Hyperiondev

TASK

Containers: Docker

Visit our website

Copyright © 2021 HyperionDev. All rights reserved.

Introduction

GETTING YOUR APP OUT THERE

At this point in the course you have had exposure to developing programs using different frameworks and libraries. You may have wondered how people keep their heads wrapped around all the different requirements of making a program work when running it somewhere that isn't your PC. There are many different ways of doing this and in this task, we'll be covering one such method that has recently become very popular: containerisation.

WHAT IS DEPLOYMENT?

Deployment is the technical name for getting your program out to the world and running on your users' devices — be it in a web browser on a laptop, as a smartphone app, or even on a smart-fridge. Back in the early days of software development this usually only happened once for a software product, since it was very difficult to update an app after its deployment. Today, with more devices connected to the internet than there are people walking the Earth, it's easier than ever to update apps. Nowadays it's common for an app to be deployed multiple times a month — just look at how frequently your smartphone's apps update!

From the developer's point of view, there are many ways to deploy an app. You've already seen one such way earlier in this course — GitHub Pages. It's wonderful for simple web pages but is not intended for anything more complex. If you want your app to store and retrieve data on a database, or handle user authentication, or almost anything else, you'll need a more comprehensive environment.

TYPICAL DEPLOYMENT

Different kinds of applications deploy differently. For instance, mobile apps deploy with systems managed by Google Play Store or iOS App Store; Windows 10 apps deploy with Microsoft Store. The most common way of deploying web-based applications is by using remote virtual machines (VMs). These are special kinds of operating systems that are built to work with data and interact with other programs (not people) — they are light on system resources and come without any graphics. In this kind of environment all your code and its dependencies would then be installed. Doing this requires a background in system administration and will not be covered in this course.

Copyright © 2021 HyperionDev. All rights reserved.

CONTAINERISATION

Luckily in 2013, a company called Docker developed a much simpler way of doing this: using containers. A container is a virtual environment you can configure any way you like with the purpose of being a fully-encompassing package of your software. So ideally you can just give the container to anyone with a PC and they'll be able to run your program with no hassle. All your application's dependencies would be included for you.

GETTING DOCKER RUNNING

Docker Desktop is an app that helps you manage Docker containers on your machine. You can install it for your operating system here: **https://www.docker.com/products/docker-desktop**. If your machine does not meet the system requirements, you'll have to do this entire task using Docker Playground.

Docker Playground is a service that gives you an online virtual space in which you can spin up and test Docker containers to your heart's content. It works by giving you access to a VM with Docker already installed. Go to this website and login with your Docker Hub account: **https://labs.play-with-docker.com/**. If you don't have a Docker Hub account, create one now. You'll be needing it later in this tutorial anyway: **https://hub.docker.com/**.

After logging in with Docker Playground, click "Start". The playground has a 4-hour time limit, after which everything you did there will be deleted. It should be enough time to do this entire task, but if you feel you need more time, we recommend doing this task in two batches: first time following along with the tutorial, and second time doing the compulsory tasks. In between these batches you should click on "Close Session" and start again.

After clicking "Start" you'll be able to create instances. These instances are each a kind of VM, where you can execute any commands you like. This is possible because of a technology called SSH (Secure Shell). SSH is the de-facto way of executing remote commands on VMs. It's the reason you are able to type commands into a terminal in your web browser while they execute on a machine in the US.

If you got Docker Desktop working, you'll notice it provides a neat graphical user interface (GUI), but we'll only be using the command line version (which comes installed with Docker Desktop) in this task. This is because the command line gives

Copyright © 2021 HyperionDev. All rights reserved.

us much more flexibility and declarability for using Docker. Its syntax is also platform independent (it works the same on any operating system), and it better helps you understand how Docker works.

HELLO WORLD

To verify that Docker is installed and works, run the following command from a terminal or command line. Note that Docker recommends always issuing commands in elevated mode (run as admin on Windows or root/sudo on macOS and Linux). Docker Playground automatically uses root to issue commands.

```
docker run hello-world
```

It will do some downloads and, finally, you'll see output starting with "Hello from Docker!". The output explains what it did, but in a nutshell, Docker downloaded an image (which is like a container specification), from which it created a container, within which it ran a program, the output of which was sent to your terminal. Without you needing to install any extra software it just did all that. And it can do much more.

CREATING A SIMPLE CONTAINER

We will now create a simple webpage and containerise it so that it can be served from any machine. The first step is to make the webpage. Using your favourite IDE, create a file called `index.html` and put anything you want in it. We recommend a few styled lines of text and perhaps an image. Feel free to use Bootstrap to style your page.

