








Docker

It's My Code in a Box. (Well, really, in a container)

What's the Problem?

- Deploying to production is difficult - more difficult than it should be
- Vendor-specific production environments hurt portability
- Environments are far more complex - multiple different systems all interconnected
- It's difficult to have dev / test environments match production - especially when your product supports multiple databases
- This results in the “matrix of doom”

“The Matrix of Doom”

Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?
							

How Does Docker Help?

- Docker creates standard “containers” that can be deployed and run anywhere the Docker engine runs
- Analogy: The shipping industry used to have custom containers for various goods. So... if goods were shipped somewhere, it was necessary to verify that it could be unpacked properly
- The shipping industry standardized on a single container - and allowed producers and consumers to get out of the shipping logistics business
- The developer - worries about what's in the box
- Infra teams - worry about what's outside of the box

What is it?

- A consistent and transportable approach for building, deploying, and running software in the cloud.
- Containers can be deployed in different vendor environments in a consistent way. (Mostly)
- A "build once, deploy anywhere" solution for the cloud.

Concepts and Components

- The Docker Engine
- Containers
- Images
- Docker Hub

The Docker Engine

- <https://www.docker.com/whatisdocker/>
- Leverages Linux's ability to sandbox processes*
- Similar to Android / iOS
- More lightweight than a VM, but more than just the app code

From Wikipedia: Docker uses resource isolation features of the [Linux kernel](#) such as [cgroups](#) and kernel [namespaces](#) to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting [virtual machines](#)

Containers and Images

- An excellent writeup: <http://paislee.io/how-to-automate-docker-deployments/>
- Image
 - An immutable file; a “snapshot” of a container
 - To save space, they’re built from lower level images
 - Different versions of the same image are supported
 - They can be built interactively, or from a Dockerfile
 - They can be managed in a registry (e.g. Docker Hub)
- Container
 - A running “instance” of an image
 - Can be launched as daemons or interactively
 - They can be linked to other containers
 - When they’re deleted, they’re gone forever

Docker Hub (<https://hub.docker.com/>)

- Their definition:
 - A centralized resource for working with Docker and its components
- My definition:
 - The GitHub of Docker images
- Hosts Docker images
- Supports discovery (search), including tagging some images as “official repos”
- A collaborative environment
- Integration with GitHub

Installation

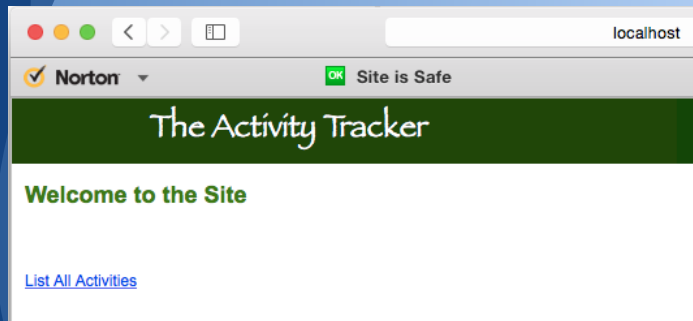
<https://docs.docker.com/#installation-guides>

- Linux - native install
- Win / Mac - Boot2Docker
- The web site has a great interactive tutorial
<https://www.docker.com/tryit/>
- Docker commands are run from a shell
- The basic “Hello World” example:

```
sudo docker images
sudo docker run --name kenny -i -t ubuntu /bin/bash
echo "Hello World"
exit
sudo docker ps -a
sudo docker rm kenny
```

Background - The Business Problem

- Activity tracker
 - Allows a user to track exercise
 - User, location, duration, distance
 - Basic CRUD operations
- Version 1: All data management done in memory



A screenshot of the 'Activity Details' form on 'The Activity Tracker' website. The form has a green header with the title 'The Activity Tracker'. Below the header, it says 'Activity Details' in green. The form contains three input fields: 'Location:', 'Distance: 0.0', and 'Duration:'. At the bottom, there is a button labeled 'Add Activity'.

A screenshot of the 'All Activities' page on 'The Activity Tracker' website. The page has a green header with the title 'The Activity Tracker'. Below the header, it says 'All Activities' in green. The page displays a table of activities with columns for 'Edit Location', 'Distance', and 'Duration'. There are two rows of data: 'Beach Run' with a distance of 12.1 and duration of 59, and 'Walk in Park' with a distance of 1.1 and duration of 60. Each row has a blue 'X' icon in the 'Edit Location' column. At the bottom, there is a blue link that says 'Log a new Activity'.

