Docker

As my project started up I was looking for a good way to create my application and run it locally and on servers. I was looking into different technologies like Chef, Puppet and Salt but when I went to start up my Digital Ocean server it had another option I hadn't heard much about. I could create an Image with Docker. Since I didn't know what this was I decided to look into it.

What is Docker

The definition from their website is

```
Docker is an open platform for developers and sysadmins to build, ship, and run distributed application
```

allows me to create a simple container that can run my application. I wouldn't run all my components on it (DB and App server) but I would run my app server and code in one container and my DB in another. This sounded great to me. I have been a big fan of a single file for deploys for a long time, this was a step further. I could take my application package it up and then ship it as a whole container through my environments. The only thing that would have to change from environment to environment was the other containers it was wired to. I also could easily have different versions of java on my machine to run the application, different versions of tomcat, or of Postgresql. I could also create a container definition for any of these items, store them on Docker Hub and anyone on the team or even the build server, could pull them down and run the exact same version.

Creating a Container

One of the places that docker helps was with setting up a local DB. It allowed all of us on the team to have the same DB set up the same way. Here are the steps I went through to set up the container. For installing Docker here are some good links How to Use Docker on OS X: The Missing Guide and the main guide Installing Docker on Mac OS X Once you have docker installed you can start to create and run your container.

Create Dockerfile

```
#
# Dockerfile for Postgresql
#

FROM ubuntu
MAINTAINER Joseph Muraski

# Add the PostgreSQL PGP key to verify their Debian packages.
# It should be the same key as https://www.postgresql.org/media/keys/ACCC4CF8.asc
RUN apt-key adv --keyserver keyserver.ubuntu.com --recv-keys B97B0AFCAA1A47F044F244A07FCC7D46ACCC4CF8

# Add PostgreSQL's repository. It contains the most recent stable release
#     of PostgreSQL, ``9.3``.
RUN echo "deb http://apt.postgresql.org/pub/repos/apt/ precise-pgdg main" > /etc/apt/sources.list.d/pgd
```

```
# Update the Ubuntu and PostgreSQL repository indexes
RUN apt-get update

# Install ``python-software-properties``, ``software-properties-common`` and PostgreSQL 9.3
#  There are some warnings (in red) that show up during the build. You can hide
#  them by prefixing each apt-get statement with DEBIAN_FRONTEND=noninteractive
RUN apt-get -y -q install python-software-properties software-properties-common
RUN apt-get -y -q install postgresql-9.3 postgresql-client-9.3 postgresql-contrib-9.3

# Note: The official Debian and Ubuntu images automatically ``apt-get clean``
# after each ``apt-get``

# Run the rest of the commands as the ``postgres`` user created by the ``postgres-9.3`` package when it
USER postgres

# Create a PostgreSQL role named ``docker`` with ``docker_pass`` as the password and
# then create a database `docker_local_db` owned by the ``docker`` role.
# Note: here we use ``&&\`` to run commands one after the other - the ``\``
#       allows the RUN command to span multiple lines.
RUN    /etc/init.d/postgresql start &&\
psql --command "CREATE USER docker WITH SUPERUSER PASSWORD 'docker_pass';" &&\
createdb -O docker docker_local_db

# Adjust PostgreSQL configuration so that remote connections to the
# database are possible.
RUN echo "host all  all    0.0.0.0/0  md5" >> /etc/postgresql/9.3/main/pg_hba.conf

# And add ``listen_addresses`` to ``/etc/postgresql/9.3/main/postgresql.conf``
RUN echo "listen_addresses='*'" >> /etc/postgresql/9.3/main/postgresql.conf

# Expose the PostgreSQL port
EXPOSE 5432

# Add VOLUMEs to allow backup of config, logs and databases
VOLUME  ["/etc/postgresql", "/var/log/postgresql", "/var/lib/postgresql"]

# Set the default command to run when starting the container
CMD ["/usr/lib/postgresql/9.3/bin/postgres", "-D", "/var/lib/postgresql/9.3/main", "-c", "config_file=/
```

Dokcerfile does a few things, it install postgresql, sets up the users, the db. EXPOSE 5432 allows docker to expose the port outside the container. The VOLUME line allows for access to the data in those directories outside of the container, it also allows for other containers to use the folders. Finally CMD is the command that is run when the container is started.

Create Build Script

Here is a bash script that is used to create the image.

```
#!/bin/sh
docker build -t db_server/postgresql .
```

very simple file in the same location as the Dockerfile will create the image with the name db_server/postgresql.

Map the Ports

Since boot2docker runs in Virtual Box you will need to map the ports out so that you can connect to them locally. This script maps port 49155 from the Virtual Box instance to 5432 on the host machine.

```
#!/bin/sh

VBoxManage modifyvm "boot2docker-vm" --natpf1 "tcp-port-dockerDb49155,tcp,,5432,,49155";
VBoxManage modifyvm "boot2docker-vm" --natpf1 "udp-port-dockerDb49155,udp,,5432,,49155";
```

Create Container

As of Docker 1.3 you can create the container without running it. Before this you would have to run the container one way the first time and then use docker start on subsequent starts. Now you can use docker create then just start the container every time.

```
#!/bin/sh

docker create -d -p 49155:5432 -P --name dockerServerDb db_server/postgresql
```

Start Docker

```
#!/bin/sh

echo "Starting docker.....\n"
boot2docker up

echo "\nStarting dockerServerDb docker instance\nType following command to connnect to db"
echo "psql -h localhost -p 5432 -d docker_local_db -U docker --password\n\n"

docker start dockerServerDb
```

script makes sure that boot2docker is running. Then it starts the container. It is also nice enought to remind you how to connect to the db if you want to have and psql installed locally.

Conclusion

While setting up Postgresql is not a simple case it is a good example of how this could help you locally and also shows all the steps you would need to do. These files could all be checked into your source control and every developer on the team would now have the same db running the same way. You could easily have different versions of postgresql running for different projects or you could add other services like Redis if needed. I find docker to be a great tool and this is only the beggining of what it can do for you.