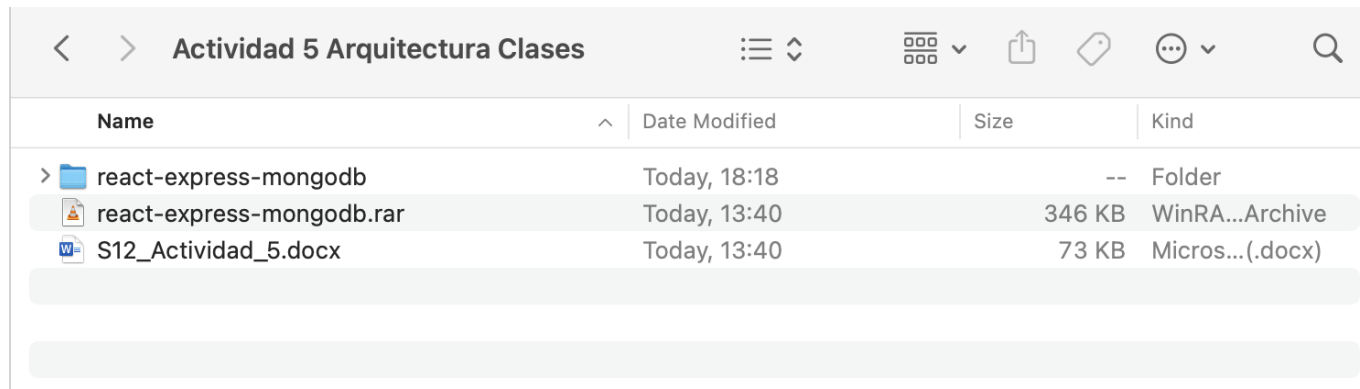


MongoDB en docker

Joaquín Leal - joleal@alumnos.uai.cl

Primer paso es extraer el archivo que se ha subido a webcursos, lo cual nos queda unicamente una carpeta.



Name	Date Modified	Size	Kind
> react-express-mongodb	Today, 18:18	--	Folder
react-express-mongodb.rar	Today, 13:40	346 KB	WinRA...Archive
S12_Actividad_5.docx	Today, 13:40	73 KB	Micros...(.docx)

Esto puede realizarse con winrar o el descompresor por excelencia que se utilice.

Luego, ingresaremos a la carpeta. Antes de utilizar docker, debemos arreglar/agregar algunas lineas de codigo dentro de nuestro archivo `compose.yaml`.

```
services:
  frontend:
    build:
      context: frontend
      target: development
    ports:
      - 3000:3000
    stdin_open: true
    volumes:
      - ./frontend:/usr/src/app
      - /usr/src/app/node_modules
    restart: always
    networks:
      - react-express
    depends_on:
      - backend
  backend:
    restart: always
    build:
      context: backend
      target: development
    volumes:
      - ./backend:/usr/src/app
      - /usr/src/app/node_modules
    depends_on:
      - mongo
    networks:
      - express-mongo
```

```
    - react-express
  expose:
    - 3000
  mongo:
    restart: always
    image: mongo:4.2.0
    volumes:
      - ./data:/data/db
    networks:
      - express-mongo
  expose:
    - 27017
  networks:
    react-express:
    express-mongo:
```

Archivo .yaml entregado en webcursos

Primero, debemos especificar la versión de docker que vamos a utilizar, el siguiente código se ingresa al inicio de nuestro .yaml

```
version: '3'
```

Luego, debemos ser un poco más precisos con los atributos que le estamos entregando al servicio de **mongo**.

Iniciaremos con agregar un atributo dentro de mongo el cual especificará los puertos que utilizará la aplicación.

```
ports:
  - '27017:27017'
```

Los números a la izquierda de **:** son los puertos físicos de nuestro ordenador mientras que los que se encuentran a la derecha son los puertos virtuales de docker. Estos puertos se comunicarán entre sí, es decir que lo que reciba el puerto físico se comunicará al puerto virtual.