Edit Location	Distance	Duration
X Beach Run	12.1	59
X Walk in Park	1.1	60

[Log a new Activity](#)

Creating Images with a Dockerfile

- To support automation, images can be created by running a script (referred to as a Dockerfile)
- Two examples:

```
FROM ubuntu:14.04
RUN echo "deb http://archive.ubuntu.com/ubuntu trusty main universe" > /etc/apt/sources.list
RUN apt-get update && apt-get -y install software-properties-common
RUN add-apt-repository ppa:webupd8team/java
RUN apt-get update && apt-get -y upgrade
RUN echo oracle-java8-installer shared/accepted-oracle-license-v1-1 select true |
/usr/bin/debconf-set-selections
RUN apt-get -y install oracle-java8-installer && apt-get clean

RUN update-alternatives --display java
RUN echo "JAVA_HOME=/usr/lib/jvm/java-8-oracle" >> /etc/environment

RUN apt-get -y install tomcat7
RUN echo "JAVA_HOME=/usr/lib/jvm/java-8-oracle" >> /etc/default/tomcat7
RUN echo "CATALINA_BASE=/var/lib/tomcat7" >> /etc/environment
EXPOSE 8080
```

```
FROM tomcat:8.0
RUN apt-get update
ADD ./cfg/tomcat-users.xml /usr/local/tomcat/conf/tomcat-users.xml
ADD ./cfg/basicweb.war /usr/local/tomcat/webapps/basicweb.war
```

Creating Images with a Dockerfile

- Using the simpler tomcat example:

```
mvn package
cd Dockerfiles
cp ../target/basicweb.war ./cfg

docker build -f Dockerfile.one -t tdc9998/srq1 .

boot2docker ip

docker run -it --rm -p 8080:8080 tdc9998/srq1
```

- Launch a browser, and open the url:



Upload to Docker Hub

- The image exists locally; it's now time to push it to Docker Hub
- For some reason, this is called 'pushing a repository'

```
bash-3.2$ docker login  
Username: <your username>  
Password: <your password>  
Email: <your e-mail>  
Login Succeeded  
bash-3.2$
```

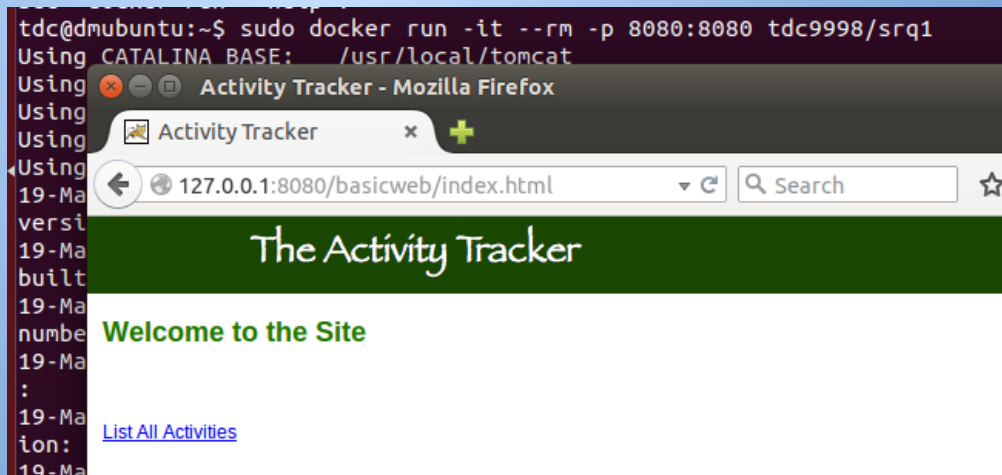
```
bash-3.2$ docker push tdc9998/srq1  
The push refers to a repository [tdc9998/srq1] (len: 1)  
Sending image list  
Pushing repository tdc9998/srq1 (1 tags)  
511136ea3c5a: Image already pushed, skipping  
...  
3b3a4796eef1: Image already pushed, skipping  
2c3e9237c2cf: Image successfully pushed  
0df212c3a283: Image successfully pushed  
6a8a1b7665d7: Image successfully pushed  
Pushing tag for rev [6a8a1b7665d7] on {https://cdn-registry-1.docker.  
io/v1/repositories/tdc9998/srq1/tags/latest}
```

