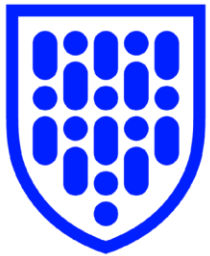


Institute of Data

2022

Software Engineering

Module 10 - Part 1

Deployment And Maintenance

Agenda

Section 1 : Introduction to Docker

Section 2 : Dockerise a NodeJs Application

Section 3 : Introduction to GitHub Actions

Section 4 : Adding GitHub Actions to your Application

Institute of Data

Section 1 : Introduction to Docker

It is natural for programs to be able to communicate with one another in order to create interactive and scalable applications. An API (short for Application Programming Interface) is a collection of rules that allows various programs to communicate with one other. These programs could be a software library (for example, the Python API), an operating system, or a web server (a web API).

One of the most significant advantages of an API is that the requester and responder do not need to be familiar with each other's software. This enables services employing various technologies to communicate in a consistent manner.

What is a software container?

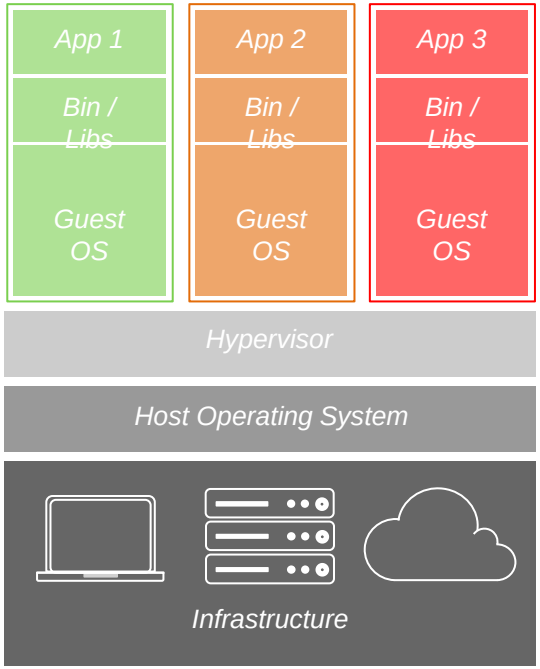
DEFINITION	*A software container is an example of "OS-level virtualisation", which enables a group of processes to run isolated in a separate space with their own file system and network interfaces.*
IMPLEMENTATION	*Containerisation is made possible by some of the characteristics of the recent Linux kernels that allow the isolation of a group of processes through the concept of "spaces" and "process groups".* *Within a single node, the kernel is shared across multiple containers that run isolated from each other.*
BENEFITS	*Containers allow the same degree of isolation and separation provided by virtual machines but are much lighter and less demanding in terms of resources.*

What is Docker?

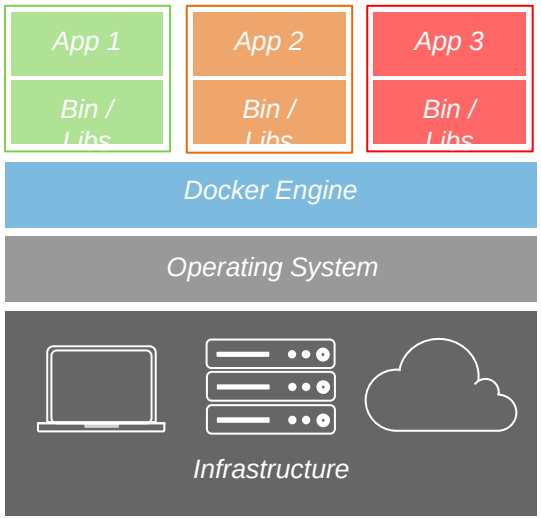
Docker is the current leading implementation of the virtualisation model based on software containers.

