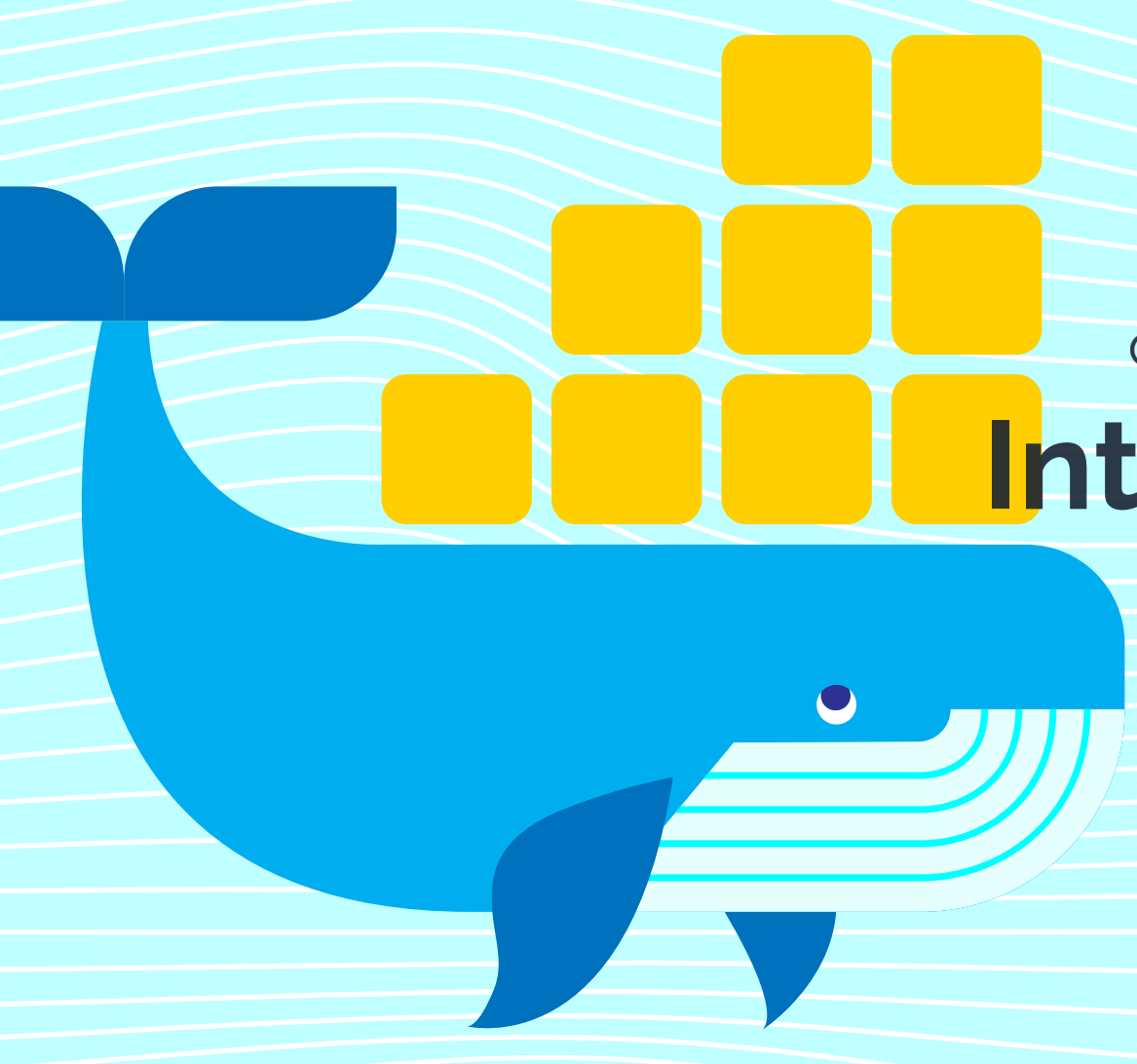

Chapter 2/Interacting-With-The-Container-Via-Volumes-And-Ports

Interacting with the container via volumes and ports

Let us get back to yt-dlp. It works yes, but it is quite laborious to get the downloaded videos to the host machine.

There are two ways to achieve this. We can use Docker [volumes](#) to make it easier to store the downloads outside the container's ephemeral storage. With [bind mount](#) we can mount a file or directory from our own machine (the host machine) into the container.

Let's start a container with the `-v` option. Prior to Docker Engine version 23, this option required an absolute path. However, as of Docker Engine version 23, you can use the `-v` option with relative paths on the host. We mount our current folder as `/mydir` in the container, overwriting everything that we have put in that folder in our Dockerfile.

```
$ docker run -v "$(pwd):/mydir" yt-dlp https://www.youtube.com/watch?v=saEpkcVi1d4
```

Now the downloaded video is saved to the working directory in the host machine, nice!