- The image / repository has now been pushed to Docker Hub

Pulling from Docker Hub

- Now, from another machine, pull the image and start it up

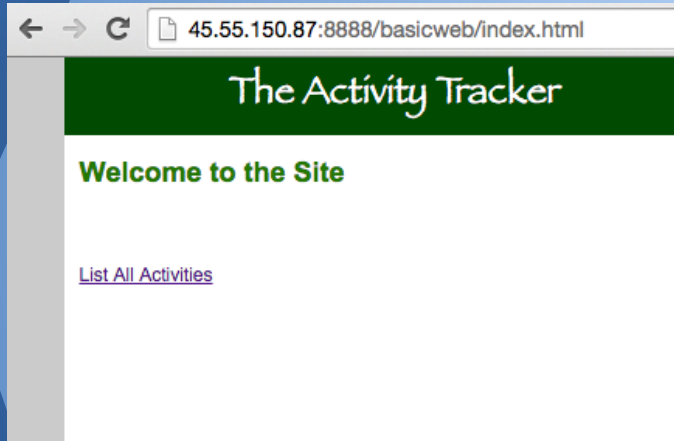
```
tdc@dmubuntu:~$ sudo docker pull tdc9998/srq1
Pulling repository tdc9998/srq1
6a8a1b7665d7: Pulling image (latest) from tdc9998/srq1, endpoint: https://regist6a8a1b7665d7:
Download complete
511136ea3c5a: Download complete
...
0df212c3a283: Download complete
Status: Downloaded newer image for tdc9998/srq1:latest
tdc@dmubuntu:~$ sudo docker run -it --rm -p 8080:8080 tdc9998/srq1
```



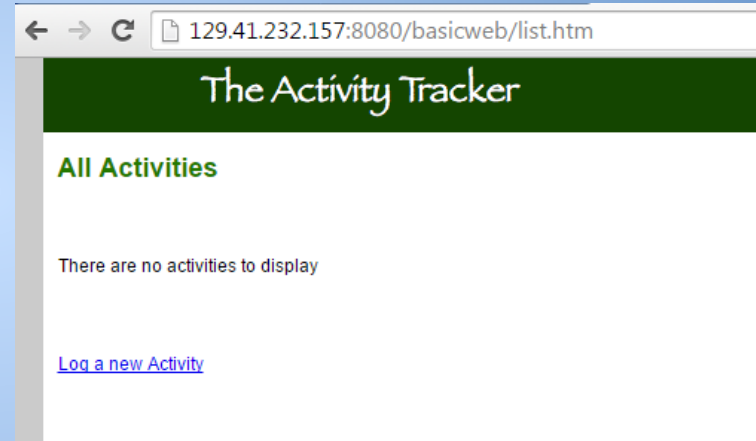
Deployment of a Container

Application deployed to both Bluemix and Digital Ocean

Digital Ocean



Bluemix



Deployment of a Container

- Bluemix uses a command line interface named “ice”

```
tdc@tdc-VirtualBox:~$ sudo docker pull tdc9998/srq1
Pulling repository tdc9998/srq1
6a8a1b7665d7: Pulling image (latest) from tdc9998/srq1, endpoint: https://regist6a8a1b7665d7:
0df212c3a283: Download complete
Status: Downloaded newer image for tdc9998/srq1:latest
```

```
tdc@tdc-VirtualBox:~$ sudo ice login
```

```
tdc@tdc-VirtualBox:~$ sudo ice --local tag -f tdc9998/srq1 registry-ice.ng.bluemix.
net/timle9/srq1
```

```
tdc@tdc-VirtualBox:~$ sudo ice --local push registry-ice.ng.bluemix.net/timle9/srq1
```

```
tdc@tdc-VirtualBox:~$ sudo ice run --publish 8080 --name timle9 registry-ice.ng.bluemix.
net/timle9/srq1:latest
```

```
tdc@tdc-VirtualBox:~$ sudo ice ip request
Successfully obtained ip: "129.41.232.xxx"
```

```
tdc@tdc-VirtualBox:~$ sudo ice ip bind 129.41.232.xxx timle9
Successfully bound ip
tdc@tdc-VirtualBox:~$ sudo ice ip unbind 129.41.232.xxx timle9
```

Checkpoint

- Docker concepts
- Review of a Java-based web app
- Build an image
- Push the image to Docker Hub
- Test everywhere
 - Locally: Mac OS, Ubuntu
 - Externally: Bluemix, Digital Ocean
- So far, there's no persistence

