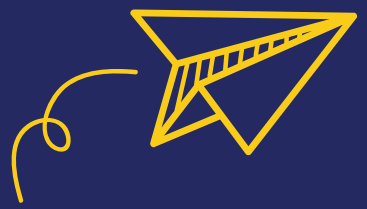


docker

WORKSHOP

SUPERNOVA

WHAT IS DOCKER?

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

Docker provides the ability to package and run an application in a loosely isolated environment called a container.

LET'S RUN A CONTAINER

You've been given a project to work on, this project requires Microsoft SQL Server. You need to get started right away and don't have time to wait for IT to setup a server for you. What do you do?

1. Install Docker
2. Run the commands: list images

```
docker pull mcr.microsoft.com/mssql/server:2022-latest
```

```
docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=yourStrong(!)Password" -p 1433:1433 -d mcr.
microsoft.com/mssql/server:2022-latest
```

NOTE: you may need to use single quotes if bash interprets the strings and complains about the exclamation mark

"docker pull" downloads the image you want to use, this can then be re-used each time you run and not have to download it again until it is updated.

"docker run" will run the container and allow you to use the app. -e is where environment variables are passed in, -p maps the port(s), -d run container in background, and the last part is the name of the image you are running.

You may now connect to the MS SQL using the username sa and password yourStrong(!)Password.

WHAT ABOUT CREATING YOUR OWN?

Let's containerize a boilerplate dotnet 6 web api. Make a working folder called "WeatherAPI" and in this folder run dotnet new webapi

Open the code in visual studio/visual studio code and in the root folder create an empty text file called Dockerfile with no extension. In this file paste:

```
# The base image we will be using, this is an image based on Alpine Linux and has aspnet 6 installed
FROM mcr.microsoft.com/dotnet/aspnet:6.0-alpine AS base
WORKDIR /app
# Lets docker know the app uses port 8080 which will need to be exposed
EXPOSE 8080

# Environment variables passed in to the app
ENV ASPNETCORE_URLS=http://  *:8080

# Our build image used for building our app, has the dotnet 6 sdk installed
FROM mcr.microsoft.com/dotnet/sdk:6.0-alpine AS build
WORKDIR "/src/WeatherAPI"
COPY "WeatherAPI.csproj" .
RUN dotnet restore "WeatherAPI.csproj"

# Copy from the source folder into the build image's working folder
COPY . .
RUN dotnet build "WeatherAPI.csproj" -c Release -o /app/build
```

```
# Publish image
FROM build AS publish
RUN dotnet publish "WeatherAPI.csproj" -c Release -o /app/publish

# Copy from the publish image into our final image
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "WeatherAPI.dll"]
```

From your command prompt/terminal run the following:

BUILD IMAGE

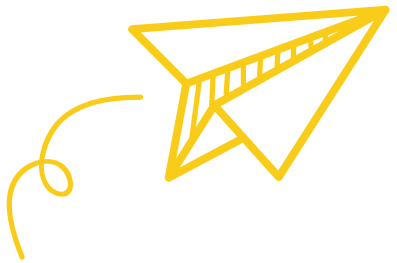
```
docker build -t nicholascookbet/weatherapi .
```

-t tags the image, in this case we tag the repository we want to use, replace the part before the / with your own docker hub id.

LIST IMAGES

```
docker image ls
```

You should see your created image in this list.