A Docker volume is essentially a shared directory or file between the host machine and the container. When a program running inside the container modifies a file within this volume, the changes are preserved even after the container is shut down and removed, as the file resides on the host machine. This is the primary advantage of using volumes; without them, any files created or modified within the container would be lost upon recreating it again from the image. Additionally, volumes facilitate file sharing between containers, enabling programs to access and load updated files seamlessly.

In our yt-dlp container we wanted to mount the whole directory since the files are fairly randomly named. If we wish to create a volume with only a single file we could also do that by pointing to it. For example `-v "$(pwd)/material.md:/mydir/material.md"` this way we could edit the file material.md locally and have it change in the container (and vice versa). **Note also that the `-v` option creates a directory if the specific file does not exist.**

Exercise:
1.9. Volumes

TRIES 0 POINTS 1/1

Your answer has been reviewed and graded. New submissions are no longer allowed.

Instructions

Image `devopsdockeruh/simple-web-service` creates a timestamp every two seconds to `/usr/src/app/text.log` when it's not given a command. Start the container with a bind mount so that the logs are created into your filesystem.

Hint: read the note that was made just before this exercise!

As the answer submit the command(s) you used to complete the exercise.

mkdir logs
touch logs/text.log
docker run -v ./logs/text.log:/usr/src/app/text.log devopsdockeruh/simple-web-service

Word count: 9

Allowing external connections into containers

This course does not provide an in-depth exploration of inter-program communication mechanisms. If you want to learn that in-depth, you should look at classes about Operating Systems or Networking. Here, you just need to know a few simple things:

- Sending messages: Programs can send messages to [URL](#) addresses such as this: http://127.0.0.1:3000 where HTTP is the *protocol*, 127.0.0.1 is an IP address, and 3000 is a *port*. Note the IP part could also be a *hostname*: 127.0.0.1 is also called *localhost* so instead you could use http://localhost:3000.
- Receiving messages: Programs can be assigned to listen to any available port. If a program is listening for traffic on port 3000, and a message is sent to that port, the program will receive and possibly process it.

The address *127.0.0.1* and hostname *localhost* are special ones, they refer to the machine or container itself, so if you are on a container and send a message to *localhost*, the target is the same container. Similarly, if you are sending the request from outside of a container to *localhost*, the target is your machine.

It is possible to **map a port on your host machine to a container port**. For example, if you map port 1000 on your host machine to port 2000 in the container, and then send a message to http://localhost:1000 on your computer, the container will receive that message if it's listening to its port 2000.

Opening a connection from the outside world to a Docker container happens in two steps:

- Exposing port
- Publishing port

Exposing a container port means informing Docker that the container listens to a certain port. This doesn't do much, except it helps humans with the configuration.

Publishing a port means that Docker will map host ports to the container ports.

To expose a port, add the line `EXPOSE <port>` in your Dockerfile

To publish a port, run the container with `-p <host-port>:<container-port>`

If you leave out the host port and only specify the container port, Docker will automatically choose a free port as the host port:

```
$ docker run -p 4567 app-in-port
```

We could also limit connections to a certain protocol only, e.g. UDP by adding the protocol at the end: `EXPOSE <port>/udp` and `-p <host-port>:<container-port>/udp`.

Security reminder: Opening a door to the internet

Since we are opening a port to the application, anyone from the internet could come in and access what you're running.

Don't haphazardly open just any ports - a way for an attacker to get in is by exploiting a port you opened to an insecure server. An easy way to avoid this is by defining the host-side port like this `-p 127.0.0.1:3456:3000`. This will only allow requests from your computer through port 3456 to the application port 3000, with no outside access allowed.

The short syntax, `-p 3456:3000`, will result in the same as `-p 0.0.0.0:3456:3000`, which truly is opening the port to everyone.

Usually, this isn't risky. But depending on the application, it is something you should consider!

Exercise:
1.10. Ports open

TRIES 0 POINTS 1/1

Your answer has been reviewed and graded. New submissions are no longer allowed.

Instructions

The image `devopsdockeruh/simple-web-service` will start a web service in port `8080` when given the argument "server". In [Exercise 1.8](#) you already did an image that can be used to run the web service without any argument.

Use now the -p flag to access the contents with your browser in http://localhost:8080. The output to your browser should be something like: `{ message: "You connected to the following path: ..." }`

Give command(s) used to start the service

docker run -p 127.0.0.1:8080:8080 web-server

Word count: 5

↑ Chapter front page

You've reached the end of this topic.

Proceed to the next topic

Next page:
Utilizing tools from the Registry