Esto sucede dado que son máquinas virtuales, son como dos ordenadores distintos, estos deben comunicarse, con la distinción que son uno solo 🤪

Además debemos especificar como queremos que se llame el contenedor que vamos a utilizar, con ello crearemos:

```
container_name: mongo
```

Disculpen la creatividad

Otro atributo a modificar es la versión de mongo que vamos a utilizar, si bien es una buena practica usar unicamente una versión dado que al actualizar se pueden deprecir funciones o atributos, al ser unicamente una versión de prueba la eliminaremos, asi se modifica el atributo `image` por lo siguiente:

```
image: mongo
```

asi utilizaremos la última versión por default.

La última modificación dentro del servicio mongo será `volumes:`, esta es mas que nada una sutileza por estar trabajando en un ambiente MacOS, asi pues:

```
volumes:
  - ./datadir:/data/db
```

Cambiaremos el sevicio a editar, nos moveremos al servicio, **no atributo**, de `networks:` que se encuentra al final de nuestro archivo .yaml y agregaremos el formato en el cual queremos que se comuniquen los frameworks de express:

```
networks:
  react-express:
    driver: bridge
  express-mongo:
    driver: bridge
```

Al especificar el atributo `driver: bridge` estamos diciendo que se comuniquen mediante una conexión de puente entre el host y el contenedor, esta conexión es bidireccional.

Asi pues, con todas las modificaciones realizadas el archivo presente es el siguiente:

```
version: '3'
services:
  frontend:
    build:
      context: frontend
      target: development
    ports:
      - 3000:3000
    stdin_open: true
    volumes:
      - ./frontend:/usr/src/app
      - /usr/src/app/node_modules
    restart: always
    networks:
      - react-express
    depends_on:
```

```
    - backend
backend:
  restart: always
  build:
    context: backend
    target: development
  volumes:
    - ./backend:/usr/src/app
    - /usr/src/app/node_modules
  depends_on:
    - mongo
  networks:
    - express-mongo
    - react-express
  expose:
    - 3000
mongo:
  restart: always
  image: mongo
  ports:
    - '27017:27017'
  container_name: mongo
  volumes:
    - ./datadir:/data/db
  networks:
    - express-mongo
  expose:
    - 27017
networks:
  react-express:
    driver: bridge
  express-mongo:
    driver: bridge
```

Ahora queda descargar las imagenes de nuestro orquestador, para ello realizamos el siguiente comando en nuestra consola:

```
docker-compose build
```

con aquel comando descargamos todas las imagenes que esten especificadas en el `compose.yaml`

Lo cual debera verse de manera similar a la siguiente imagen:

```

macbookpro@Joaquins-MacBook-Pro react-express-mongodb % docker-compose build
[+] Building 4.2s (17/17) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 87B                                              0.0s
=> [internal] load .dockerignore                                                0.0s
=> => transferring context: 89B                                                0.0s
=> resolve image config for docker.io/docker/dockerfile:1.4                  2.2s
=> [auth] docker/dockerfile:pull token for registry-1.docker.io              0.0s
=> CACHED docker-image://docker.io/docker/dockerfile:1.4@sha256:9ba7531bd80fb0a858632727cf7a 0.0s
=> [internal] load .dockerignore                                                0.0s
=> [internal] load build definition from Dockerfile                            0.0s
=> [internal] load metadata for docker.io/library/node:lts-buster-slim        1.5s
=> [auth] library/node:pull token for registry-1.docker.io                  0.0s
=> [development 1/6] FROM docker.io/library/node:lts-buster-slim@sha256:d99a40755da4a6bd5545 0.0s
=> [internal] load build context                                                0.0s
=> => transferring context: 2.01kB                                             0.0s
=> CACHED [development 2/6] WORKDIR /usr/src/app                             0.0s
=> CACHED [development 3/6] COPY package.json /usr/src/app/package.json       0.0s
=> CACHED [development 4/6] COPY package-lock.json /usr/src/app/package-lock.json 0.0s
=> CACHED [development 5/6] RUN npm ci                                         0.0s
=> CACHED [development 6/6] COPY . /usr/src/app                               0.0s
=> exporting to image                                                         0.0s
=> => exporting layers                                                         0.0s
=> => writing image sha256:36574828d14e77cd2b63d31d60a2f087ffa039b01cceb40bc6d40cef6c8c7e4 0.0s
=> => naming to docker.io/library/react-express-mongodb-backend              0.0s
[+] Building 2.1s (15/15) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 87B                                              0.0s
=> [internal] load .dockerignore                                                0.0s
=> => transferring context: 89B                                                0.0s
=> resolve image config for docker.io/docker/dockerfile:1.4                  0.7s
=> CACHED docker-image://docker.io/docker/dockerfile:1.4@sha256:9ba7531bd80fb0a858632727cf7a 0.0s
=> [internal] load build definition from Dockerfile                            0.0s
=> [internal] load .dockerignore                                                0.0s
=> [internal] load metadata for docker.io/library/node:lts-buster            0.6s
=> [development 1/6] FROM docker.io/library/node:lts-buster@sha256:df5a66ed15950f6933d438198 0.0s
=> [internal] load build context                                                0.0s
=> => transferring context: 2.24kB                                             0.0s
=> CACHED [development 2/6] WORKDIR /usr/src/app                             0.0s

