Defining start conditions for the container

Next, we will start moving towards a more meaningful image. yt-dlp is a program that downloads YouTube and Imgur videos. Let's add it to an image - but this time, we will change our process. Instead of our current process where we add things to the Dockerfile and hope it works, let's try another approach. This time we will open up an interactive session and test stuff before "storing" it in our Dockerfile.

By following the yt-dlp installation instructions we will start as follows:

```
$ docker run -it ubuntu:22.04

  root@8c587232a608:/# curl -L https://github.com/yt-dlp/yt-dlp/releases/latest/download/yt-dlp -o /usr
  bash: curl: command not found
```

..and, as we already know, curl is not installed - let's add `curl` with `apt-get` again.

```
$ apt-get update && apt-get install -y curl
$ curl -L https://github.com/yt-dlp/yt-dlp/releases/latest/download/yt-dlp -o /usr/local/bin/yt-dlp
```

At some point, you may have noticed that *sudo* is not installed either, but since we are *root* we don't need it.

Next, we will add permissions and run the downloaded binary:

```
$ chmod a+rx /usr/local/bin/yt-dlp
$ yt-dlp
/usr/bin/env: 'python3': No such file or directory
```

Okay, documentation mentions that Python 3.8 or later is needed to run yt-dlp. So let us install that:

```
$ apt-get install -y python3
```

We can now try to run the app again:

```
$ yt-dlp

  Usage: yt-dlp [OPTIONS] URL [URL...]

  yt-dlp: error: You must provide at least one URL.
  Type yt-dlp --help to see a list of all options.
```

It works, we just need to give it a URL.

So now when we know exactly what we need. Starting FROM ubuntu:22.04, we'll add the above steps to our `Dockerfile`. We should always try to keep the most prone to change rows at the bottom, by adding the instructions to the bottom we can preserve our cached layers - this is a handy practice to speed up the build process when there are time-consuming operations like downloads in the Dockerfile. We also added WORKDIR, which will ensure the videos will be downloaded there.

```
FROM ubuntu:22.04

WORKDIR /mydir

RUN apt-get update && apt-get install -y curl python3
RUN curl -L https://github.com/yt-dlp/yt-dlp/releases/latest/download/yt-dlp -o /usr/local/bin/yt-dlp
RUN chmod a+x /usr/local/bin/yt-dlp

CMD ["/usr/local/bin/yt-dlp"]
```

We have also overridden `bash` as our image command (set on the base image) with *yt-dlp* itself. This will not quite work, but let's see why.

Let us now build the Dockerfile as image `yt-dlp` and run it:

```
$ docker build -t yt-dlp .
  ...

$ docker run yt-dlp

  Usage: yt-dlp [OPTIONS] URL [URL...]

  yt-dlp: error: You must provide at least one URL.
  Type yt-dlp --help to see a list of all options.
```

So far so good. The natural way to use this image would be to give the URL as an argument but unfortunately, it does not work:

```
$ docker run yt-dlp https://www.youtube.com/watch?v=uTZSILGTskA

  docker: Error response from daemon: failed to create task for container: failed to create shim task:
  ERRO[0000] error waiting for container: context canceled
```

As we now know, *the argument we gave it is replacing the command* or `CMD`:

```
$ docker run -it yt-dlp ps
  PID TTY          TIME CMD
    1 pts/0    00:00:00 ps
$ docker run -it yt-dlp ls -l
total 0
$ docker run -it yt-dlp pwd
/mydir
```

We need a way to have something *before* the command. Luckily we have a way to do this: we can use ENTRYPOINT to define the main executable and then Docker will combine our run arguments for it.

```
FROM ubuntu:22.04

WORKDIR /mydir

RUN apt-get update && apt-get install -y curl python3
RUN curl -L https://github.com/yt-dlp/yt-dlp/releases/latest/download/yt-dlp -o /usr/local/bin/yt-dlp
RUN chmod a+x /usr/local/bin/yt-dlp

# Replacing CMD with ENTRYPOINT
ENTRYPOINT ["/usr/local/bin/yt-dlp"]
```

