data) to the respective folder on our local host.
-p 2910:2910	Map the container port to the port in our Image.
-d ecd/ia:4.1	Name of the image used for creating this container
--node=xdb	A parameter passed to the entrypoint.sh file, which launches the respective node in this container.
--licensekey=....	The license key to be used while running the embedded xDB database.

Let's navigate to the Kitematic interface you should now see one container running named ("ia-xdb"). Once we click on the container name from the sidebar, we see the container's stdout and other information such the Docker port assigned and the list of volumes.

A more thorough list of settings is available under the Setting tab on the right hand side of the screen.



```
INFO: [USER DB] Started to process log file: xhive_000000000.wal
Oct 17, 2016 10:25:23 PM com.xhive.kernel.LogManagerBase recoveryRedoLoop
INFO: [USER DB] Finished  processing 1 record(s) in log file: xhive_000000000.wal, time: 0ms, average time: 0ms, total records
processed: 1, remaining file(s): 0
Oct 17, 2016 10:25:23 PM com.xhive.kernel.LogManager runRecoveryRedoPass
INFO: [USER DB] Finished redo recovery, firstNonForcedLSN = 9652, largestTransactionId = 2, outputFileNumber = 0
Oct 17, 2016 10:25:23 PM com.xhive.kernel.LogManager$FsFileAccess deleteLogFilesOlderThan
INFO: Checkpoint triggered log truncation, logPath: /opt/infoarchive/data/xdb, truncateLSN: 9652, preserveFileno: 0, attempted deletion:
[]
Oct 17, 2016 10:25:23 PM com.xhive.kernel.LogManager$FsFileAccess deleteLogFilesOlderThan
INFO: Checkpoint triggered log truncation, logPath: /opt/infoarchive/data/xdb, truncateLSN: 9879, preserveFileno: 0, attempted deletion:
[]
Warning: Option --federation is deprecated, use --bootstrap instead.
Oct 17, 2016 10:25:24 PM com.xhive.kernel.LogManager runRecoveryRedoPass
INFO: [SYSTEM DB] Started redo recovery: checkpointLSN = 9879, systemRedoLSN = 9879, redoAllRecordsLSN = 9879, largestTransactionId = 4
Oct 17, 2016 10:25:24 PM com.xhive.kernel.LogManagerBase recoveryRedoLoop
INFO: [SYSTEM DB] Started to redo log records
Oct 17, 2016 10:25:24 PM com.xhive.kernel.LogManagerBase recoveryRedoLoop
INFO: [SYSTEM DB] Started to process log file: xhive_000000000.wal
Oct 17, 2016 10:25:24 PM com.xhive.kernel.LogManagerBase recoveryRedoLoop
INFO: [SYSTEM DB] Finished  processing 1 record(s) in log file: xhive_000000000.wal, time: 0ms, average time: 0ms, total records
processed: 1, remaining file(s): 0
Oct 17, 2016 10:25:24 PM com.xhive.kernel.LogManager runRecoveryRedoPass
INFO: [SYSTEM DB] Finished redo recovery, firstNonForcedLSN = 9914, largestTransactionId = 4, outputFileNumber = 0
Oct 17, 2016 10:25:24 PM com.xhive.kernel.LogManager runRecoveryRedoPass
INFO: [USER DB] Started redo recovery: checkpointLSN = 9879, systemRedoLSN = 9879, redoAllRecordsLSN = 9879, largestTransactionId = 4
Oct 17, 2016 10:25:24 PM com.xhive.kernel.LogManagerBase recoveryRedoLoop
INFO: [USER DB] Started to redo log records
Oct 17, 2016 10:25:24 PM com.xhive.kernel.LogManagerBase recoveryRedoLoop
INFO: [USER DB] Started to process log file: xhive_000000000.wal
Oct 17, 2016 10:25:24 PM com.xhive.kernel.LogManagerBase recoveryRedoLoop
INFO: [USER DB] Finished  processing 1 record(s) in log file: xhive_000000000.wal, time: 0ms, average time: 0ms, total records
processed: 1, remaining file(s): 0
Oct 17, 2016 10:25:24 PM com.xhive.kernel.LogManager runRecoveryRedoPass
INFO: [USER DB] Finished redo recovery, firstNonForcedLSN = 9914, largestTransactionId = 4, outputFileNumber = 0
Oct 17, 2016 10:25:24 PM com.xhive.kernel.LogManager$FsFileAccess deleteLogFilesOlderThan
INFO: Checkpoint triggered log truncation, logPath: /opt/infoarchive/data/xdb, truncateLSN: 9914, preserveFileno: 0, attempted deletion:
[]
xDB 11_0_2 server listening at xhive://0:0:0:0:0:0:0:0:2910
```

