MICROSERVICIOS
Trabajo Práctico: Docker

Mediante el uso de herramientas que se utilizan actualmente en los diferentes equipos de desarrollo, observar como podemos llevar a un próximo nivel los diferentes desarrollos que realizamos.

**OBJETIVOS**
**GENERALES**

- Comprender el funcionamiento de Docker, ventajas y desventajas.

**ESPECÍFICOS**

- Comprender el funcionamiento de un contenedor docker
- Utilizar diferentes imágenes para entender su funcionamiento en docker
- Conocer el funcionamiento de docker hub

**HERRAMIENTAS EMPLEADAS**

- Node.js
- Web Browser (Firefox, Chrome).
- Docker
- Express Framework
- Docker compose
- Mongodb
- Mongoose
- Terminal
- Editor de texto
- Swagger

**PROCEDIMIENTO**

1. Crear un nuevo proyecto node
    a. mkdir helloworld
    b. cd helloworld
    c. npm init
    d. Confirmar valores por defecto
    e. Npm creará package.json de forma automática

2. Agregar Express framework como dependencia
    a. npm install express --save

```
{
  "name": "helloworld",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

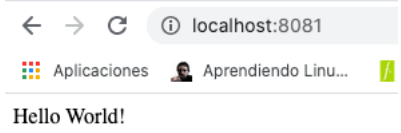
3. Agregar el archivo index.js
    a. touch index.js
    b. agregar el siguiente texto

```
//Load express module with `require` directive
var express = require('express')
var app = express()

//Define request response in root URL (/)
app.get('/', function (req, res) {
  res.send('Hello World!')
})

//Launch listening server on port 8081
app.listen(8081, function () {
  console.log('app listening on port 8081!')
})
```

4. Correr la aplicación
    a. node index.js
    b. abrir el navegador y probar: `http://localhost:8081/`




5. Instalar docker Siguiendo las instrucciones de la siguiente página

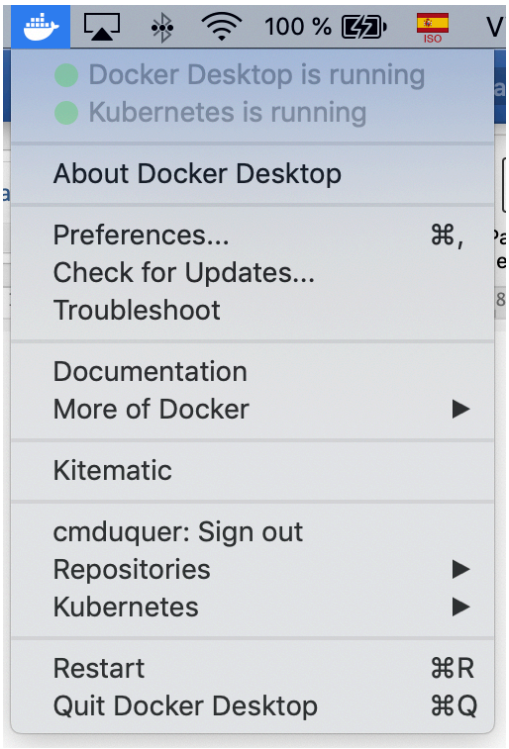
   https://docs.docker.com/docker-for-windows/install/#start-docker-for-windows

   https://docs.docker.com/docker-for-windows/install-windows-home/

https://docs.docker.com/engine/install/ubuntu/

6. Instala Kitematic a docker

   https://kitematic.com/




7. Verificar las imágenes de referencia para node de docker
   a. https://hub.docker.com/_/node

8. Crear archivo Dockerfile
   a. touch Dockerfile

```
# specify the node base image with your desired version node:<version>
FROM node:latest
# replace this with your application's default port
EXPOSE 8081
```

9. Establezca el directorio de trabajo en el contenedor a / app. Utilizaremos este directorio para almacenar archivos, ejecutar npm y ejecutar nuestra aplicación:

```
# specify the node base image with your desired version node:<version>
```

```
FROM node:latest
# replace this with your application's default port
EXPOSE 8081

WORKDIR /app
```

10. Copie la aplicación al directorio / app e instale dependencias. Si agrega primero el package.json y ejecuta npm install más tarde, Docker no tendrá que instalar las dependencias nuevamente si cambia el archivo package.json. Esto se debe a la forma en que se está creando la imagen de Docker (capas y caché), y esto es lo que debemos hacer:

```
# specify the node base image with your desired version node:<version>
FROM node:latest
# replace this with your application's default port
EXPOSE 8081

WORKDIR /app

COPY package.json /app
RUN npm install
COPY . /app
```

11. La última línea describe lo que debe ejecutarse cuando se inicia la imagen de Docker. Lo que queremos hacer es ejecutar nuestra aplicación:

```
# specify the node base image with your desired version node:<version>
FROM node:latest
# replace this with your application's default port
EXPOSE 8081

WORKDIR /app

COPY package.json /app
RUN npm install
COPY . /app

CMD node index.js
```

12. Construir la imagen de docker. El punto al final indica que se está trabajando en el contexto del directorio actual.
    a. `docker build -t your_dockerhub_username/helloworld .`
    b. `docker images`
    c. `docker run -p 8081:8081 your_dockerhub_username/helloworld`

13. Para subirlo a dockerhub creamos un tag de la siguiente forma
    a. `docker tag your_dockerhub_username/helloworld your_dockerhub_username/helloworld:0.0.1-SNAPSHOT`

b. `docker login`
c. `docker push your_dockerhub_username/helloworld`

14. Verificar que suba la imagen a docker hub
    a. https://hub.docker.com/r/cmduquer/helloworld/
15. Descargar la imagen de docker hub y hacerla correr en un contenedor local
    a. `docker run -p 8081:8081 cmduquer/helloworld:0.0.1-SNAPSHOT`
16. Verificar los procesos que hay corriendo en docker

```
MacBook-Pro-de-Carlos-2:helloworld cmduquer$ docker ps
CONTAINER ID        IMAGE                                COMMAND                  CREATED             STATUS              PORTS                    NAMES
824ec0fb8962        cmduquer/helloworld:0.0.1-SNAPSHOT   "docker-entrypoint.s…"   8 seconds ago       Up 7 seconds        0.0.0.0:8081->8081/tcp   flamboyant_franklin
MacBook-Pro-de-Carlos-2:helloworld cmduquer$
```

17. Agregar nodemon
    a. `npm i -g nodemon`
    b. `nodemon -L index.js`
18. Agregar swagger al proyecto
    a. https://www.npmjs.com/package/swagger-ui-express
    b. `npm i swagger-ui-express`
    c. `npm i swagger-jsdoc`
19. Agregar lo siguiente al index .js

```
const swaggerJsDoc = require("swagger-jsdoc");
const swaggerUi = require("swagger-ui-express");
// Extended: https://swagger.io/specification/#infoObject
const swaggerOptions = {
    swaggerDefinition: {
       info: {
        title: "HelloWorld API",
        description: "Hello World Class",
        contact: {
          name: "cmduquer"
        },
        servers: ["http://localhost:8081"]
      }
    },
apis: ["index.js"]
};

const swaggerDocs = swaggerJsDoc(swaggerOptions);
app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerDocs));
```

20. Agregar las siguientes rutas nuevas al index.js

```
/**
 * @swagger
 * /customers:
 *  get:
 *    description: Use to request all customers
 *    responses:
 *      '200':
 *        description: A successful response
 */
app.get('/customers', function (req, res) {
    res.status(200).send('Customer results')
  })

/**
 * @swagger
 * /customers:
 *  put:
 *    description: Use to update a customers
 *    parameters:
 *      - name: customer
 *        in: query
 *        description: Name of our customer
 *        required: false
 *        schema:
 *          type: string
 *          format: string
 *    responses:
 *      '201':
 *        description: A successful response
 */
app.put('/customers', function (req, res) {
    res.status(201).send('Successfully updated customer')
  })
```

21. Probar con swagger
    a. http://localhost:8081/api-docs/

22. Ejemplos de anotaciones con swagger jsdoc
    https://github.com/Surnet/swagger-jsdoc/blob/master/docs/GETTING-STARTED.md

```
/**
 * @swagger
 * /api/puppies:
```

```
 *   get:
 *     tags:
 *       - Puppies
 *     description: Returns all puppies
 *     produces:
 *       - application/json
 *     responses:
 *       200:
 *         description: An array of puppies
 *         schema:
 *           $ref: '#/definitions/Puppy'
 */
router.get('/api/puppies', db.getAllPuppies);
/**
 * @swagger
 * /api/puppies/{id}:
 *   get:
 *     tags:
 *       - Puppies
 *     description: Returns a single puppy
 *     produces:
 *       - application/json
 *     parameters:
 *       - name: id
 *         description: Puppy's id
 *         in: path
```

```
 *         required: true
 *         type: integer
 *     responses:
 *       200:
 *         description: A single puppy
 *         schema:
 *           $ref: '#/definitions/Puppy'
 */
/**
 * @swagger
 * /api/puppies:
 *   post:
 *     tags:
 *       - Puppies
 *     description: Creates a new puppy
 *     produces:
 *       - application/json
 *     parameters:
 *       - name: puppy
 *         description: Puppy object
 *         in: body
 *         required: true
 *         schema:
 *           $ref: '#/definitions/Puppy'
 *     responses:
 *       200:
```

```
 *         description: Successfully created
 */
/**
 * @swagger
 * /api/puppies/{id}:
 *   put:
 *     tags: Puppies
 *     description: Updates a single puppy
 *     produces: application/json
 *     parameters:
 *       name: puppy
 *       in: body
 *       description: Fields for the Puppy resource
 *       schema:
 *         type: array
 *         $ref: '#/definitions/Puppy'
 *     responses:
 *       200:
 *         description: Successfully updated
 */
/**
 * @swagger
 * /api/puppies/{id}:
 *   delete:
 *     tags:
 *       - Puppies
```

```
 *     description: Deletes a single puppy
 *     produces:
 *       - application/json
 *     parameters:
 *       - name: id
 *         description: Puppy's id
 *         in: path
 *         required: true
 *         type: integer
 *     responses:
 *       200:
 *         description: Successfully deleted
 */
```

23. Crear una nueva imagen de el proyecto siguiendo los pasos anteriores y subirla a docker hub.
24. Instalar docker compose
    a. https://docs.docker.com/compose/install/

25. Crear archivo docker-compose
    a. `touch docker-compose.yml`

```yaml
version: "1"
services:
  app:
    container_name: app
    restart: always
    build: .
    ports:
      - "8081:8081"
    links:
      - mongo
  mongo:
    container_name: mongo
    image: mongo
```

```
    ports:
      - "27018:27017"
```

a. `npm i mongoose`
b. `Modificar index.js`

```
//Load express module with `require` directive
const express = require('express')
const app = express()
const mongoose = require('mongoose')

const port = process.env.PORT || 8081;

const db_link = "mongodb://mongo:27017/helloworlddb";

const options = {
    useNewUrlParser: true,
    useUnifiedTopology: true
  };

  mongoose.connect(db_link, options).then( function() {
    console.log('MongoDB is connected');
  })
    .catch( function(err) {
    console.log(err);
  });

const swaggerJsDoc = require("swagger-jsdoc");
const swaggerUi = require("swagger-ui-express");
// Extended: https://swagger.io/specification/#infoObject
const swaggerOptions = {
    swaggerDefinition: {
      info: {
        title: "HelloWorld API",
        description: "Hello World Class",
        contact: {
          name: "cmduquer"
        },
        servers: ["http://localhost:8081"]
      }
    },
    apis: ["index.js"]
  };

  const swaggerDocs = swaggerJsDoc(swaggerOptions);
  app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerDocs));
```

```
//Define request response in root URL (/)
app.get('/', function (req, res) {
  res.send('Hello World!')
})


/**
 * @swagger
 * /customers:
 *  get:
 *    description: Use to request all customers
 *    responses:
 *      '200':
 *        description: A successful response
 */
app.get('/customers', function (req, res) {
    res.status(200).send('Customer results')
  })

/**
 * @swagger
 * /customers:
 *  put:
 *    description: Use to update a customers
 *    parameters:
 *      - name: customer
 *        in: query
 *        description: Name of our customer
 *        required: false
 *        schema:
 *          type: string
 *          format: string
 *    responses:
 *      '201':
 *        description: A successful response
 */
app.put('/customers', function (req, res) {
    res.status(201).send('Successfully updated customer')
  })



//Launch listening server on port 8081
app.listen(port, function () {
  console.log('app listening on port 8081!')
})
```

c. `Docker-compose build`

d. `docker-compose up`

Ejercicio

- Crear un api utilizando, nodejs, express, mongodb, swagger.
- El api debe contener el crud para la creación de usuarios de una aplicación móvil.
- Los atributos de la entidad serán a elección de cada uno.
- Entregar el repositorio git con las fuentes del proyecto y el archivo docker compose.
- Entregar el enlace de dockerhub con la imagen creada.
- Utilizar los códigos de respuesta correcta para operación con éxito y para operación con error.
- La api debe estar documentada usando swagger.
- Deben utilizar al menos los cuatros métodos del http:
    - Post
    - Get
    - Put
    - Delete