And now it works like it should:

```
$ docker build -t yt-dlp .
$ docker run yt-dlp https://www.youtube.com/watch?v=XsqlHHTGQrw
[youtube] Extracting URL:https://www.youtube.com/watch?v=XsqlHHTGQrw
[youtube] uTZSILGTskA: Downloading webpage
[youtube] uTZSILGTskA: Downloading ios player API JSON
[youtube] uTZSILGTskA: Downloading android player API JSON
[youtube] uTZSILGTskA: Downloading m3u8 information
[info] uTZSILGTskA: Downloading 1 format(s): 22
[download] Destination: Master's Programme in Computer Science | University of Helsinki [XsqlHHTGQrw].
[download] 100% of    6.29MiB in 00:00:00 at 9.95MiB/s
```

With *ENTRYPOINT* `docker run` now executed the combined `/usr/local/bin/yt-dlp https://www.youtube.com/watch?v=uTZSILGTskA` inside the container!

`ENTRYPOINT` vs `CMD` can be confusing - in a properly set up image, such as our yt-dlp, the command represents an argument list for the entrypoint. By default, the entrypoint in Docker is set as `/bin/sh -c` and this is passed if no entrypoint is set. This is why giving the path to a script file as CMD works: you're giving the file as a parameter to `/bin/sh -c`.

If an image defines both, then the CMD is used to give default arguments to the entrypoint. Let us now add a CMD to the Dockerfile:

```
FROM ubuntu:22.04

WORKDIR /mydir

RUN apt-get update && apt-get install -y curl python3
RUN curl -L https://github.com/yt-dlp/yt-dlp/releases/latest/download/yt-dlp -o /usr/local/bin/yt-dlp
RUN chmod a+x /usr/local/bin/yt-dlp

ENTRYPOINT ["/usr/local/bin/yt-dlp"]

# define a default argument
CMD ["https://www.youtube.com/watch?v=Aa55RKWZxxI"]
```

Now (after building again) the image can be run without arguments to download the video defined in CMD:

```
$ docker run yt-dlp

  youtube] Extracting URL: https://www.youtube.com/watch?v=Aa55RKWZxxI
  [youtube] Aa55RKWZxxI: Downloading webpage
  [youtube] Aa55RKWZxxI: Downloading ios player API JSON
  [youtube] Aa55RKWZxxI: Downloading android player API JSON
  ...
  [download] 100% of    5.60MiB in 00:00:00 at 7.91MiB/s
```

The argument defined by CMD can be *overridden* by giving one in the command line:

```
$ docker run yt-dlp https://www.youtube.com/watch?v=DptFY_MszQs
[youtube] Extracting URL: https://www.youtube.com/watch?v=DptFY_MszQs
[youtube] DptFY_MszQs: Downloading webpage
[youtube] DptFY_MszQs: Downloading ios player API JSON
[youtube] DptFY_MszQs: Downloading android player API JSON
[youtube] DptFY_MszQs: Downloading player 9bb09009
[youtube] DptFY_MszQs: Downloading m3u8 information
[info] DptFY_MszQs: Downloading 1 format(s): 22
[download] Destination: Welcome to Kumpula campus! | University of Helsinki [DptFY_MszQs].mp4
[download] 100% of   29.92MiB in 00:00:04 at 7.10MiB/s
```

In addition to all seen, there are two ways to set the ENTRYPOINT and CMD: **exec** form and **shell** form. We've been using the exec form where the command itself is executed. In shell form the command that is executed is wrapped with `/bin/sh -c` - it's useful when you need to evaluate environment variables in the command like `$MYSQL_PASSWORD` or similar.

In the shell form, the command is provided as a string without brackets. In the exec form the command and it's arguments are provided as a list (with brackets), see the table below:

Dockerfile	Resulting command
ENTRYPOINT /bin/ping -c 3 CMD localhost	/bin/sh -c '/bin/ping -c 3' /bin/sh -c localhost
ENTRYPOINT ["/bin/ping","-c","3"] CMD localhost	/bin/ping -c 3 /bin/sh -c localhost
ENTRYPOINT /bin/ping -c 3 CMD ["localhost"]	/bin/sh -c '/bin/ping -c 3' localhost
ENTRYPOINT ["/bin/ping","-c","3"] CMD ["localhost"]	/bin/ping -c 3 localhost

As the command at the end of Docker run will be the CMD we want to use ENTRYPOINT to specify what to run, and CMD to specify which command (in our case url) to run.