You can access this container using the following IP address and port:

DOCKER PORT	ACCESS URL
2910/tcp	192.168.99.100:2910

VOLUMES:
- /opt/infoarchive/data
- /home/infoarchive/ia
- /opt/infoarchive/config
- /opt/infoarchive/config/serv...
- /opt/infoarchive/logs

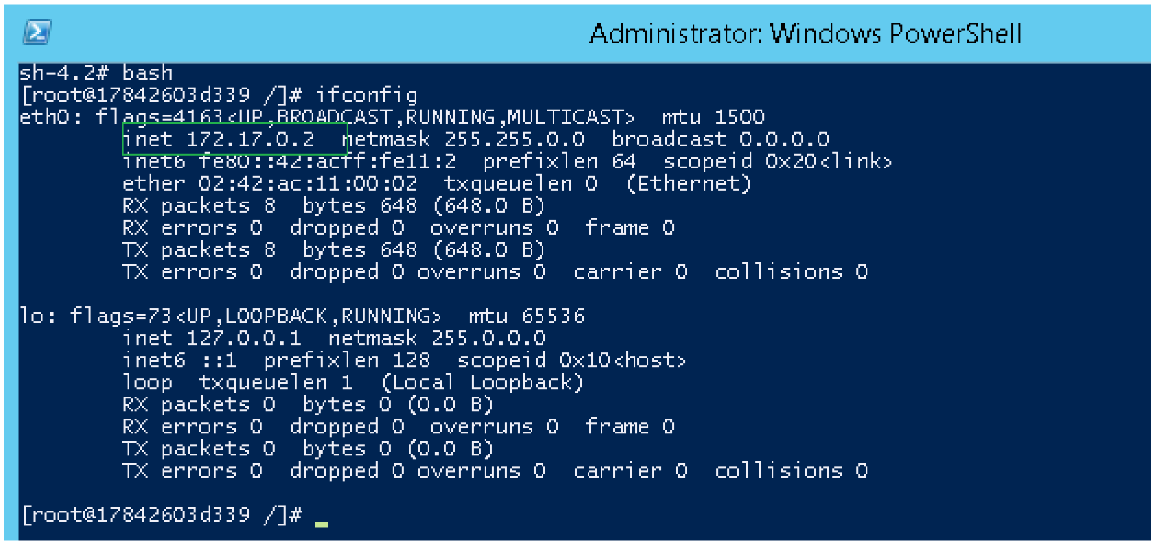
Figure 17 Kitematic view of the ia-xdb container

In order to launch the IA Server we require the ip address of the xdb node that we just created.

Click on the Exec Button in the Kitematic screen. A windows power shell with launch with an ssh client inside the docker container. We could bash inside the machine and explore the IA xDB container we just created.

To get the ip address of the container Type the following command inside the Windows power shell:-

Ifconfig



```
sh-4.2# bash
[root@17842603d339 /]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 0.0.0.0
        inet6 fe80::42:acff:fe11:2  prefixlen 64  scopeid 0x20<link>
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 8  bytes 648 (648.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 648 (648.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@17842603d339 /]# _
```

Figure 18 Ipconfig

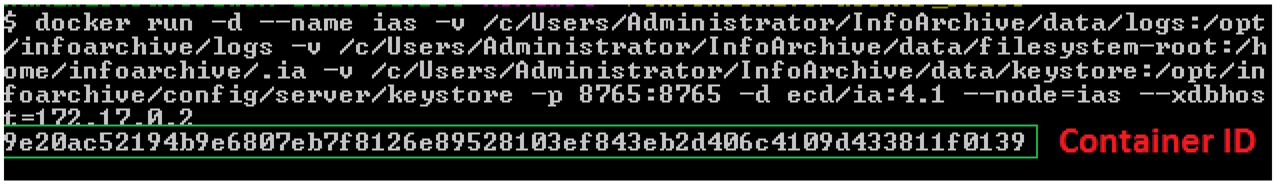
Now that we have the ip address (**172.17.0.2**) for this container, let us run the InfoArchive Server.

