docker run -v

Though we executed docker run commands with various argument combinations in the previous chapter([More on docker run command (docker run -it, docker run --rm, etc.)](http://www.bogotobogo.com/DevOps/Docker/Docker_Run_Command.php)),docker run was not doing useful operations.

So, in this chapter, we'll learn more about docker run commands that doing more useful things.

We're going to run docker run command with -v argument:

**k@laptop:~$ docker run -help**

**Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]**

**Run a command in a new container**

**-v, --volume=[] Bind mount a volume (e.g., from the host: -v /host:/container, from Docker: -v /container)**

**--volumes-from=[] Mount volumes from the specified container(s)**

Let's do it:

**k@laptop:~$ docker run -it --rm -v /home/k/myDocker:/k busybox sh**

**/ # cd k**

**/k # ls**

**/k # touch bogotobogo.txt**

**/k # exit**

**k@laptop:~$ cd /home/k/myDocker**

**k@laptop:~/myDocker$ ls**

**bogotobogo.txt**

**k@laptop:~/myDocker$ ls -la**

**total 8**

**drwxrwxr-x 2 k k 4096 Nov 22 12:16 .**

**drwxr-xr-x 89 k k 4096 Nov 22 12:15 ..**

**-rw-r--r-- 1 root root 0 Nov 22 12:16 bogotobogo.txt**

Here in the argument, we're binding a folder in our local machine (/home/k/myDocker) with the folder in Docker container (k) so that they can share files:

**-v /home/k/myDocker:/k busybox**

As we can see from the output, the two folders are sharing a file that was created in the container. Also note that the permissions on the file, "bogotobogo.txt". It was created with "root" user permission. A special care should be given so as not to the permission things not to be messed up. There is a way to work around it:

**k@laptop:~/myDocker$ sudo rm bogotobogo.txt**

**[sudo] password for k:**

**k@laptop:~/myDocker$ id k**

**uid=1000(k) gid=1000(k) groups=1000(k),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lpadmin),124(sambashare),1005(svn),131(docker)**

**k@laptop:~/myDocker$ docker run -it --rm -v /home/k/myDocker:/k -u 1000:1000 busybox sh**

**/ $ cd k**

**/k $ touch bogotobogo.txt**

**/k $ ls**

**bogotobogo.txt**

**/k $ exit**

**k@laptop:~/myDocker$ ls**

**bogotobogo.txt**

**k@laptop:~/myDocker$ ls -la**

**total 8**

**drwxrwxr-x 2 k k 4096 Nov 22 12:28 .**

**drwxr-xr-x 89 k k 4096 Nov 22 12:15 ..**

**-rw-r--r-- 1 k k 0 Nov 22 12:28 bogotobogo.txt**

This way we can keep the ownership remains the same. Since this may cause another problem, we need to be very careful when we mount the volume to the container.

docker run with port argument

In this section, we'll learn how to use port argument, -p, with Nginx web server.

**k@laptop:~$ docker run -help**

**Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]**

**Run a command in a new container**

**-d, --detach=false Detached mode: run the container in the background and print the new container ID**

**-P, --publish-all=false Publish all exposed ports to the host interfaces**

**-p, --publish=[] Publish a container's port to the host**

**format: ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort | containerPort**

**(use 'docker port' to see the actual mapping)**

We'll map container's port 80 to the host.

**k@laptop:~/myDocker$ docker run -d -p 80 nginx**

**Unable to find image 'nginx' locally**

**nginx:latest: The image you are pulling has been verified**

**f10807909bc5: Pull complete**

**f6fab3b798be: Pull complete**

**d21beea329f5: Pull complete**

**04499cf33a0e: Pull complete**

**34806d38e48d: Pull complete**

**4cae2a7ca6bb: Pull complete**

**23f7e46a4bbc: Pull complete**

**9dfd3384699f: Pull complete**

**475220486d0e: Pull complete**

**30bb1926e17f: Pull complete**

**ef45dc12127b: Pull complete**

**e426f6ef897e: Pull complete**

**511136ea3c5a: Already exists**

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

**72780dcf6c7ea4e14e497810722d297a8f4f8157099ea122e9345b76b0bab822**

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

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

**72780dcf6c7e nginx:latest "nginx -g 'daemon of 7 seconds ago Up 5 seconds 443/tcp, 0.0.0.0:49153->80/tcp condescending\_elion**