PORTABLE	*Originally made possible by the virtualisation extensions implemented in the Linux kernel (initially **lxc** and later **lib-container**), it is now supported also on Windows platforms, thus increasing its popularity and reach.*
IMAGE BASED	*Similarly to a VM, a docker container is based on the concept of "image", which is a built-in configuration of the software stack running in the container.*
DE FACTO STANDARD	*Characterised by a rich set of features to build software systems and a set of tools for their management, Docker has become the predominant container technology. Support for docker container is widely spread among both PaaS and IaaS vendor.*

Containers versus virtual machines



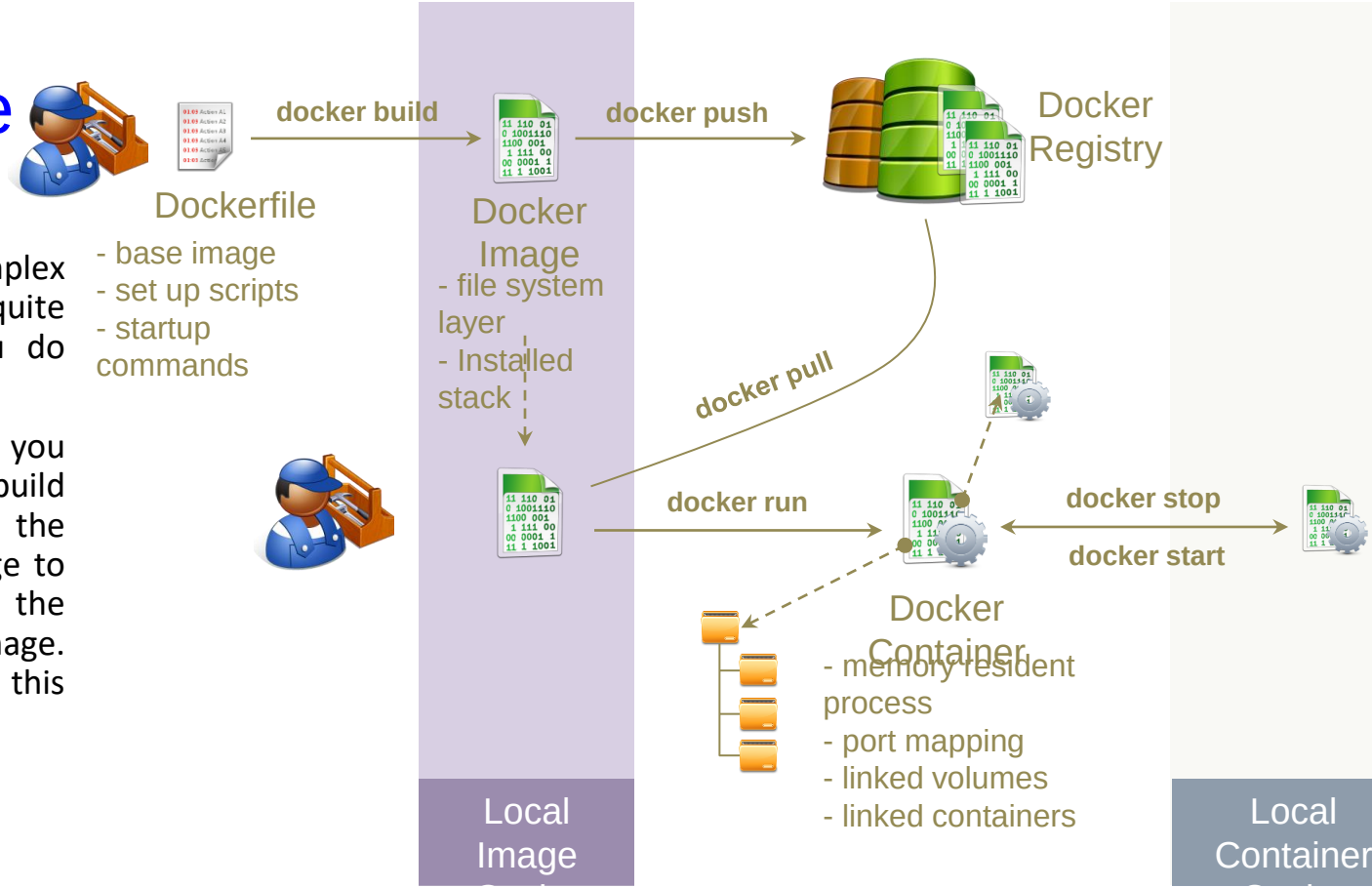
VIRTUAL MACHINES



DOCKER CONTAINERS

Containers allow us to easily manage multiple applications and saving resources. We can easily run tens of docker containers vs a few virtual images.