The Business Problem - Updated

- Add support for a relational database
- Checkout code branch - master

Data Volume Containers

- You may have noticed that all containers are transient; when they're deleted, they're gone forever
- That doesn't work so well if you'd like to have a database ;-)
- To persist data, use a data volume container
 - They can be shared
 - Changes are made directly (See: Union File System)
 - You can update an image without corrupting the data
 - They persist - even after a container is deleted
- The Dockerfile and associated commands:

```
FROM ubuntu
VOLUME /var/lib/mysql
```

```
sudo docker build -t mysqldataimage -f Dockerfile.dataVolume .
sudo docker run --name mysqldatactr mysqldataimage
```

MySQL - Create the database

- Now, create a database by using the data container with MySQL

Mac:

```
docker run --name mysqlctr -p 0.0.0.0:3306:3306 -e MYSQL_ROOT_PASSWORD=vanhalen --  
volumes-from mysqldatactr -d mysql
```

Linux:

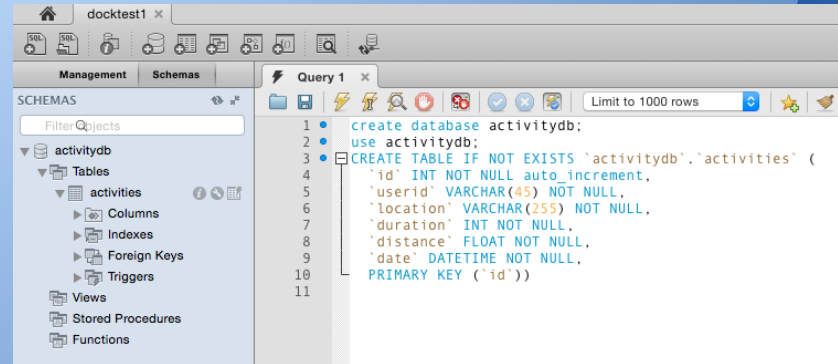
```
sudo docker run --name mysqlctr -p 3306:3306 -e MYSQL_ROOT_PASSWORD=vanhalen --  
volumes-from mysqldatactr -d mysql
```

- Since it's exposed on the local machine, I can connect to it with standard data modeling software (e.g. MySQL Workbench)
- I can manually create the database
- When I stop mysql, the data persists

```
docker stop --time=15 mysqlctr
```

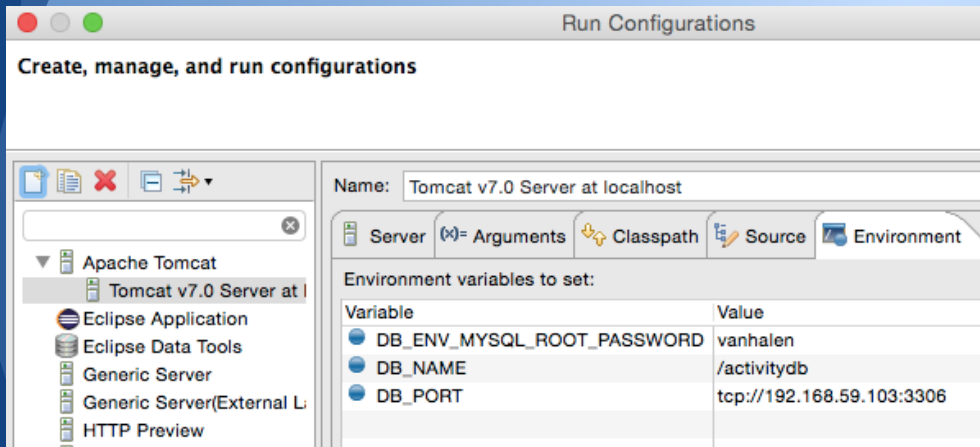
```
docker rm mysqlctr
```

```
docker run --name mysqlctr -p 0.0.0.0:3306:3306  
-e MYSQL_ROOT_PASSWORD=vanhalen --volumes-from  
mysqldatactr -d mysql
```