We did docker run in detached mode (-d) meaning making it running in background. As we can see from the PORT column in the output docker ps command, the Nginx on Docker container mapped port 80 of Nginx to 49153 port of host. So the port 49153 on local machine will go to port 80 of Docker Nginx.



If we want to specify the exact port on host, suppose 8099, we can do it:

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

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

**72780dcf6c7e nginx:latest "nginx -g 'daemon of 23 minutes ago Up 23 minutes 443/tcp, 0.0.0.0:49153->80/tcp condescending\_elion**

Stop the container and remove it:

**k@laptop:~/myDocker$ docker stop 72780dcf6c7e**

**72780dcf6c7e**

**k@laptop:~/myDocker$ docker rm 72780dcf6c7e**

**72780dcf6c7e**

Then, specify the host port number (8099) we want to use:

**k@laptop:~/myDocker$ docker run -p 8099:80 -d nginx**

**5444b242bf8d1f01229e11e0838ce11e918df03a038540b5c2dd66ec52023f08**

**k@laptop:~/myDocker$**



We can always check how the ports are mapped:

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

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

**5444b242bf8d nginx:latest "nginx -g 'daemon of 5 minutes ago Up 5 minutes 443/tcp, 0.0.0.0:8099->80/tcp mad\_ptolemy**

docker run -e : passing environment variable

Occasionally, we may need to passing in environment variable to docker run with -eargument.

**k@laptop:~/myDocker$ docker run -it --rm -e DOCK\_VAR=BOGOTOBOGO busybox sh**

**/ # echo $DOCK\_VAR**

**BOGOTOBOGO**

**/ # exit**

Passing in environment variables is useful when we deal with MySQL or password etc.

The 2nd sample : docker run -v

Suppose we have a file written in **go**, but we do not have the compiler. So, we decided to use Docker's **go** image, run the container, and compile it. Since we can share a file between host and container, after the compile, we get the executable on our host machine.

Here is our **go** file:

**k@laptop:~/golang$ cat HelloWorld.go**

**package main**

**import "fmt"**

**func main() {**

**fmt.Println("Hello World!");**

**}**

**docker run**:

**k@laptop:~/golang$ docker run -it --rm -v $(pwd):/go -u 1000:1000 golang:latest go build -o HelloWorld.out**

**Unable to find image 'golang:latest' locally**

**latest: Pulling from golang**

**902b87aaaec9: Pull complete**

**...**

**golang:latest: The image you are pulling has been verified. Important: image verification is a tech preview feature and should not be relied on to provide security.**

**Digest: sha256:2d94c1307f3ed7f679141f392ce545673397fd51e9395fcefe96500b610b68bb**

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

**k@laptop:~/golang$ ls**

**HelloWorld.go HelloWorld.out**

Now, we have an executable **HelloWorld.out** on our local host machine. Let's run it:

**k@laptop:~/golang$ ./HelloWorld.out**

**Hello World!**

Note that we were able to compile and run the **go** file even though **go** is not installed on our machine!

Note also that the **owner:group** is **k:k** because we used **1000:1000** in the **docker run**command:

**k@laptop:~/golang$ ls -la**

**total 2320**

**drwxrwxr-x 2 k k 4096 Aug 19 23:39 .**

**drwxr-xr-x 129 k k 4096 Aug 19 23:19 ..**

**-rw-rw-r-- 1 k k 75 Aug 19 23:21 HelloWorld.go**

**-rwxr-xr-x 1 k k 2361088 Aug 19 23:39 HelloWorld.out**