If you're using Docker Playground, we recommend you use the command line text editor `vim` to create and edit files. Guide **here**. To get files you have on GitHub on your machine, use `git clone [repo_url]` to get it on the VM.

A **Docker container** is an environment in which your code will run. A **Docker image** is the bundle of your code and its dependencies that can be created into said container. We need to create a Docker image and, to do so, we must tell Docker exactly what we need in it. We do this by specifying a Docker file. Create a file called `Dockerfile` (no extension) and place the following lines in it:

Copyright © 2021 HyperionDev. All rights reserved.

```
FROM nginx
COPY . /usr/share/nginx/html
```

Docker has a great open source community of developers publishing their images for everyone to use for free, called Docker Hub. Later in this task, you'll be publishing your own Docker image and becoming part of that community too. One of these developer groups created an image called **`nginx`**, which is a popular web server. The first line in our Docker file tells Docker to fetch this image from the Docker Hub because we'll be building on it and using it for our own image. This is common practice in containerisation.

The second line tells Docker to copy everything in our directory (that's what the dot means) to a specific directory inside the container. This directory is where **`nginx`** serves its web content. So we definitely want our **`index.html`** and other web files copied in there.

We will now create a Docker image. Make sure all your web files and the Docker file are in the same directory and then use **`cd`** to go to that directory in your terminal. Run the following command:

```
docker build -t my-website ./
```

This will first download the nginx image and then copy your web files into a new image-based thereon. The **`-t`** flag specifies the tag (name) of your image. The **`./`** argument indicates that the image should be built from the current directory. The image will now have been created and exists somewhere on your machine. You don't have to worry about where it is or what it looks like — all you need to know is how to create it from your Docker file. This is the idiomatic way of using and exporting images.

RUNNING IT

To create the container, you "run" the image. To do so, issue the following command:

```
docker run -d -p 80:80 my-website
```

The **`-d`** flag tells Docker to run the image in detached mode (in the background). The **`-p`** flag specifies that your machine's network port 80 should link to the container's port 80. This creates a network tunnel of sorts between your computer

Copyright © 2021 HyperionDev. All rights reserved.

and the container. You'll see that the command outputs a long alphanumeric string. This is the ID of the running container. You'll be needing it later, but don't worry about writing it down — there are easy ways of getting that ID again from Docker.

If you're running Docker on your PC. open your browser and go to **http://localhost**. If you're using Docker Playground, you should see a button with the number "80" next to "OPEN PORT" at the top of the webpage. Click on it to visit that VM's exposed port 80.

You should now see your website!

By default, **nginx** will keep serving indefinitely. Shut down your container by first getting its ID from this command:

```
docker ps
```

You'll notice that it only prints the first 12 characters of the container's ID. This is because Docker is smart enough to understand which container you're referring to without listing the entire ID. Go ahead and shut down the container using only the first three characters of its ID:

```
docker stop [ID]
```

DEPLOYING YOUR DOCKER IMAGE

Now that your Docker image is ready to run, we'll look at how we can deploy it so that anyone on any machine can run it. The first step is to publish it on Docker Hub. Go create a free account now if you haven't yet: **https://hub.docker.com/**.

Click the "Create Repository" button to create your own Docker repository. It's much like a GitHub repository, except it's just for Docker images. Also, every Docker program is automatically configured to interact with Docker Hub easily. Call it something simple, like **my-website**, and make sure it's public.

Before uploading it, you need to rename your image so that it matches your repository. The format is `user/repo`, where **user** is your Docker Hub username and **repo** is the repository's name.

Copyright © 2021 HyperionDev. All rights reserved.

```
docker tag my-website [user]/[repo]
```

Before you can upload your image, you need to login from the command line. Use the command below. It will prompt you for your username and password.

```
docker login
```

You can now upload your image.

```
docker push [user]/[repo]
```

RUNNING YOUR IMAGE ELSEWHERE

To prove that your Docker image can run anywhere, we'll be running it on a VM hosted in the USA. If you haven't been using Docker Playground thus far, go to this website and login with your Docker Hub account: **https://labs.play-with-docker.com/**. Then click "Start", and on the next screen click "add new instance".