IA Server Container

Close the Windows PowerShell and return to the docker command line.

To start the InfoArchive Server container, type the following into the command line:

docker run -d --name ias -v /c/Users/Administrator/InfoArchive/data/logs:/opt/infoarchive/logs -v /c/Users/Administrator/InfoArchive/data/filesystem-root:/home/infoarchive/.ia -v /c/Users/Administrator/InfoArchive/data/keystore:/opt/infoarchive/config/server/keystore -p 8765:8765 -d ecd/ia:4.1 --node=ias --xdbhost=172.17.0.2

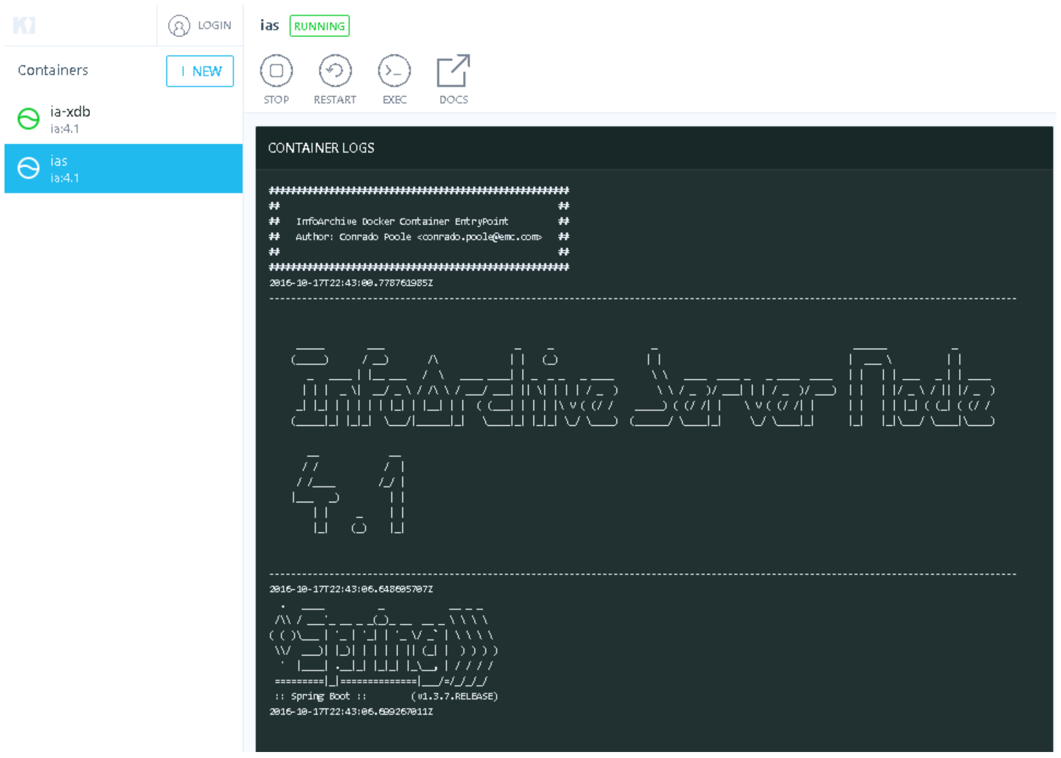


```
$ docker run -d --name ias -v /c/Users/Administrator/InfoArchive/data/logs:/opt
/infoarchive/logs -v /c/Users/Administrator/InfoArchive/data/filesystem-root:/h
ome/infoarchive/.ia -v /c/Users/Administrator/InfoArchive/data/keystore:/opt/in
foarchive/config/server/keystore -p 8765:8765 -d ecd/ia:4.1 --node=ias --xdbhos
t=172.17.0.2
9e20ac52194b9e6807eb7f8126e89528103ef843eb2d406c4109d433811f0139
```

Parameter	Definition
--name ias	The parameter specifies a name for container that is created and run by this command in this case (ias). This makes it easier for us to locate this container instance
-v /c/users..:/opt/infoarchive/logs	The –v parameter informs Docker to map the following volume (/opt/infoarchive/logs) to the respective folder on our local host.
-p 8765:8765	Map the container port to the port in our Image.
-d ecd/ia:4.1	Name of the image used for creating this container
--node=ias	A parameter passed to the entrypoint.sh file, which launches the respective node (IAS) in this container.
--xdbhost=172.17.0.2	Specify the xdb host ipaddress for the InfoArchive Server to use.

