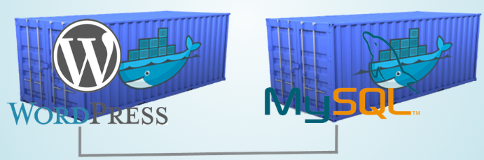
How to create one container that can access a service on another?

In this article, we'll show how Docker does it. We'll launch WordPress site using Docker. In this article, we'll create 3 different Docker images:

1. One image for WordPress itself.
2. It's going to be linked to MySQL container.
3. Then, it will be storing data. We call it "Data Container". Unlike the WordPress and MySQL containers, Data container can be run only once, and does not have to be running. It's enough to just define the volume and share the volume with MySQL. Then, the WordPress will be linked to the volume so that it can have an access to MySQL database's port, locally instead of remotely.



Where Docker stores its files?

Let's look inside **/var/lib/docker** folder in Ubuntu:

**$ sudo ls /var/lib/docker**

**aufs execdriver init repositories-aufs trust volumes**

**containers graph linkgraph.db tmp vfs**

When Docker starts an image and create a container that has a volume defined, it will store those in a UUID named directory inside two directories (**volumes** and **vsf**). Now the two directories are empty since we haven't defined any volume:

**k@laptop:~$ sudo ls /var/lib/docker/volumes**

**k@laptop:~$ sudo ls /var/lib/docker/vfs/dir**

Creating a volume

We're going to create a volume using tiny linux distro called [busybox](http://www.busybox.net/about.html" \t "_blank).

Docker system should maintain the consistency of the data store we're creating now in**/var/lib/mysql** directory.

For reference, here are the **docker run** args we're going to use to create a volume:

1. **-v**  
   Bind mount a volume (e.g., from the host: -v /host:/container, from Docker: -v /container)
2. **--name**  
   Assign a name to the container
3. **-d**  
   Detached mode: run the container in the background and print the new container ID.

Then, run docker. The following command just runs a container and exits as we can see from the output of **docker ps -a**:.

**k@laptop:~$ docker run -v /var/lib/mysql --name=my\_datastore -d busybox echo "My Datastore"**

**k@laptop:~$ docker ps -a**

**CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES**

**6a7e05a1e120 busybox "echo 'My Datastore' 3 minutes ago Exited (0) 3 minutes ago my\_datastore**

However, by running once, it has created a volume and this is the key: data container does not have to be running. If we inspect **my\_datastore**:

**k@laptop:~$ docker inspect my\_datastore**

**[**

**{**

**...**

**"Volumes": {**

**"/var/lib/mysql": "/var/lib/docker/volumes/7ec328ed9c22b7190c9998d13d3dc49d8d8249e336fd84667c5bf7fc724b70bc/\_data"**

**},**

**"VolumesRW": {**

**"/var/lib/mysql": true**

**},**

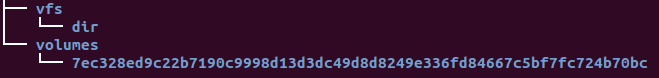
**...**

**}**

**}**

**]**

**k@laptop:~$**



So, from now on, whenever anything write to will be stored in this area of local host system!

MySQL container

Now, we need to start MySQL container ([OFFICIAL REPOSITORY mysql](https://hub.docker.com/_/mysql/)).

**k@laptop:~$ docker run --name my\_mysql -e MYSQL\_ROOT\_PASSWORD=my-secret-pw --volumes-from my\_datastore -d mysql**

**...**

**Status: Downloaded newer image for mysql:latest**

**9173d3457bc8ec440fbf5df30d30db8102ede04d4e344c9593a1421057f99786**

Note that we added **my\_mysql** for the container name, passed environment variable**MYSQL\_ROOT\_PASSWORD**, and most importantly, added our datastore for **--volumes-from**, and running it in detached mode.

**k@laptop:~$ docker ps**

**CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES**

**9173d3457bc8 mysql "/entrypoint.sh mysq 2 minutes ago Up 2 minutes 3306/tcp my\_mysql**

Now, we have MySQL up and running.

If we inspect **my\_mysql** container, we can see it's using the volume we specified:

**k@laptop:~$ docker inspect my\_mysql**

**...**

**"Volumes": {**

**"/var/lib/mysql": "/var/lib/docker/volumes/7ec328ed9c22b7190c9998d13d3dc49d8d8249e336fd84667c5bf7fc724b70bc/\_data"**

**},**

**"VolumesRW": {**

**"/var/lib/mysql": true**

**},**

**...**

Container for WordPress site

Note our WordPress container should have a link to our MySQL container!

Let's look into the inspection in the previous section, and we can locate the "ExposedPorts":

**k@laptop:~$ docker inspect my\_mysql**

**...**

**"ExposedPorts": {**

**"3306/tcp": {}**

**},**

**...**

We'll link to this port to WordPress so that it's available for WordPress internally:

**k@laptop:~$ docker run --link=my\_mysql:mysql -p 8888:80 -d wordpress**

Here, we used **--link** to add link to WordPress container in the form of **container-name:alias**, specified the ports: **hostPort:containerPort**, and run it in detached mode as before.

Now, we have two containers running:

**k@laptop:~$ docker ps**

**CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES**

**9b149061e09b wordpress "/entrypoint.sh apac 6 seconds ago Up 5 seconds 0.0.0.0:8888->80/tcp jovial\_noyce**

**9173d3457bc8 mysql "/entrypoint.sh mysq 30 minutes ago Up 30 minutes 3306/tcp my\_mysql**

Actually, we have three containers: two are running, and another holds the volume for datastore:

**k@laptop:~$ docker ps -a**

**CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES**

**9b149061e09b wordpress "/entrypoint.sh apac 2 minutes ago Up 2 minutes 0.0.0.0:8888->80/tcp jovial\_noyce**

**9173d3457bc8 mysql "/entrypoint.sh mysq 32 minutes ago Up 32 minutes 3306/tcp my\_mysql**

**6a7e05a1e120 busybox "echo 'My Datastore' About an hour ago Exited (0) About an hour ago my\_datastore**

Now, we're able to access the instance from the host without the container's IP, since we used standard port mappings.

Then, access it via http://localhost:8888 or http://host-ip:8888 in a browser:

