Kubi

Groupe :
Lyne ARMAND
Dylan Hochenberger
Mathieu SOMMER

SOMMAIRE

01 Project

We have to deploy a e-commerce administration app in a Kubernetes cluster.

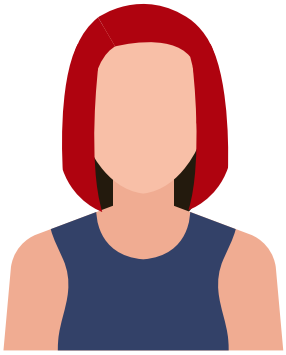
Through this app, a visitor can manage a catalog and search products by keywords.
It has been built in a microservice architecture, composed of:

- a frontend application in Angular,
- an API in Laravel (PHP),
- an indexer in Node.JS,
- a reporting job in Go,
- 3 databases (MySQL, RabbitMQ, Elasticsearch).

delivery method : Github
repository name : $CourseCode-$GroupName.git

Team 02

We are group 3.



Lyne ARMAND

Dylan HOCHBERGER

Matthieu SOMMER

03 Dockerizing frontend

Containerize a Angular application with Docker and NGINX

1. Creating a file .dockerignore

app > front > .dockerignore

```
.git
node_modules
npm-debug.log
dist
```

Stage often forgotten, but indispensable! If we do not use . dockerignore, we will send all the files in the context of Docker, significantly slowing down the build process.

2. Creating a Dockerfile

app > front > Dockerfile > ...

```
# Stage 1: Compile and Build angular codebase

# Use official node image as the base image
FROM node:8.9 as build

# Set the working directory
WORKDIR /usr/local/app

# Add the source code to app
COPY ./ /usr/local/app/

# Install all the dependencies
RUN npm install

# Generate the build of the application
RUN npm run build


# Stage 2: Serve app with nginx server

# Use official nginx image as the base image
FROM nginx:latest

# Copy the build output to replace the default nginx contents.
COPY --from=build /usr/local/app/dist/front /usr/share/nginx/html

# Expose port 80
EXPOSE 80
```

3. Build the image

Run the following command to build the Docker image. The -t flag lets you tag your image so it's easier to find later using the docker images command.

```
➜  CLO902-group4 git:(dev_lyne) ✗
➜  CLO902-group4 git:(dev_lyne) ✗ docker build -t lyne/front-angular  .
```

4. Run the image

Running the image with -d runs the container in detached mode, leaving the container running in the background. The -p flag redirects a public port to a private port inside the container.

```
➜  front git:(dev_lyne) ✗ docker run -d -p 8080:80 lyne/front-angular
```

Print the output of your app :

```
# Get container ID
$ docker ps

# Print app output
$ docker logs <container id>

# Example
Running on http://localhost:8080
```

5. Test

To test the app, get the port of app that Docker mapped.

```
➜  front git:(dev_lyne) ✗ curl -i localhost:8080
HTTP/1.1 200 OK
Server: nginx/1.21.6
Date: Fri, 06 May 2022 07:46:47 GMT
Content-Type: text/html
Content-Length: 970
Last-Modified: Thu, 05 May 2022 14:52:25 GMT
Connection: keep-alive
ETag: "6273e4a9-3ca"
Accept-Ranges: bytes

<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Front</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">

   <!-- Compiled and minified CSS -->
   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css">
   <!-- Compiled and minified JavaScript -->
   <script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js"></script>
   <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">

</head>
<body>
  <app-root></app-root>
<script type="text/javascript" src="runtime.js"></script><script type="text/javascript" src="polyfills.js"></script><script type="text/javascript" src="styles.js"></script><script type="
text/javascript" src="vendor.js"></script><script type="text/javascript" src="main.js"></script></body>
</html>
```

04 Dockerizing backend

```
FROM php:7.4-fpm

# Arguments defined in docker-compose.yml
ARG user
ARG uid

# Copy composer.lock and composer.json into the working directory
COPY composer.lock composer.json /var/www/

# Set working directory
WORKDIR /var/www/

# Install dependencies for the operating system software
RUN apt-get update && apt-get install -y \
    build-essential \
    libpng-dev \
    libjpeg62-turbo-dev \
    libfreetype6-dev \
    locales \
    zip \
    jpegoptim optipng pngquant gifsicle \
    vim \
    libzip-dev \
    unzip \
    git \
    libonig-dev \
    curl

# Clear cache
RUN apt-get clean && rm -rf /var/lib/apt/lists/*

# Install extensions for php
RUN docker-php-ext-install pdo_mysql mbstring zip exif pcntl bcmath
RUN docker-php-ext-configure gd --with-freetype --with-jpeg
RUN docker-php-ext-install gd

# Install composer (php package manager)
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer

# Copy existing application directory contents to the working directory
COPY . /var/www

# Create system user to run Composer and Artisan Commands
RUN useradd -G www-data,root -u $uid -d /home/$user $user
RUN mkdir -p /home/$user/.composer && \
    chown -R $user:$user /home/$user

USER $user

# Expose port 9000 and start php-fpm server (for FastCGI Process Manager)
EXPOSE 9000
CMD ["php-fpm"]
```

Dockerizing Databases 05

1. Elasticsearch

```
# https://github.com/elastic/elasticsearch-docker
FROM docker.elastic.co/elasticsearch/elasticsearch-oss:6.6.0

```

2. Kibana

```
# https://github.com/elastic/kibana-docker
FROM docker.elastic.co/kibana/kibana-oss:6.6.0
```

3. Logstah

```
# https://github.com/elastic/logstash-docker
FROM docker.elastic.co/logstash/logstash-oss:6.6.0

```

06 Dockerizing indexer

```
FROM node:fermium-alpine

WORKDIR /indexer

COPY package.json yarn.lock ./

RUN yarn install

COPY . .

CMD ["yarn", "start"]
```

07 Dockerizing reporting

```
FROM golang:1.16

RUN curl -sSfL https://raw.githubusercontent.com/cosmtrek/air/master/install.sh | sh -s -- -b $(go env GOPATH)/bin

WORKDIR /opt/app/api
COPY . .
CMD ["air"]
```

Docker compose 08

Compose is a tool for defining and running multi-container Docker applications.
So, we have 8 services :

- MySQL
- RabbitMQ
- Elasticsearch
- PHP
- webserver
- Frontend
- Indexer
- Reporting

1. MySQL Service

```yaml
#MySQL Service
db:
  image: mysql:5.7.32
  container_name: db
  restart: unless-stopped
  tty: true
  ports:
    - "3306:3306"
  environment:
    MYSQL_DATABASE: ${DB_DATABASE}
    MYSQL_ROOT_PASSWORD: ${DB_PASSWORD}
    MYSQL_PASSWORD: ${DB_PASSWORD}
    MYSQL_USER: ${DB_USERNAME}
    SERVICE_TAGS: dev
    SERVICE_NAME: mysql
  volumes:
    - dbdata:/var/lib/conf/mysql/
    - ./conf/mysql/my.cnf:/etc/conf/mysql/my.cnf
  networks:
    - app-network
  labels:
    kompose.service.type: nodeport
    kompose.service.expose: true
```

2. RabbitMQ Service

```yaml
#RabbitMQ Service
rabbitmq3:
  container_name: "rabbitmq"
  image: rabbitmq:3.10.2-management-alpine
  # environment:
  #   - RABBITMQ_DEFAULT_USER=myuser
  #   - RABBITMQ_DEFAULT_PASS=mypassword
  ports:
    # AMQP protocol port
    - "5672:5672"
    # HTTP management UI
    - "15672:15672"
  networks:
    - app-network
  volumes:
    - rabbitmq-data:/var/lib/rabbitmq
  healthcheck:
    test: rabbitmq-diagnostics -q check_running
    interval: 30s
    timeout: 30s
    retries: 5
    start_period: 10s
  labels:
    kompose.service.type: nodeport
    kompose.service.expose: true
```

3. Elasticsearch Service

```
#Elasticsearch service
elasticsearch:
  build:
    context: conf/elasticsearch/
  volumes:
    - ./conf/elasticsearch/config/elasticsearch.yml:/usr/share/conf/elasticsearch/config/elasticsearch.yml:ro
    - elasticsearch:/usr/share/conf/elasticsearch/data/:rw
  ports:
    - "127.0.0.1:9200:9200"
    - "127.0.0.1:9300:9300"
  environment:
    - discovery.type=single-node
    - "ES_JAVA_OPTS=-Xms256m -Xmx256m"
  networks:
    - app-network
  depends_on:
    - db
    # mysql:
    #   condition: service_started
  healthcheck:
    test:
      [
        "CMD-SHELL",
        "curl --silent --fail localhost:9200/_cluster/health || exit 1",
      ]
    interval: 30s
    timeout: 10s
    retries: 5
  image: docker.elastic.co/elasticsearch/elasticsearch:7.13.2
  labels:
    kompose.service.type: nodeport
    kompose.service.expose: true
```