The docker run command follows the same structure as that of the InfoArchive xDB node. The only difference is that we specify a different node (--node=ias), which identifies this container as an InfoArchive Server Container for the entrypoint file. Additionally, we specify the location of the xdbhost (--xdbhost=172.17.0.2), notice that we entered the ip address for the xdb container we got in the previous step.

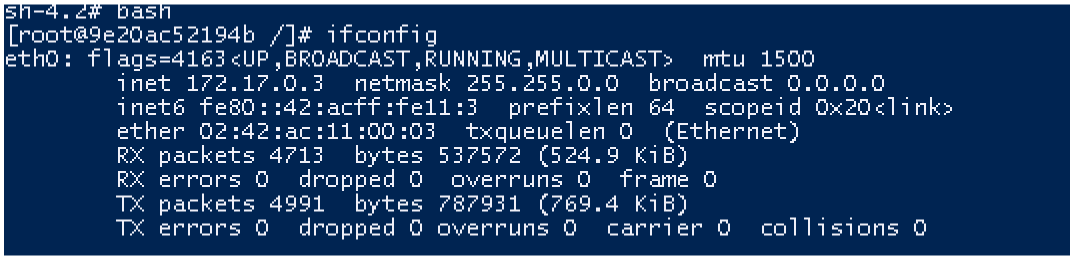
In the Kitematic screen, we should now see a new container added (ias), and the stdout of the container will show once we click on the container name from the sidebar.



```
##############################################
##                                          ##
##   InfoArchive Docker Container EntryPoint  ##
##   Author: Conrado Poole <conrado.poole@emc.com>  ##
##                                          ##
##############################################
2016-10-17T22:43:00.778761985Z
--------------------------------------------------------------------------------
2016-10-17T22:43:06.648605707Z
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.3.7.RELEASE)
2016-10-17T22:43:06.689267011Z
```