Docker Lifecycle



This may look more complex than it should, but it is quite straightforward once you do the process once.

From your program, you create a dockerfile, you build an image, you publish the image, you push the image to your repository, you pull the image and run the image. Easier done than said this time.

Install Docker

Docker Desktop for OSX or Windows is the best method to get started designing containerised applications. Docker Desktop makes it simple to instal Kubernetes or Swarm on your local development PC, allowing you to enjoy all of the orchestrator's functionality straight away, without the need for a cluster.

Follow the installation instructions appropriate for your operating system:

- OSX
- Windows
- Ubuntu

Important Terminologies in Docker

Docker Image: It's a multi-layered file that's used to run programmes within a Docker container. They're a collection of guidelines for building Docker containers.

Docker Container: It's an image instance created at runtime. Allows developers to package an application with all of the necessary components, including libraries and other dependencies.

Docker File: It's a text file that contains the commands that, when run, assist in the creation of a Docker image. A Docker file is used to construct a Docker image.

Docker Engine: Docker Engine is the programme that runs the containers. Docker Engine is a web-based client-server application.

- Server: It is in charge of developing and managing Docker images, containers, networks, and volumes. A daemon process is what it's called.
- Rest API: It defines how applications can communicate with the Server and tells it what to perform.
- Client: The Client is a Docker command-line interface (CLI) that allows us to interact with Docker through the use of Docker commands.

Docker Hub: Docker Hub is the official web repository where you may find additional Docker Images to utilise. It makes finding, managing, and sharing container photos with others a breeze.

Why Use Docker ?

Portability: We can ship not just our application code, but also our environment, thanks to the Dockerfile. Not only can we publish the app's source code to git, but we can also include the Dockerfile. When someone else pulls our repository, they can use the Dockerfile to construct the source code in a Docker container.

Version Control and CI/CD: We can keep track of changes in our Docker file, as stated in portability. This allows us to try out newer versions of our software. For example, we can build a branch in which we experiment with the most recent version of Python.

Isolation: When your code runs in a container, it has no impact on other programmes. It is a fully stand-alone system. If you've ever had unexpected failures in one of your scripts after updating a globally installed library, you'll be familiar with this issue.

Compose Container: The majority of the time, many containers must work together to create all of the functions of an application. For instance, a website, API, and database must all be linked together. Docker Compose enables us to accomplish just that. We can make a file that specifies how containers are linked to one other. This file may be used to create all of the Dockerfiles for all of our containers at once!

Section 2: Dockerise a NodeJs Application

So let try to create a docker file and docker image for the application we created in the previous module and see how we can utilise it to put our application on Docker Hub.

For this purpose the first thing to do is to create an account on Docker Hub where we will be hosting our public docker image of the application.

Creating an image

So now let's try to dockerise the application we created in our last module.

First we create two files in our project **Dockerfile** and **.dockerignore,** we put them in the root of our application.

The file .dockerignore would have the exact same contents that you have in your .gitignore file so you can simply copy all the contents of .gitignore and paste it in .dockerignore

Now let's write the contents of our Dockerfile.

Dockerfile

```
from node:16-alpine
WORKDIR /app
COPY . .
EXPOSE 8080
RUN npm install
CMD ["npm","start"]
```

First we tell the docker what OS we want to use here we are using **node:16-alpine** which is a node-based environment and only uses those systems of an OS that are needed to run a node application. We do this by adding the command **FROM node:16-alpine**

Next we tell our docker application that you need to set the working directory to /app where all our code would be copied. We do this by using the command **WORKDIR /app**

After that we use the command **COPY . .** , which basically copies all the code to the working directly ignoring all the things we asked our dockerignore to be ignore

Next we want to expose the port at which our application runs, So as our application run on port 8080 so we expose that port by saying **EXPOSE 8080**

Now that our code is copied we need to install the node modules on our Docker container so we run the command **RUN npm install**

And Lastly we want to tell our application the command we want to use to run the application. We do that by using the command **CMD[“npm”,”start”].** Thing to notice here is that each word in our run command is written in double quotes separated by a comma. That is a notation used by docker.

And that's all we have successfully written the docker file for our application. But that's not all we still need to build and run the file in order to create docker image and container.

Build the image

Now that we have written the docker file we need to create an image for docker to run.

So we go to our terminal and use this command to build an image.

```
$ docker build -t <image name> .
```

So for this example we are using the mvc-structure project so we write the command like this

```
$ docker build -t navitchoudhary22/mvc-structure .
```

In this command navitchoudhary22 is an example docker username. You should use your Docker Hub username in place of this username. Also we use the dot in the end so that we tell docker that you need to build the docker file in our current folder.

Check the images

Once the image has been completed we can see our image created by using the command

```
$ docker images
```

```
Navit@alessios-Mini mvc-structure % docker images
REPOSITORY                      TAG       IMAGE ID       CREATED          SIZE
navitchoudhary22/mvc-structure  latest    bb8d08be6adf   40 seconds ago   137MB
mysql                           5.7       11d8667108c2   4 weeks ago      450MB
mysql                           latest    17b062d639f4   7 weeks ago      519MB
adminer                         latest    52d7d4d01b51   7 weeks ago      81.9MB
confluentinc/cp-kafka           latest    5069d65bcc55   2 months ago     780MB
confluentinc/cp-zookeeper       latest    3a7ea656f1af   2 months ago     780MB
wurstmeister/kafka              latest    5a3eb4db42f0   3 months ago     508MB
wurstmeister/zookeeper          latest    3f43f72cb283   3 years ago      510MB
Navit@alessios-Mini mvc-structure %
```

You can see here an image was successfully created.

Now the only part left is to run our docker image.

Run the image

So in order to run our docker image created, we use the command

```
$ docker run -d -p 7000:8080 <image name>
```

```
$ docker run -d -p 7000:8080 navitchoudhary22/mvc-structure
```

When you see the command you can see that we used a different port instead of 8080, that is a good thing that docker gives us the ability that we can bind any available port to our application. So the first port is the port where we want the application to run and the second port is the port where the application runs internally and the one we exposed in our docker file. So we can give any port for first port like 4000,5000,8080 or even 80 just keep in mind that that port must not be used by something else already and the second port is always going to be 8080 as our application runs on 8080.

Once you have run that you can see the app running by using the command

```
$ docker ps -a
```

Add Image to Docker Hub

So in order to push our docker image to Docker Hub. We use the command

```
$ docker push <image name>
```

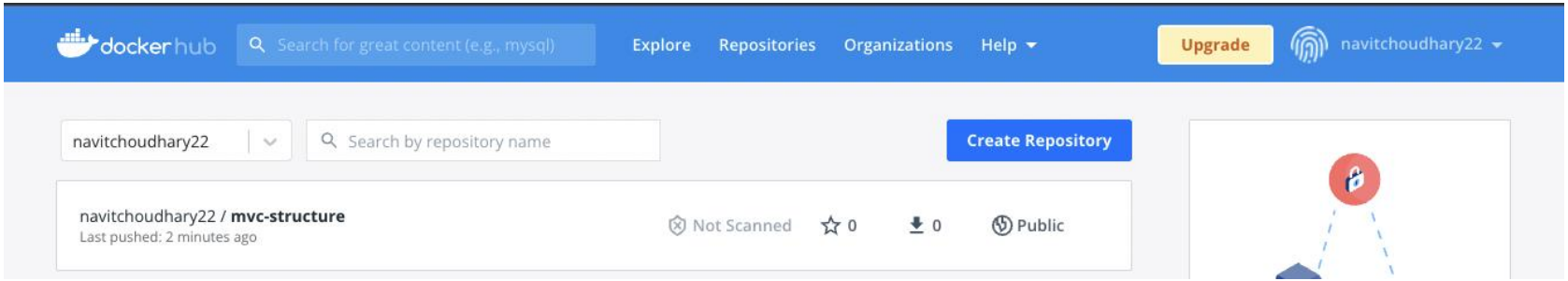
```
$ docker push navitchoudhary22/mvc-structure
```

This would create a public version of the app you created on Docker Hub in your account and then you can use that image to be pulled on any remote server and run that image on that remote server.

```
Navit@alessios-Mini mvc-structure % docker push navitchoudhary22/mvc-structure
Using default tag: latest
The push refers to repository [docker.io/navitchoudhary22/mvc-structure]
96fbcdf7f7d1: Pushed
f3a657b833b2: Pushed
2b4c85b01e66: Pushed
866364cccc74: Pushed
d93b1c055b05: Mounted from library/node
b09f522ed107: Mounted from library/node
433206fe79ca: Mounted from library/node
4f4ce317c6bb: Mounted from library/node
latest: digest: sha256:ebb6570a83e678ed636630b57e61001a74ca4c152da4897c20d28a689c5cff8f size: 1993
Navit@alessios-Mini mvc-structure %
```

Add Image to Docker Hub

You can also see the image was successfully added to Docker Hub.



In order to pull the image from Docker Hub all we need is to use the command

```
$ docker pull <image name>
```

```
$ docker pull navitchoudhary22/mvc-structure
```

So this was a very easy way to dockerise your application.
Try the application by simply visiting the address – in our case is localhost:5000 . If you push it to DockerHub, everyone will be able to run your application!

Some key points to keep in mind:

- Whenever you have made a change in your application and you want to see the changes on the docker version of your application you will have to redo all the steps
- First stop your docker container, then rebuild the docker image, and then run the docker container
- You can run as many containers you want, simply provide a different port!

You can refer to code here.

Exercise 1

Try creating your own DockerFile for your Nodejs application and try creating a running image of your application. Provide your public image link to the trainer and they should be able to run your application using your docker image.

Section 3: Introduction to GitHub Actions

You may use GitHub Actions to construct unique software development lifecycle workflows right in your GitHub repository. These workflows are made up of many activities, or actions, that can be executed automatically in response to particular triggers.

This allows you to incorporate features like as continuous integration (CI) and continuous deployment (CD) right in your repository.

Why care about GitHub Actions ?

Let's talk about why developers should care about GitHub Actions in the first place and what benefits they give before we dive into the technical details.

Built into GitHub: Because GitHub Actions is fully integrated into GitHub, it does not require a separate website. This means it can be managed alongside your other repository-related features like pull requests and bugs in the same place.

Multi-container testing: By adding support for Docker and docker-compose files to your workflow, you can test multi-container setups.

Multiple CI templates: GitHub offers a variety of templates for various CI (Continuous Integration) configurations, making it relatively simple to get started. You can also make your own templates, which you can then share on the GitHub Marketplace as an Action.

Why care about GitHub Actions ?

Great free plan: Every open-source repository receives free actions, which include 2000 free build minutes per month for all your private repositories, which is comparable to most CI/CD free plans. If that isn't enough for your needs, you can upgrade to a different plan or go self-hosted.

Core Concepts

The basic principles utilised in GitHub Actions are listed below, and you should be familiar with them before using it or reading the instructions.

Actions: Actions are the simplest and most portable building blocks of a process, and they can be combined to form a job. You can build your own Actions or use Actions from the Marketplace that have been posted publicly.