4. PHP Service

```
#PHP Service
app:
  build:
    args:
      user: ${DB_USERNAME}
      uid: 1000
    context: ./back
    dockerfile: Dockerfile
  image: cloudsigma.com/php
  container_name: app
  restart: unless-stopped
  tty: true
  working_dir: /var/www/
  volumes:
    - ./back:/var/www/
    - ./back/php/laravel.ini:/usr/local/etc/php/conf.d/laravel.ini
  networks:
    - app-network
  environment:
    - SERVICE_NAME=app
    - SERVICE_TAGS=dev
    - APP_NAME=${APP_NAME}
    - APP_ENV=${APP_ENV}
    - APP_KEY=${APP_KEY}
    - APP_DEBUG=${APP_DEBUG}
    - APP_URL=${APP_URL}
    - LOG_CHANNEL=${LOG_CHANNEL}

    - DB_CONNECTION=${DB_CONNECTION}
    - DB_HOST=${DB_HOST}
    - DB_PORT=${DB_PORT}
    - DB_DATABASE=${DB_DATABASE}
    - DB_USERNAME=${DB_USERNAME}
    - DB_PASSWORD=${DB_PASSWORD}

    - BROADCAST_DRIVER=${BROADCAST_DRIVER}
    - CACHE_DRIVER=${CACHE_DRIVER}
    - QUEUE_CONNECTION=${QUEUE_CONNECTION}
    - SESSION_DRIVER=${SESSION_DRIVER}
    - SESSION_LIFETIME=${SESSION_LIFETIME}

    - RABBITMQ_HOST=${RABBITMQ_HOST}
    - RABBITMQ_PORT=${RABBITMQ_PORT}
    - RABBITMQ_USER=${RABBITMQ_USER}
    - RABBITMQ_PASSWORD=${RABBITMQ_PASSWORD}
    - RABBITMQ_VHOST=${RABBITMQ_VHOST}

    - ELASTICSEARCH_URI=${ELASTICSEARCH_URI}
    - ELASTICSEARCH_HOST=${ELASTICSEARCH_HOST}

  labels:
    kompose.service.type: nodeport
    kompose.service.expose: true
```

5. Webserver Service

```yaml
  webserver:
    image: nginx:1.17-alpine
    container_name: webserver
    restart: unless-stopped
    tty: true
    #depends_on:
    #  - app
    #  - frontend
    ports:
      - "8000:80"
      - "443:443"
    volumes:
      - ./back:/var/www/
      - ./nginx/conf.d/app.conf:/etc/nginx/conf.d/default.conf
    networks:
      - app-network
```

6. Frontend Service

```yaml
  # Frontend Service
  frontend:
    #depends_on:
    #- app
    #- rabbitmq3
    # backend:
    #   condition: service_started
    # rabbitmq3:
    #   condition: service_healthy
    build:
      context: front
      dockerfile: Dockerfile
    restart: always
    ports:
      - 3000:80
    networks:
      - app-network
    labels:
      kompose.service.type: nodeport
      kompose.service.expose: true
```

7. Indexer Service

```
  ### INDEXER ###
  indexer:
    depends_on:
      - app
      - rabbitmq3
      # backend:
      #   condition: service_started
      # rabbitmq3:
      #   condition: service_healthy
    build:
      context: indexer
      dockerfile: Dockerfile
    environment:
      - AMQP_URI=${RABBITMQ_URI}
      - ELASTICSEARCH_URI=${ELASTICSEARCH_URI}
    ports:
      - 8081:8081
    networks:
      - app-network
    labels:
      kompose.service.type: nodeport
      kompose.service.expose: true
```