Figure 19 Kitematic InfoArchive Server

Using the exec button, let's open a command line inside the IA Server Container and retrieve the ip address for the InfoArchive Web App Container to use.



```
sh-4.2# bash
[root@9e20ac52194b /]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.3  netmask 255.255.0.0  broadcast 0.0.0.0
        inet6 fe80::42:acff:fe11:3  prefixlen 64  scopeid 0x20<link>
        ether 02:42:ac:11:00:03  txqueuelen 0  (Ethernet)
        RX packets 4713  bytes 537572 (524.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4991  bytes 787931 (769.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

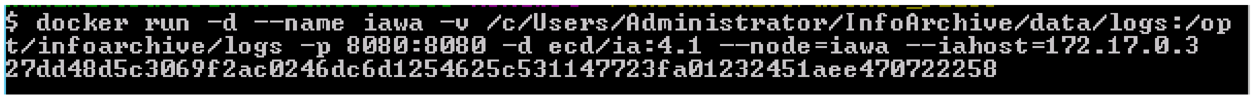
Figure 20 InfoArchive Server Ip Address

Note that for this container the ip address is (**172.17.0.3**).

IA Web App Container

To start the InfoArchive Web App container, type the following into the command line:

docker run -d --name iawa -v /c/Users/Administrator/InfoArchive/data/logs:/opt/infoarchive/logs -p 8080:8080 -d ecd/ia:4.1 --node=iawa --iahost=172.17.0.3

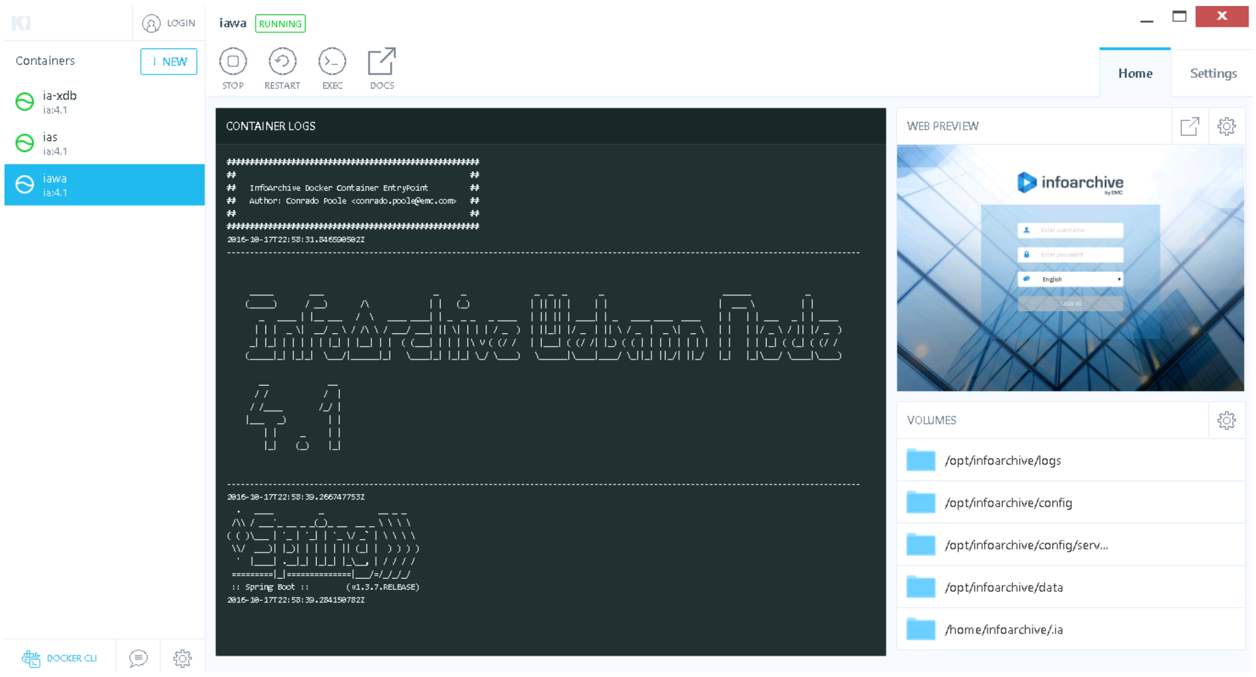


```
$ docker run -d --name iawa -v /c/Users/Administrator/InfoArchive/data/logs:/op
t/infoarchive/logs -p 8080:8080 -d ecd/ia:4.1 --node=iawa --iahost=172.17.0.3
27dd48d5c3069f2ac0246dc6d1254625c531147723fa01232451aee470722258
```

Parameter	Definition
--name iawa	The parameter specifies a name for container that is created and run by this command in this case (iawa). This makes it easier for us to locate this container instance
-v /c/users..:/opt/infoarchive/logs	The –v parameter informs Docker to map the following volume (/opt/infoarchive/logs) to the respective folder on our local host.
-p 8080:8080	Map the container port to the port in our Image.
-d ecd/ia:4.1	Name of the image used for creating this container
--node=iawa	A parameter passed to the entrypoint.sh file, which launches the respective node (IAWA) in this container.
--iahost=172.17.0.2	Specify the InfoArchive Server ipaddress for the InfoArchive Web Application to use.

Notice that in this docker run command we specified that the node type as InfoArchive Web app (--node=iawa) and that the InfoArchive Server is located at 172.17.0.3 (--iahost=172.17.0.3).

If we open the Kitematic Screen, we should now see three containers running. When clicking on the iawa container, we see the stdout on the right, and in the webpreview pane we will see the InfoArchive home page once the server starts.