Events: Events are one-time occurrences that cause a process to run. When someone pushes to the repository or creates a pull request, for example, a workflow is initiated. Webhooks can be used to configure events to listen for external events.

Runner: A runner is a computer that runs the GitHub Actions runner programme. The runner then waits for jobs to become available, which it can subsequently execute. They run the task's actions after picking up a job and report the progress and outcomes to GitHub. Runners can be self-hosted on your own machines/servers or hosted on GitHub.

Core Concepts

Job: A job is made up of numerous phases that run in a virtual environment instance. Jobs can execute in parallel or in sequence, depending on whether the current job relies on the success of the prior one.

Step: A step is a set of tasks that a job can do. Steps can be used to execute instructions or operations.

Workflow: A Workflow is an automated procedure that can be initiated by an event and is made up of one or more jobs. Workflows are defined in the .github/workflows directory using a YAML file.

Section 4: Adding GitHub Actions to Application

So in order to add GitHub actions to our application and create a CI/CD pipeline where we can automatically push our latest image to the docker hub we need to set up a few things.

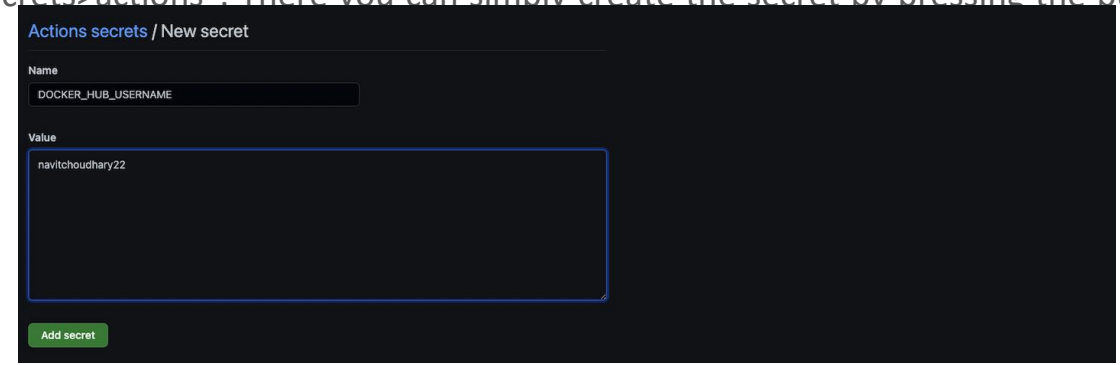
Pre-Requisites:

- Create a Docker Hub access token (Save the token somewhere safe as it would only be generated one time)
- Add Repository Secrets to your GitHub application

Adding Repository Secrets

Repository secrets are a place where you can store your secret keys in GitHub and then you can use them in your GitHub action workflows and being stored in repository secrets they are safe and outside people won't be able to access those secret keys.

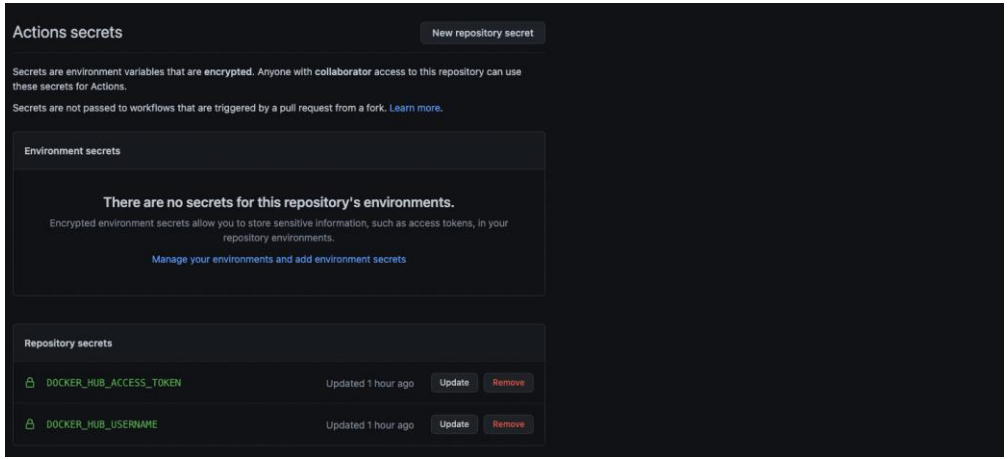
For creating secrets you can simply go to the settings section of your GitHub application and then from the side navigation go to “secrets>actions”. There you can simply create the secret by pressing the button “new repository secret”.