```

ahora debemos crear nuestros contenedores con:

```
docker-compose up
```






















esto utilizará las imagenes que descargamos previamente y las utilizara en un pequeño contenedor que le estamos asignando. el output por consola deberá ser similar al siguiente:

```

macbookpro@Joaquins-MacBook-Pro react-express-mongodb % docker-compose up
[+] Running 5/5
:: Network react-express-mongodb-react-express Created                                0.1s
:: Network react-express-mongodb-express-mongo Created                             0.1s
:: Container mongo Created                                                         0.3s
:: Container react-express-mongodb-backend-1 Created                             2.8s
:: Container react-express-mongodb-frontend-1 Created                            15.4s
Attaching to mongo, react-express-mongodb-backend-1, react-express-mongodb-frontend-1

```

este cambio tambien podra ser visualizado desde la aplicación de docker en la sección de contenedores.

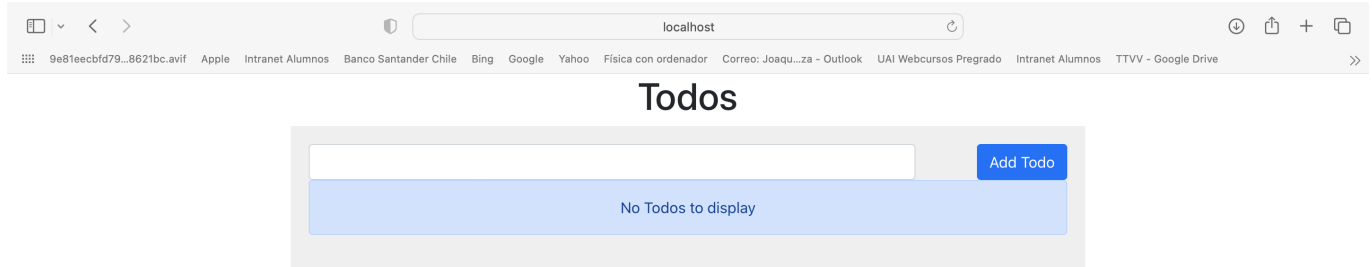
<input type="checkbox"/>	<div><div>✓</div></div>	react-express-mongodb	-	Running (3/3)	2 minutes ago				
<input type="checkbox"/>	<div></div>	mongo 6fcd470d2619 	mongo	Running	27017:27017 	2 minutes ago			
<input type="checkbox"/>	<div></div>	backend-1 40297dc79203 	react-express-mongodb-back	Running		2 minutes ago			
<input type="checkbox"/>	<div></div>	frontend-1 f22f651ca1ec 	react-express-mongodb-front	Running	3000:3000 	2 minutes ago			

Como pueden ver, tenemos un frontend, un backend y nuestra aplicación de mongo corriendo.

Ahora bien, si quisieramos acceder a la nuestra aplicación debemos acceder mediante nuestra aplicación de frontend que esta hosteada en:

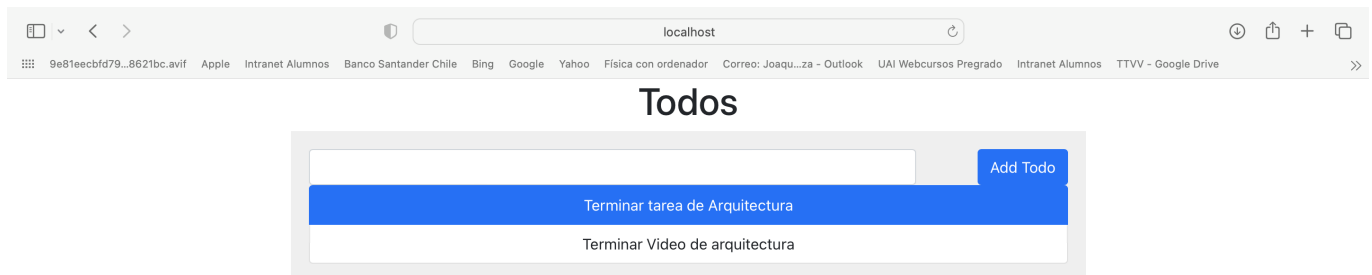
```
localhost:3000
```

Si ingresamos a esta url en el navegador, nos conectaremos al frontend de nuestra aplicación. así estaremos visualizando lo siguiente:



Así es, nuestra aplicación de tareas

Al ser una aplicación de Tareas por realizar podemos ingresar elementos en la barra del y estos se agregaran a nuestra base de datos controlada por MongoDB:



Ejemplo de la aplicación funcionando.

Además podemos acceder a los datos mediante **Mongo Compass**, como se ha registrado en la tarea anterior.

Por último para cerrar la app, debemos escribir en nuestra consola:

```
docker-compose down

[macbookpro@Joaquins-MacBook-Pro react-express-mongodb % docker-compose down ]
[+] Running 5/5
  ⚙ Container react-express-mongodb-frontend-1   Removed      1.4s
  ⚙ Container react-express-mongodb-backend-1    Removed      1.2s
  ⚙ Container mongo                               Remov...     0.8s
  ⚙ Network react-express-mongodb_express-mongo  Removed      0.3s
  ⚙ Network react-express-mongodb_react-express  Removed      0.2s
macbookpro@Joaquins-MacBook-Pro react-express-mongodb %
```

Lo cual eliminará los contenedores que acabos de crear liberando espacio y recursos, podemos confirmarlo verificando dentro de nuestra aplicación de docker:

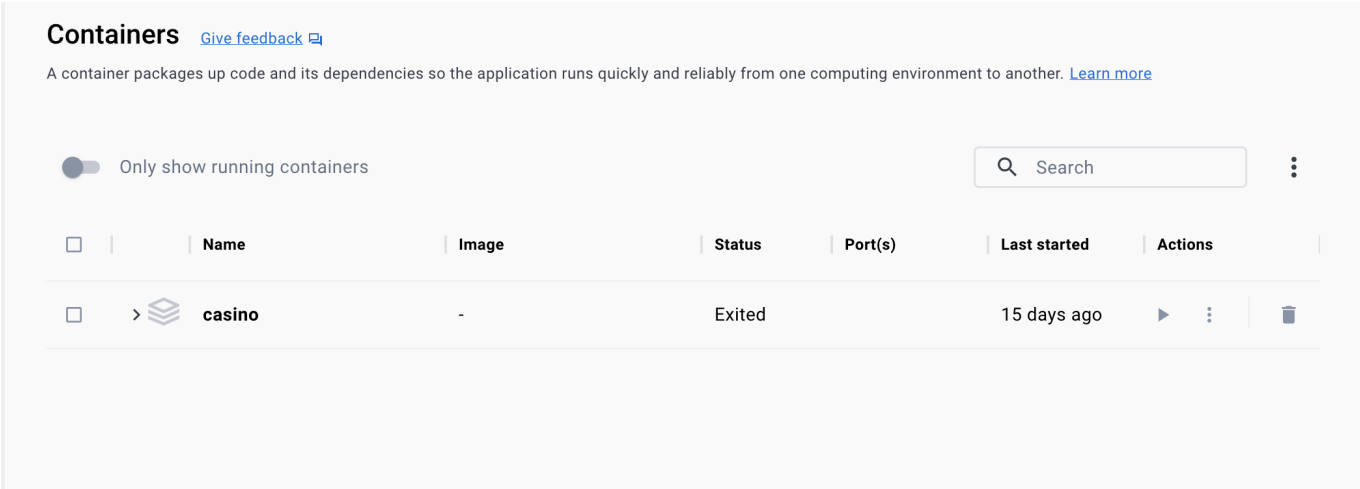


Imagen de muestra, no existen los contenedores.

Para referencias existe el video de Youtube que explica exactamente los mismos pasos:

- [MongoDB en docker](#)