RUN YOUR IMAGE

```
docker run -p 8080:8080 nicholascookbet/weatherapi
```

In your browser open: http://localhost:8080/weatherforecast

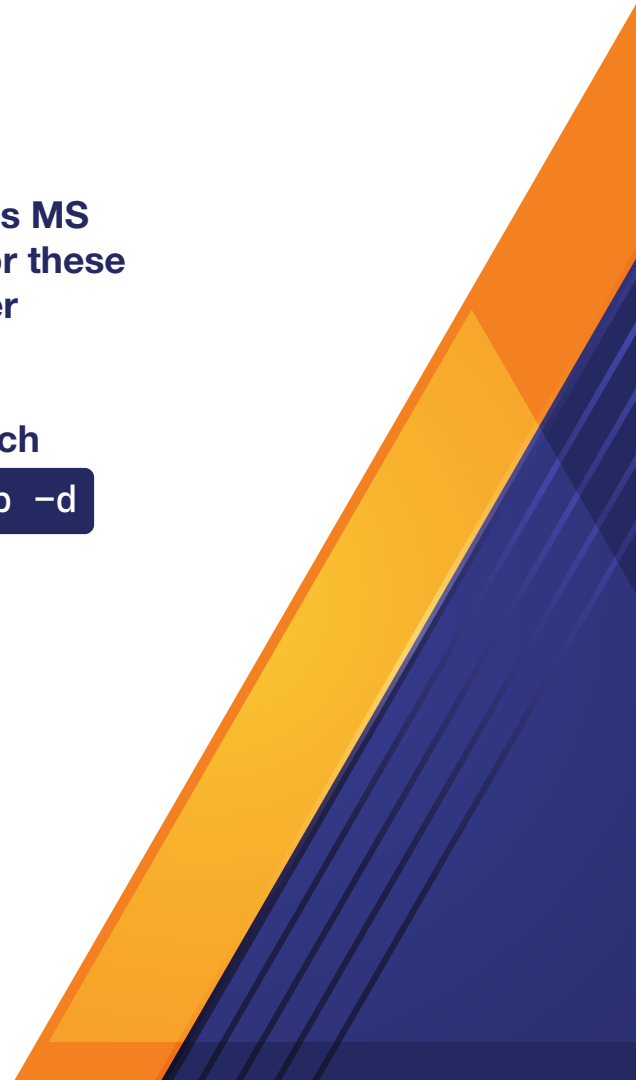
That's it!

IS THERE MORE?

Docker Compose

Docker Compose is a great tool for development. Let's say you are developing a project which requires MS SQL, CockroachDB & Kafka. Every day when you start development you will need to run docker run for these 3 images, and remember the settings needed. This is time consuming and painful, this is where Docker Compose comes in!

Docker Compose allows you to create a file, which you can safely store in your source code repo, which defines what you want. So all you have to do before starting development is run `docker-compose up -d`



```
version: "3.9"
services:
  zookeeper:
    image: 'bitnami/zookeeper:3.7'
    ports:
      - '2181:2181'
    environment:
      - ALLOW_ANONYMOUS_LOGIN=yes
    healthcheck:
      test:  "echo mntr | nc -w 2 -q 2 localhost 2181"
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - public

  kafka:
    hostname: kafka
    image: 'bitnami/kafka:3.0.0'
    ports:
      - '9092:9092'
      - '29092:29092'
    environment:
      - KAFKA_BROKER_ID=1
      - ALLOW_PLAINTEXT_LISTENER=yes
      - KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP=CLIENT:PLAINTEXT,EXTERNAL:PLAINTEXT
      - KAFKA_CFG_LISTENERS=CLIENT:  :9092,EXTERNAL:  :29092
      - KAFKA_CFG_ADVERTISED_LISTENERS=CLIENT:  kafka:9092,EXTERNAL:  localhost:29092
      - KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181
      - KAFKA_CFG_INTER_BROKER_LISTENER_NAME=CLIENT
```

```
healthcheck:
     test: "kafka-topics.sh --bootstrap-server kafka:9092 --list"
     interval: 10s
     timeout: 10s
     retries: 10
   depends_on:
     - zookeeper
   networks:
     - public

 cockroach:
   image: cockroachdb/cockroach:v22.1.3
   command: start-single-node --insecure
   container_name: cockroachdb
   hostname: cockroach_node_1
   depends_on:
     - kafka
   ports:
     - 26257:26257
     - 8081:8080
   mem_limit: 512m
   mem_reservation: 128M
   cpus: 0.5
   networks:
     - public
```

```
mssql:
    container_name: mssql
    image: mcr.microsoft.com/azure-sql-edge:latest
    hostname: mssql
    ports:
      - 1433:1433
    environment:
      SA_PASSWORD: "yourStrong(!)Password"
      ACCEPT_EULA: "Y"
    networks:
      - public

networks:
  public:
    driver: bridge
```

Run `docker-compose up -d` in the same folder as the file and watch the containers come to life.

Run docker ps to see the running containers. docker-compse down will stop them all.