Most of the time we can ignore ENTRYPOINT when building our images and only use CMD. For example, Ubuntu image defaults the ENTRYPOINT to bash so we do not have to worry about it. And it gives us the convenience of allowing us to overwrite the CMD easily, for example, with bash to go inside the container.

We can test how some other projects do this. Let's try Python:

```
$ docker pull python:3.11
...
$ docker run -it python:3.11
Python 3.11.8 (main, Feb 13 2024, 09:03:56) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
Hello, World!
>>> exit()

$ docker run -it python:3.11 --version
  docker: Error response from daemon: OCI runtime create failed: container_linux.go:370: starting conta

$ docker run -it python:3.11 bash
  root@1b7b99ae2f40:/#
```

From this experimentat, we learned that they have ENTRYPOINT as something other than Python, but the CMD is Python and we can overwrite it, here with bash. If they had ENTRYPOINT as Python we'd be able to run `--version`. We can create our own image for personal use as we did in a previous exercise with a new Dockerfile:

```
FROM python:3.11
ENTRYPOINT ["python3"]
CMD ["--help"]
```

The result is an image that has Python as ENTRYPOINT and you can add the commands at the end, for example --version to see the version. Without overwriting the command, it will output the help.

Now we have two problems with the yt-dlp project:

- Major: The downloaded files stay in the container
- Minor: Our container build process creates many layers resulting in increased image size

We will fix the major issue first. The minor issue will get our attention in part 3.

By inspecting `docker container ls -a` we can see all our previous runs. When we filter this list with

```
$ docker container ls -a --last 3

  CONTAINER ID        IMAGE               COMMAND                  CREATED                  STATUS
  be9fdbcafb23        yt-dlp          "/usr/local/bin/yout…"    Less than a second ago   Exited (0) Abo
  b61e4029f997        f2210c2591a1        "/bin/sh -c \"/usr/lo…"   Less than a second ago   Exited (2)
  326bb4f5af1e        f2210c2591a1        "/bin/sh -c \"/usr/lo…"   About a minute ago       Exited (2)
```

We see that the last container was `be9fdbcafb23` or `determined_elion` for us humans.

```
$ docker diff determined_elion
  C /root
  A /root/.cache
  A /root/.cache/yt-dlp
  A /root/.cache/yt-dlp/youtube-nsig
  A /root/.cache/yt-dlp/youtube-nsig/9bb09009.json
  C /mydir
  A /mydir/Welcome to Kumpula campus! | University of Helsinki [DptFY_MszQs].mp4
```

Let's try `docker cp` command to copy the file from the container to the host machine. We should use quotes now since the filename has spaces.

```
$ docker cp "determined_elion://mydir/Welcome to Kumpula campus! | University of Helsinki [DptFY_MszQs
```

And now we have our file locally and we can watch it if the machine has a suitable player installed. Sadly, the use of `docker cp` is not proper to fix our issue. In the next section, we will improve this.

Improved curler

With `ENTRYPOINT` we can make the curler of the Exercise 1.7. more flexible.

Change the script so that it takes the first argument as the input:

```
#!/bin/bash

echo "Searching..";
sleep 1;
curl http://$1;
```

And change the CMD to ENTRYPOINT with the format `["./script.sh"]`. Now we can run

```
$ docker build . -t curler-v2
$ docker run curler-v2 helsinki.fi

  Searching..
    % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                   Dload  Upload   Total   Spent    Left  Speed
  100   232  100   232    0     0  13647      0 --:--:-- --:--:-- --:--:-- 13647
  <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
  <html><head>
  <title>301 Moved Permanently</title>
  </head><body>
  <h1>Moved Permanently</h1>
  <p>The document has moved <a href="https://www.helsinki.fi/">here</a>.</p>
  </body></html>
```

Edit this page

Previous
« In-depth dive into images

Next
Interacting with the container via volumes and ports »