Adding a MariaDB Database Container

Posted At : April 19, 2018 2:15 PM | Posted By : Cutter
Related Categories: Development, Application Setup, Docker

> This multi-part series goes in depth in converting this site infrastructure to a containerized setup with Docker Compose. See the bottom of this post for other posts in the series.

Building on our **last post**, we're going to continue our step-by-step setup be talking more about the database setup. I had decided to use **MariaDB** for my database. For anyone unfamiliar, MariaDB was a fork of MySQL created by many of MySQL's core development team when Oracle bought MySQL, to maintain an open source alternative. Since this blog was using a MySQL database on the shared hosting platform, I needed something I could now use in our **DigitalOcean** Droplet.

In that last post I showed you the beginnings of our **Docker Compose** configuration.

```
version: "3.3"
services:
  database:
    container_name: mydb
    image: mariadb:latest
    env_file:
      - mariadb.env
    volumes:
      - type: bind
        source: ./sqlscripts
        target: /docker-entrypoint-initdb.d
    networks:
      my-network:
        aliases:
          - mysql
          - mydb
    restart: always
```

networks: my-network:

I explained the basics of this in the last post, but now let me go into some more depth on the finer points of the MariaDB container itself. First, most of the magic comes by using **Environment** variables. There are three different ways of handling setting environment variables with Docker Compose. First, you can define environment variables in a *.env* file at the root of your directory, with variables that would apply to all of your containers. Secondly, you can create specific environment variable files (in this case the *mariadb.env* file) that you can attach to containers using the **env_file**

configuration attribute, like we did above. And a third way is to add environment variables to a specific container using the **environment** configuration attribute on a service.

Why so many different ways to do the same thing? Use cases. The *.env* method is for variables shared across all environments. The *env_file* method can take multiple files, where you may need to define variables for more than one container and share them to another, but not all, and the *environment* method is just on that one container. There may even be instances where you use all three methods.

In that vein, let's look at a possible use case for a "global" environment variable. I want to use the same timezone in all of my containers. In my *.env* file I put the following:

```
TIMEZONE=America/Chicago
TZ=America/Chicago
```

I applied the same value to two separate keys, because some prebuilt containers look for it one way while others look for it another, but this is a perfect example of a "global" environment variable.

Now we can look at environment variables that are specific to our MariaDB container. Here's where things can get tricky. Some prebuilt containers are fairly well documented, some have no documentation at all, and most lie somewhere in between. The MariaDB container documentation is pretty good, but sometimes you have to dig in to get everything you need. Let's step in.

First, I needed MariaDB to setup the service. To do this right, you have to define the password for the **root** user. This is something that can go in your container specific environment variables, or the container specific environment variable file.

mariadb.env

```
MYSQL_ROOT_PASSWORD=mydbrootuserpw
```

While this will get the service up and running, it's not enough. I needed by blog database automatically setup by the build, as well as the *user* that my blog would use

to access the database. Luckily, the prebuilt MariaDB container makes this pretty easy as well.

mariadb.env

```
MYSQL_DATABASE=databaseiwantmade
MYSQL_USER=userofthatdb
MYSQL_PASSWORD=passwordofthatuser
```

Boom! Without any extra code I created my database and the user I needed. But...

This was just the first step. I now have the service, the database, and the user, but no data. How would I preseed my blog data without manual intervention? Turns out that was fairly simple as well. Though it's barely glossed over in the container documentation, you can provide scripts to fill your database, and more. Remember these lines from the Docker Compose service definition?

```
  ...
    volumes:
      - type: bind
        source: ./sqlscripts
        target: /docker-entrypoint-initdb.d
  ...
```

I was *binding* a local directory to a specific directory in the container. I can place any *.sql* or *.sh* file in that directory that I want, and the container will automatically run them in alphabetical order during the start up of the container.

OK. Backup. What? So, the container documentation says you can do this, but it doesn't really tell you how, or go into any kind of depth. So, I went and looked at that containers *Dockerfile* and found the following near the end:

```
ENTRYPOINT ["docker-entrypoint.sh"]
```

This is a Docker command that says "when you start up, and finish all the setup above me, go ahead and run this script." And, that script is in the GitHub repo for the MariaDB container as well. There's a lot of steps there as it sets up the service, and

creates that base database and user for you, and then there's this bit of magic:

docker-entrypoint.sh

```
for f in /docker-entrypoint-initdb.d/*; do
  case "$f" in
    *.sh)     echo "$0: running $f"; . "$f" ;;
    *.sql)    echo "$0: running $f"; "${mysql[@]}" < "$f"; echo ;;
    *.sql.gz) echo "$0: running $f"; gunzip -c "$f" | "${mysql[@]}"; echo ;;
    *)        echo "$0: ignoring $f" ;;
  esac
  echo
done
```

The secret sauce. Now, I don't do a ton of shell scripting, but I am a linguist who's been programming a long time, so I know this is a loop that runs files. It runs shell files, it runs the sql scripts, it'll even run sql scripts that have been zipped up gzip style. Hot Dog!

So, what it tells me is that the *files* it will automatically process need to be located in a directory ***/*docker-entrypoint-initdb.d**, which you see I mapped to a local directory in my Docker Compose service configuration. To try this out, I took my *blogcfc.sql* file, dropped it into my local *sqlscripts* mapped directory, and started things up. I was then able to use the command line to log into my container and **mysqlshow** to verify that not only was the database setup, but that it was loaded with data as well.

But, it gets better. I needed a database for my **Examples** domain as well. This required another database, another user, and data. Now, I like to keep the *.sql* script for data, and use a *.sh* file for setting up the db, user and permissions. I also wanted to put needed details in my *mariadb.env* file that I'll probably need in another (Lucee) container later.

mariadb.env

```
...
EXAMPLES_DATABASE=dbname
EXAMPLES_USER=dbuser
EXAMPLES_PASSWORD=userpw
...
```

Then, I created my shell script for setting up the Examples database, and dropped it

into that *sqlscripts* directory.

examples-setup.sh

```
#!bin/bash

mysql -uroot -p"${MYSQL_ROOT_PASSWORD}"<<MYSQL_SCRIPT
CREATE DATABASE IF NOT EXISTS $EXAMPLES_DATABASE;
CREATE USER '$EXAMPLES_USER'@'%' IDENTIFIED BY '$EXAMPLES_PASSWORD';
GRANT ALL PRIVILEGES ON $EXAMPLES_DATABASE.* TO '$EXAMPLES_USER'@'%';
FLUSH PRIVILEGES;
MYSQL_SCRIPT

echo "$EXAMPLES_DATABASE created"
echo "$EXAMPLES_USER given permissions"
```

Drop in an accompanying *.sql* script to the same directory, to populate the database (remember that all these scripts are run in alphabetical order), and now I have a database service to fulfill my needs. Multiple databases, multiple users, pre-seeded data, we have the whole shebang.

By the way, remember this?

.env

```
TIMEZONE=America/Chicago
TZ=America/Chicago
```

The MariaDB container took that second variable (TZ) and automatically set the service's timezone for us as well. Snap!

This post covered our first container, in our Docker Compose setup. Next post we'll continue our journey to setup a full environment.