Adding Repository Secrets

You need to create two secrets.

- DOCKER_HUB_USERNAME
- DOCKER_HUB_ACCESS_TOKEN



Creating GitHub Action

So now let's try to add GitHub actions to the application we created in our last module.

First we create a folder called .github and inside that folder we create another folder called workflows. This is where our GitHub actions file will reside. Next we add a file called **cicd.yml** in our workflows folder.

Now let's write the contents of our **cicd.yml**

```yaml
name: CI/CD

on:
 push:
   branches: [ master ]

jobs:
 build:
   runs-on: ubuntu-latest
   strategy:
     matrix:
       node-version: [16.x]

   steps:
   - name: Checkout repository
     uses: actions/checkout@v2

   - name: Set up Node.js ${{ matrix.node-version }}
     uses: actions/setup-node@v1
     env:
       PUPPETEER_SKIP_CHROMIUM_DOWNLOAD: 'true'
     with:
       node-version: ${{ matrix.node-version }}

   - name: Install dependencies
     run: npm install

   - name: Login to Docker Hub
     uses: docker/login-action@v1
     with:
       username: ${{ secrets.DOCKER_HUB_USERNAME }}
       password: ${{ secrets.DOCKER_HUB_ACCESS_TOKEN }}

   - name: Set up Docker Buildx
     id: buildx
     uses: docker/setup-buildx-action@v1

   - name: Build and push
     id: docker_build
     uses: docker/build-push-action@v2
     with:
       context: ./
       file: ./Dockerfile
       push: true
       tags: ${{ secrets.DOCKER_HUB_USERNAME }}/mvc-structure:latest

   - name: Image digest
     run: echo ${{ steps.docker_build.outputs.digest }}
```

Creating GitHub Action

So let's try to understand what all we did in our cicd.yml file.

The **name** basically just tells the name of the whole workflow process. The second part will tell when to run the workflow which is mentioned as “on > push > master” telling us to activate this workflow whenever we push something in our master branch.

Next we mention the **Job** that needs to be carried out. We tell the workflow we are going to run a **build** job in this workflow (we can also tell the workflow to run multiple jobs one after another). The build job is divided into two parts the **environment (runs-on and strategy)** that it needs to build in and the **steps** that need to be carried out.

Creating GitHub Action

So let's see what is the **Environment** we set up for our application (part written under runs-on and strategy). So we mention two things here:

- Runs-on: which tell us the OS (ubuntu-latest)
- Strategy: which tells us the environment that needs to be installed (nodejs version)

Next we tell our workflow all the steps needed

- Checkout repository (so the GitHub actions knows the secrets)
- Set up Node Js version from the strategy matrix
- Install Dependencies (npm install)
- Login to Docker Hub (using the secrets we created earlier)
- Set up docker build (tells our environment that we would be creating a docker build)
- Create the build and push it to Docker Hub (this also picks the name of the docker account from secrets)
- Output the docker digest (so we know the build was successful)

Creating GitHub Action

So now whenever we push something to our master branch the automatic CI/CD pipeline we run and create a new build and push it to our docker hub. So that is a very simple CI/CD pipeline that you can create using GitHub actions. Refer to code here

Exercise 2

Try creating your own GitHub Actions for your Nodejs application and try creating a running CI/CD pipeline for your application.

End of Presentation