Romin Irani Follow
My passion is to help developers succeed. ¯_(ツ)_/¯
Jul 31, 2015 · 4 min read

# Docker Tutorial Series : Part 8 : Linking Containers

This is part 8 of the Docker Tutorial Series.

In this part, we shall take a look at how to link Docker Containers. By linking containers, you provide a secure channel via which Docker containers can communicate to each other.

Think of a sample web application. You might have a Web Server and a Database Server. When we talk about linking Docker Containers, what we are talking about here is the following:

1. We can launch one Docker container that will be running the Database Server.
2. We will launch the second Docker container (Web Server) with a link flag to the container launched in Step 1. This way, it will be able to talk to the Database Server via the link name.

This is a generic and portable way of linking the containers together rather than via the networking port that we saw earlier in the series. Keep in mind that this chapter covers Linking Containers via the—link flag. A new tool Docker Compose is the recommended way moving forward but for this tutorial, I will cover the—link flag only and leave it to the reader to look at Docker Compose.

Let us begin first by launching the popular NoSQL Data Structure Server Redis. Like other software, Redis too has its official Docker image available in the Docker Hub.

First, let us pull down the Redis image via the following command:

```
docker@boot2docker:~$ docker pull redis
```

Next, let us launch a Redis container (named **redis1**) in detached mode as follows:

```
docker@boot2docker:~$ docker run -d --name redis1 redis
37f174130f758083d243541e8adab7e2d8be2012e555cbdbd5fc67ca
9d26526d
```

We can check that **redis1** container has started via the following command:

```
docker@boot2docker:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
37f174130f75 redis "/entrypoint.sh redi 30 seconds ago
Up 29 seconds 6379/tcp redis1
```

Notice that it has started on port 6379.

Now, let us run a another container, a **busybox** container as shown below:

```
docker@boot2docker:~$ docker run -it --link redis1:redis
--name redisclient1 busybox
```

Notice the**—link** flag. The value provided to the**—link** flag is **sourcecontainername:containeraliasname**. We have chosen the value **redis1** in the **sourcecontainername** since that was the name that was given to our first container that we launched earlier. The **containeraliasname** has been selected as **redis** and it could be any name of your choice.

The above launch of container (**redisclient1**) will lead you to the shell prompt.

Now, what has this launch done for you. Let us observe first what entry has got added in the **/etc/hosts** file of the **redisclient1** container:

```
/ # cat /etc/hosts
172.17.0.23 26c37c8982e9
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
```

```
ff02::2 ip6-allrouters
172.17.0.21 redis 37f174130f75 redis1
/ #
```

Notice an entry at the end, where the container **redis1** has got associated with the **redis** name.

Now, if you do a ping by the host name i.e. alias name (**redis**)—it will work:

```
/ # ping redis
PING redis (172.17.0.21): 56 data bytes
64 bytes from 172.17.0.21: seq=0 ttl=64 time=0.218 ms
64 bytes from 172.17.0.21: seq=1 ttl=64 time=0.135 ms
64 bytes from 172.17.0.21: seq=2 ttl=64 time=0.140 ms
64 bytes from 172.17.0.21: seq=3 ttl=64 time=0.080 ms
```

If you print out the environment variables you will see the following (I have removed the other environment variables):

```
/ # set
....

REDIS_ENV_REDIS_DOWNLOAD_SHA1='0e2d7707327986ae652df7170
59354b358b83358'
REDIS_ENV_REDIS_DOWNLOAD_URL='http://download.redis.io/r
eleases/redis-3.0.3.tar.gz'
REDIS_ENV_REDIS_VERSION='3.0.3'
REDIS_NAME='/redisclient1/redis'
REDIS_PORT='tcp://172.17.0.21:6379'
REDIS_PORT_6379_TCP='tcp://172.17.0.21:6379'
REDIS_PORT_6379_TCP_ADDR='172.17.0.21'
REDIS_PORT_6379_TCP_PORT='6379'
REDIS_PORT_6379_TCP_PROTO='tcp'
....
```

You can see that various environment variables were auto-created for you to help reach out to the **redis1** server from the **redisclient1**.

Sounds good? Now , let us get a bit more adventurous.

Exit the **redis1** container and come back to the **boot2docker** prompt.

Let us launch a container based on the **redis** image but this time instead of the default command that will launch the redis server, we will simply

go into the shell so that all the redis client tools are ready for us. Note that the **redis1** server (container) that we launched is still running.

```
docker@boot2docker:~$ docker run -it --link redis1:redis
--name client1 redis sh
```

This will lead you to the prompt. Do a ping of redis i.e. our alias name and it should work fine.

```
# ping redis
PING redis (172.17.0.21): 48 data bytes
56 bytes from 172.17.0.21: icmp_seq=0 ttl=64 time=0.269
ms
56 bytes from 172.17.0.21: icmp_seq=1 ttl=64 time=0.248
ms
56 bytes from 172.17.0.21: icmp_seq=2 ttl=64 time=0.205
ms
```

Next thing is to launch the redis client (**redis-cli**) and connect to our redis server (running in another container and to which we have linked) as given below:

```
# redis-cli -h redis
redis:6379>
```

You can see that we have been able to successfully able to connect to the redis server via the alias name that we specified in the**—link** flag while launching the container. Ofcourse if we were running the Redis server on another port (other than the standard 6379) we could have provided the **-p** parameter to the **redis-cli** command and used the value of the environment variable over here (**REDIS_PORT_6379_TCP_PORT**). Hope you are getting the magic!

Now, let us execute some standard Redis commands:

```
redis:6379> PING
PONG
redis:6379> set myvar DOCKER
OK
redis:6379> get myvar
```

```
"DOCKER"
redis:6379>
```

Everything looks fine. Let us exit this container and launch another client (**client2**) that wants to connect to the same Redis Server that is still running in the first container that we launched and to which we added a string key / value pair.

```
docker@boot2docker:~$ docker run -it --link redis1:redis
--name client2 redis sh
```

Once again, we execute a few commands to validate things:

```
# redis-cli -h redis
redis:6379> get myvar
"DOCKER"
redis:6379>
```

You are all set now to link containers together.

## Additional Reading

Check out Docker Compose, which provides a mechanism to do the linking but by specifying the containers and their links in a single file. Then all one needs to do is use the docker-compose tool to run this file and it will figure out the relationships (which container needs to launch first, etc) and launch the containers for you. Moving forward, do expect Docker Compose to be the standard way to do linking.