```
############################################
##                                        ##
##   InfoArchive Docker Container EntryPoint  ##
##   Author: Conrado Poole <conrado.poole@emc.com>  ##
##                                        ##
############################################
2016-10-17T22:58:31.846590502Z
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
2016-10-17T22:58:39.266747753Z
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.3.7.RELEASE)
2016-10-17T22:58:39.284150782Z
```

Volumes
/opt/infoarchive/logs
/opt/infoarchive/config
/opt/infoarchive/config/serv...
/opt/infoarchive/data
/home/infoarchive/.ia

Figure 21 InfoArchive Web App Container

Click anywhere inside the Web Preview pane, and a browser will launch pointing to the InfoArchive Web home page.

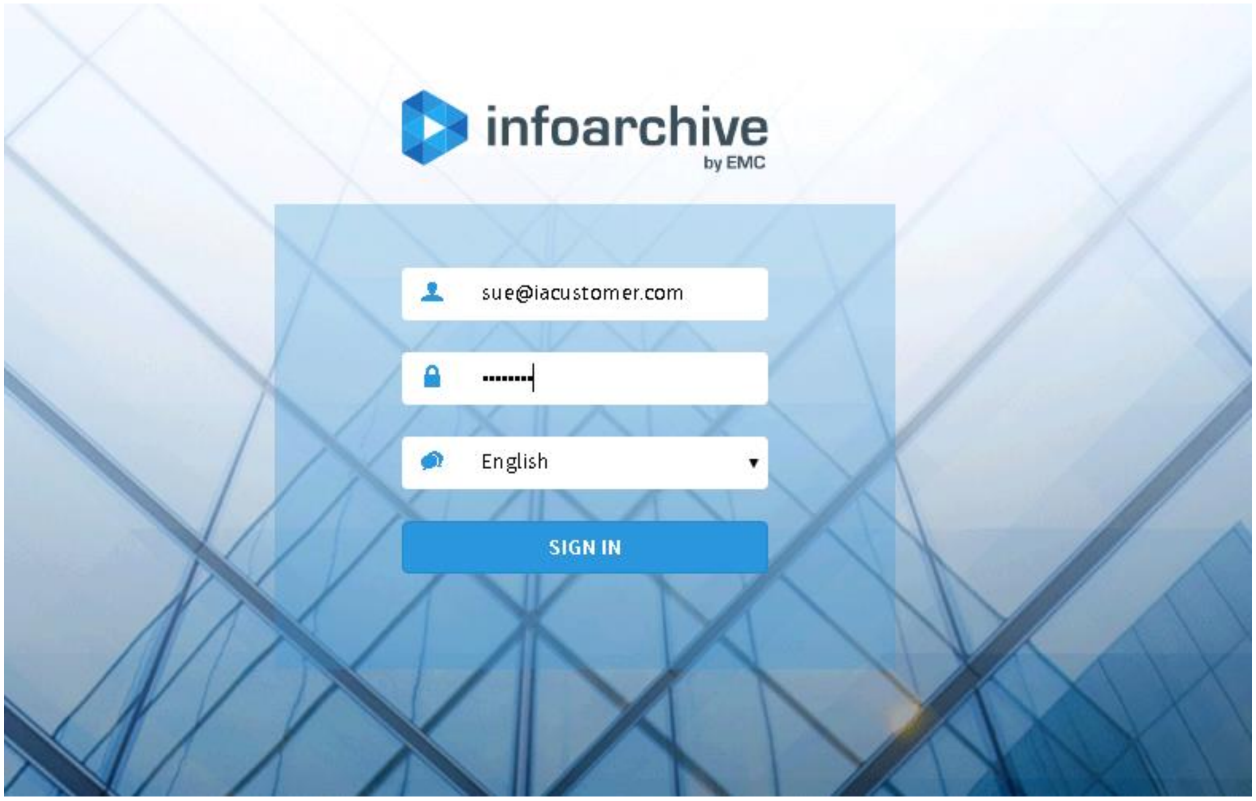
NOTE: It could take a couple of minutes for the InfoArchive Web Server to start and begin accepting responses.

This InfoArchive environment is current using the default configuration, and has a default in-memory authentication list

Try logging in with the different users provided in the following table and explore the InfoArchive 4.1 release.

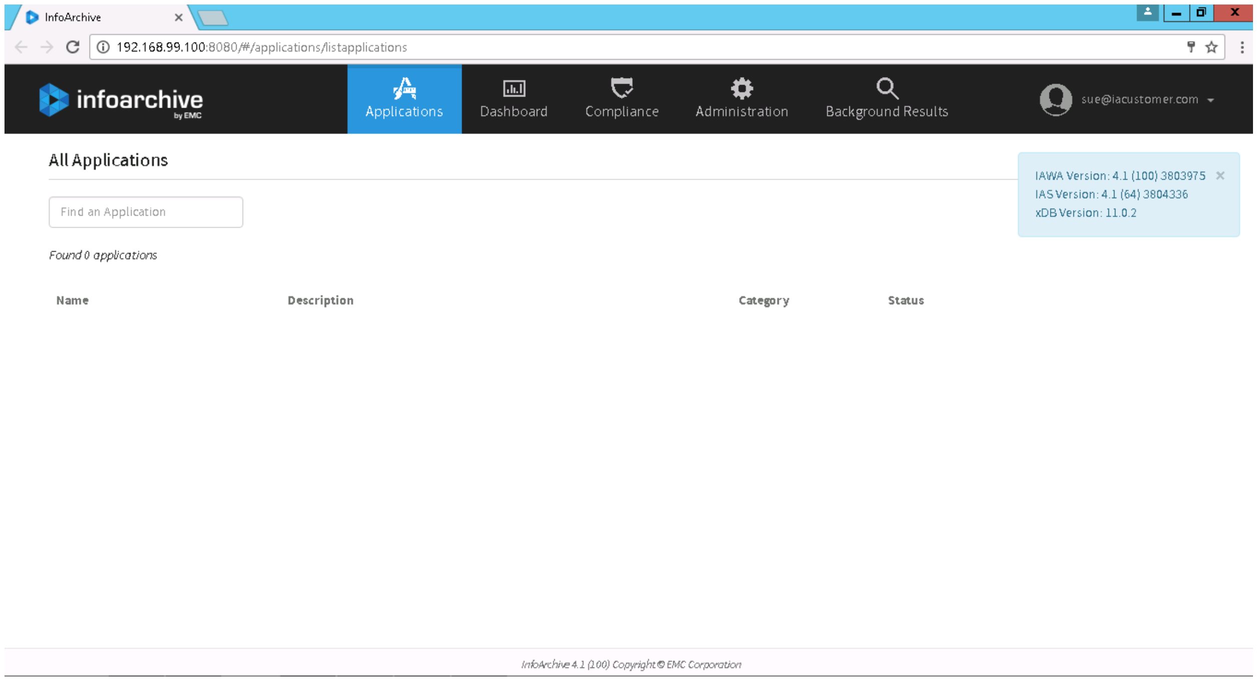
Role	Name	password
Administrator	adam@iacustomer.com	password
Business Owner	bob@iacustomer.com	password
Developer	connie@iacustomer.com	password
End User	emma@iacustomer.com	password

It Owner	Imran@iacustomer.com	password
Retention Manager	sue@iacustomer.com	password
All Roles	sue@iacustomer.com	password



sue@iacustomer.com
••••••••
English

Figure 22 InfoArchive Login



All Applications

IAWA Version: 4.1 (100) 3803975
IAS Version: 4.1 (64) 3804336
xDB Version: 11.0.2

Found 0 applications

Name	Description	Category	Status

InfoArchive 4.1 (100) Copyright © EMC Corporation