Use the Database - In Dev

- Interesting point: No need to have MySQL installed locally!
- I can connect my app (in dev) directly to the container
- I can even use different databases (dev, debug, test)
- Database-specific values are made available as environment variables
- This can be simulated in Eclipse / dev



```
9 @Configuration
10 public class DbConf {
11
12     @Bean
13     public DriverManagerDataSource dataSource() {
14         DriverManagerDataSource dmds = new DriverManagerDataSource();
15         Map<String,String> allEnv = System.getenv();
16
17         // Will be something like: tcp://172.17.0.2:3306 Remove the tcp
18         String ipAddr = allEnv.get("DB_PORT");
19         if ((ipAddr != null) && (ipAddr.length() > 4)) {
20             ipAddr = ipAddr.substring(3);
21         }
22         String pw = allEnv.get("DB_ENV_MYSQL_ROOT_PASSWORD");
23         String dbName = allEnv.get("DB_NAME");
24         String fullURL = "jdbc:mysql:"+ipAddr+dbName;
25         System.out.println("Full URL: " + fullURL);
26
27         dmds.setUrl(fullURL);
28         dmds.setDriverClassName("com.mysql.jdbc.Driver");
29         dmds.setUsername("root");
30         dmds.setPassword(pw);
31         return dmds;
32     }
33 }
```


Use the Database - In Production

- Tomcat exists in one container, and mysql exists in another
- They can communicate between each other via the --link option
 - --link <container name>:<alias>
- The alias is used to create environment variables, which are shared to the source container

```
mvn package
cd Dockerfiles
cp ../target/basicweb.war ./cfg
docker build -f Dockerfile.two -t tdc9998/srq2
.
```

```
docker run -t -i -p 8080:8080 --name srq2ctr --
link mysqlctr:db tdc9998/srq2
```

Critical environment variables
(Prefixed with DB_)



Environment Variables:

```
Key: DB_PORT_3306_TCP val: tcp://172.17.0.4:3306
Key: TERM val: xterm
Key: TOMCAT_VERSION val: 8.0.20
Key: TOMCAT_TGZ_URL val: https://www.apache.org/dist/tc
Key: JAVA_DEBIAN_VERSION val: 7u75-2.5.4-2
Key: DB_ENV_MYSQL_MAJOR val: 5.6
Key: PATH val: /usr/local/tomcat/bin:/usr/local/sbin:/usr/local/
Key: DB_PORT_3306_TCP_ADDR val: 172.17.0.4
Key: HOSTNAME val: e847bf79984b
Key: CATALINA_HOME val: /usr/local/tomcat
Key: PWD val: /usr/local/tomcat
Key: HOME val: /root
Key: TOMCAT_MAJOR val: 8
Key: DB_PORT_3306_TCP_PROTO val: tcp
Key: DB_PORT_3306_TCP_PORT val: 3306
Key: DB_PORT val: tcp://172.17.0.4:3306
Key: DB_ENV_MYSQL_VERSION val: 5.6.23
Key: DB_NAME val: /srq2ctr/db
Key: JAVA_VERSION val: 7u75
Key: DB_ENV_MYSQL_ROOT_PASSWORD val: vanhalen
```

Deployment 2.0

Take the new version and deploy it to Digital Ocean

Note: I manually re-create the database (this should be automated!)

```
root@tim1e9:~# docker run -it --link mysqlctr:mysql --rm mysql sh -c 'exec mysql -
h"$MYSQL_PORT_3306_TCP_ADDR" -P"$MYSQL_PORT_3306_TCP_PORT" -uroot -p"$MYSQL_ENV_MYSQL_ROOT_PASSWORD"'

mysql> create database activitydb;
Query OK, 1 row affected (0.00 sec)

mysql> use activitydb;
Database changed
mysql> CREATE TABLE IF NOT EXISTS `activitydb`.`activities` (
  ->   `id` INT NOT NULL auto_increment,
  ->   `userid` VARCHAR(45) NOT NULL,
  ->   `location` VARCHAR(255) NOT NULL,
  ->   `duration` INT NOT NULL,
  ->   `distance` FLOAT NOT NULL,
  ->   `date` DATETIME NOT NULL,
  ->   PRIMARY KEY (`id`))
  -> ENGINE = InnoDB;
Query OK, 0 rows affected (0.04 sec)
mysql> exit
Bye
root@tim1e9:~#
```


Deployment 2.0 - Digital Ocean

- The droplet is too small :-(I had to create a swap file
- <https://www.digitalocean.com/community/tutorials/how-to-configure-virtual-memory-swap-file-on-a-vps>
- Once configured, all worked well.

Checkpoint

- Data Volume Containers
- A server / daemon container
- Container linkage

Additional Concepts

- Create an image from a container
- Linking to the file system to run regression tests
- Backup / Restore
- Private registries
- The future
 - Docker Compose
 - Docker Machine
 - Docker Swarm