8. Reporting Service

```
  ############################ REPORTING ############################

  reporting:
    depends_on:
      - app
      # backend:
      #   condition: service_started
    build:
      context: reporting
      dockerfile: Dockerfile
    environment:
      - DB_URI=${DB_USERNAME}:${DB_PASSWORD}@tcp(mysql:${DB_PORT})/${DB_DATABASE}
      - WEBHOOK_URL=${WEBHOOK_URL}

    ports:
      - 8888:8888
    networks:
      - app-network
    labels:
      kompose.service.type: nodeport
      kompose.service.expose: true
```

Registre Docker privé 09

Le Registry Docker est un système de stockage et de distribution d'image Docker open-source (sous la licence Apache), déployé côté serveur. Il permet aux utilisateurs d'extraire et insérer des images Docker dans un dépôt avec les autorisations d'accès appropriées. La même image peut avoir plusieurs versions différentes, identifiées par leurs tags.

1. Le stockage

Premièrement, on crée le dossier de stockage sur notre machine hôte:

```
mkdir data
```

2. Le chiffrement

On gère nos propres certificats à l'aide de l'outil openssl

```
openssl req \
    -newkey rsa:4096 -nodes -sha256 -keyout "$(pwd)"/certs/localhost.key \
    -x509 -days 365 -out "$(pwd)"/certs/localhost.crt
```

Openssl demande quelque information, on laisse les options par défaut en appuyant sur la touche entrée.
Il faut juste rentrer une adresse mail valide.

3. Restriction d'accès

On commence d'abord par créer un dossier auth pour stocker notre fichier :

```
mkdir auth
```

Par la suite on génère notre fichier htpasswd:

```
docker run --rm --entrypoint htpasswd registry:2.7.0 -Bbn testuser testpwd > "$(pwd)"/auth/htpasswd
```

4. Docker compose

Actuellement nous avons :

un volume pour stocker nos images envoyées par l'utilisateur
une communication chiffrée
un système d'authentification basique
Nous allons reprendre toutes ces fonctionnalités et les rajouter dans un Docker Compose :

```
version: "3.7"
services:
  registry:
    restart: always
    image: registry:2.7.1
    container_name: registryDocker
    ports:
      - 5000:5000
    environment:
      REGISTRY_HTTP_TLS_CERTIFICATE: /certs/localhost.crt
      REGISTRY_HTTP_TLS_KEY: /certs/localhost.key
      REGISTRY_AUTH: htpasswd
      REGISTRY_AUTH_HTPASSWD_PATH: /auth/htpasswd
      REGISTRY_AUTH_HTPASSWD_REALM: Registry Realm
    volumes:
      - "$(pwd)"/data:/var/lib/registry
      - "$(pwd)"/certs:/certs
      - "$(pwd)"/auth:/auth
```

5. Connexion à votre Docker Registry privé

```
docker login localhost:5000
```

Si tout se déroule comme prévu, on a le message suivant :

```
Login Succeeded
```

Une fois authentifié, vous pouvez alors envoyer votre image dans votre registry privé:

- Créer un nouveau tag de votre image

```
docker tag app_frontend localhost:5000/app_frontend
docker tag app_indexer localhost:5000/app_indexer
docker tag app_indexer localhost:5000/app_reporting
docker tag cloudsigma.com/php localhost:5000/cloudsigma
docker tag rabbitmq:3.10.2-management-alpine localhost:5000/rabbit
docker tag nginx:1.17-alpine localhost:5000/nginx
docker tag mysql:5.7.32 localhost:5000/mysql
docker tag docker.elastic.co/elasticsearch/elasticsearch:7.13.2 localhost:5000/elasticsearch
```

- Envoyer votre image vers le registry docker privé

```
docker push localhost:5000/app_frontend
docker push localhost:5000/app_indexer
docker push localhost:5000/app_reporting
docker push localhost:5000/cloudsigma
docker push localhost:5000/rabbit
docker push localhost:5000/nginx
docker push localhost:5000/mysql
docker push localhost:5000/elasticsearch
```

6. Récupérer les images depuis le registre docker privé

```
docker pull localhost:5000/app_frontend
docker pull localhost:5000/app_indexer
docker pull localhost:5000/app_reporting
docker pull localhost:5000/cloudsigma
docker pull localhost:5000/rabbit
docker pull localhost:5000/nginx
docker pull localhost:5000/mysql
docker pull localhost:5000/elasticsearch
```