Figure 23 InfoArchive 4.1 Applications

InfoArchive Step 6 – Cleaning Things up

In the lab, we created three images – one being the Docker Hello World image, an InfoArchive Image a CentOS image that was used to build the InfoArchive image.. Let's delete those images and the containers we created from the system.

Remove Containers

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

We will need to grab the container ids before removing them. From the command prompt, enter in the following command –

docker ps –a

A list of container will be printed in the command line. When referencing a container you do not need to write down the entire container id. The first couple of characters will be enough.

From the command prompt, enter in the following command to stop the containers – we will need to perform this operation for all containers–

docker stop <container_id>

From the command prompt, enter in the following command to delete the containers from storage - we will need to perform this operation for all containers –

docker rm –f <container_id>

Verify the containers are not there any longer. From the command prompt, enter in the following command –

docker ps –a

Since we removed all containers we should not see any containers when running the docker ps command.

Remove Images

Now we will delete the images from the system.

From the command prompt, enter in the following command –

docker images

The command prints out the available images.

From the command prompt, enter in the following command –

docker rmi -f <imagename>:<tag>

For example to delete the InfoArchive image you will need to type the following command:-

docker rmi -f ecd/ia:4.1

After deleting all images. Enter the following command in the command line to see that there are no remaining images:-

docker images