In the terminal, issue the following command to automatically download and run your image:

```
docker run -d -p 80:80 [user]/[repo]
```

Docker on the VM will download your image from Docker Hub and do all the relevant preparations and installation to make your app work. Docker is pretty smart when it comes to images. It doesn't just make a sealed package with all the necessary code and dependencies. Instead it uses a modular approach and splits an image up into layers - each having only what is necessary to build upon the previous layer. For example, the image you just built has an nginx layer, and then your layer (your code). Nginx will have its own set of layers too. You can see each layer being downloaded and set up individually in the docker command's output.

Luckily all this is managed automatically by Docker and you need only to concern yourself with your layer.

When the run command is done, you should see a button with the number "80" next to "OPEN PORT" at the top of the webpage. Click on it to visit that VM's

Copyright © 2021 HyperionDev. All rights reserved.

exposed port 80. You should see your website now running completely on its own in the world wide web.

CONSOLE-BASED CONTAINERS

Containers are usually used for web applications, but sometimes you may want to use it for console-based apps too. In a new directory, create a file called **hello.py** and make it print something simple to output. Ensure that it works normally. If you're using Docker Playground, test the code on your machine first before putting it on the VM. The VM comes with Python installed. However, we'll specifically be using a different Python interpreter that doesn't come installed with the VM, PyPy.

Create a Docker file in the same directory with the following contents (if you're using Docker Playground, use one of the described methods to get it on the VM):

```
FROM pypy:latest
WORKDIR /app
COPY . /app
CMD python hello.py
```

- The first line bases the image off of PyPy, which is an alternative implementation of Python. We specified the latest version. The Python version you've been using is commonly referred to as CPython. However, there are different implementations of Python's specification that are better optimised for a range of scenarios. PyPy is one of them.
- The `WORKDIR` is the working directory where your app will be built and executed.
- Finally, `CMD` specifies the main command to be run when the image boots.
- Note that `WORKDIR` and `CMD` aren't needed for our web app, because `nginx` takes care of its own execution.

Build and run the image:

```
docker build -t python-app ./
docker run python-app
```

Note that we do not have to specify any ports or create a network tunnel. Docker automatically pipes the output of the `CMD` command to your console.

Copyright © 2021 HyperionDev. All rights reserved.

You should now see the output of your application printed directly to your console. If your Python script takes command-line input, you should specify the `-i` flag, which passes your command line input directly to the containerised app. E.g:

```
docker run -i python-app
```

Afterwards the container automatically shuts down because it has nothing further to do (unlike our web app that keeps serving at all times).

Try pushing your new image to Docker Hub and running it in Docker Playground to check that it works anywhere.

CAVEATS

Containers are cool, but they can't do everything. Most notably, containers are very bad at running GUI programs. This is because GUI programs expect special libraries, temporary directories, and internal network ports in order to interact with your desktop environment properly. While it is possible, it's very difficult to set up and quite prone to bugs. For this course you'll only ever be expected to manage containers for console-based and network-based apps. This is generally true in industry too.

Compulsory Task 1

Containerise your Python app application from the last capstone of Level I. Do all of the following instructions:

- Create an appropriate `Dockerfile` for your code.
- Create an image from this `Dockerfile`.
- Upload your image to Docker Hub.
- Ensure that your image can run on a machine that isn't your computer by using Docker Playground.

Submit the following:

- The `Dockerfile` you created.
- A link to your Docker Hub repository in a text file called **docker1.txt**.

Copyright © 2021 HyperionDev. All rights reserved.

Compulsory Task 2

Containerise your console-based Python app from any previous capstone. Follow all of the following instructions:

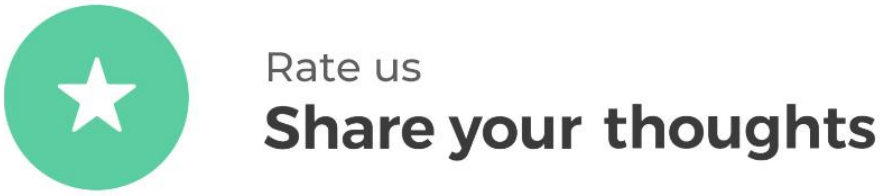
- Create an appropriate `Dockerfile` for your code.
- Create an image from this `Dockerfile`.
- Upload your image to Docker Hub.
- Ensure that your image can run on a machine that isn't your computer by using Docker Playground.

Submit the following:

- The `Dockerfile` you created.
- A link to your Docker Hub repository in a text file called **docker2.txt**.

Optional Bonus Task

For all past and future capstone projects, if you decide to publish your code on GitHub, include a working Dockerfile. This shows future recruiters that you understand containerisation, which is a big bonus in the software developer market.



HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.

Copyright © 2021 HyperionDev. All rights reserved.