



Universidad Austral de Chile

Conocimiento y Naturaleza

Sistema de egresados

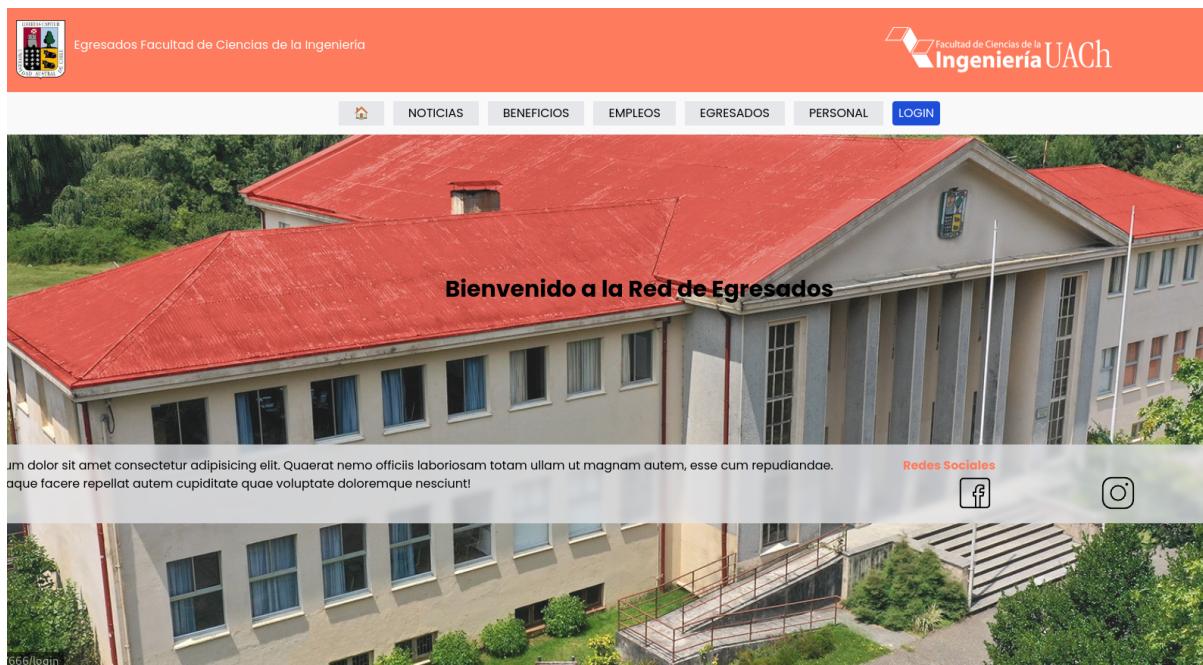
Manual de despliegue

INFO-248 Ingeniería de software

Integrantes:

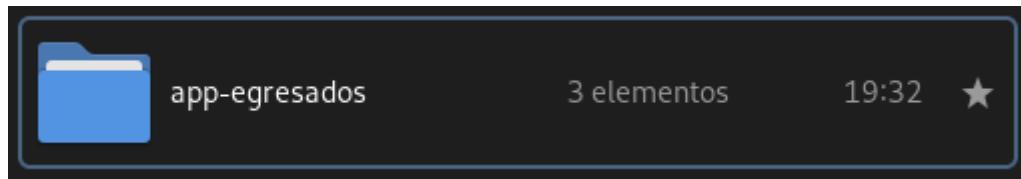
- Gerson Andrade Meza
- Jorge Andrade
- Manuel Care
- Geovanni Curguan.
- José Nicolas Aillapi Gomez
- Alex Garnica Hernández
- Diego Hinrichs
- Juan Pablo Pezo
- Jhonatan Friz
- Estefano Espinoza

Carrera: Ingeniería Civil Informática

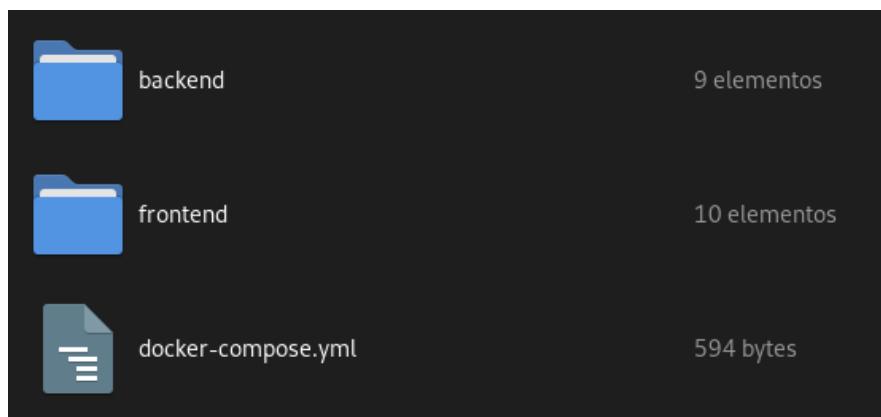


Preparación aplicación

Para poder despegar la aplicación en un servidor remoto con la ayuda de docker, necesitamos la aplicación. (Link del repositorio en anexo)



Dentro de este directorio debemos tener una estructura similar a la siguiente:



El archivo `docker-compose.yml`, se definirán los servicios que se levantarán en el servidor remoto.

```
version: '3.5'

services:
  mongodb-egresados:
    image: mongo:4.4.18
    container_name: mongodb
    restart: always
    ports:
      - 7668:27017
```

Establecemos la versión de la sintaxis del archivo de composición de Docker. En este caso, se utiliza la **'3.5'**.

Declaramos el servicio o contenedor **mongodb-egresados**, en este caso utilizamos la **imagen: mongo:4.4.18**, con **restart: always**, el contenedor se reiniciará automáticamente siempre que se inicie Docker. Finalmente mapeamos el puerto **7668** del host (servidor remoto) al puerto **27017**.

```
version: '3.5'

services:
  ...
  back-egresados:
    build:
      context: ./backend
      dockerfile: Dockerfile
      container_name: back-egresados
  ...
```

Similar a lo anterior construimos el contenedor del backend, le asignamos el nombre **back-egresados**, construimos utilizando como **imagen** el contexto local: **context: ./backend**. Se construirá utilizando el archivo **Dockerfile** que se encuentra dentro del contexto.

```
# Dockerfile back-egresados
FROM node:18

WORKDIR /app

COPY package*.json .

RUN npm install

COPY . .
```

```
EXPOSE 7667

RUN npm run build
CMD [ "npm", "run", "start" ]
```

Se establece como imagen de referencia `node:18` (node versión 18), se crea un directorio de trabajo en la raíz del contenedor `WORKDIR /app`, seguido copiamos en el contenedor todos los archivos que coincidan con `package*.json` (`package.json` y `package-lock.json`) y se instalan las dependencias del backend dentro del directorio `/app` en el contenedor con el comando `npm install`. Copiamos el resto de archivos dentro de `/app` (src, public, etc), se expone el puerto `7667` del contenedor y finalmente construimos y ejecutamos el proyecto, con `npm run build` y `npm run start`.

```
version: '3.5'

services:
...
back-egresados:
  build:
    context: ./API
    dockerfile: Dockerfile
  container_name: back-egresados
  restart: always
  depends_on:
    - mongodb-egresados
  ports:
    - 7667:7667
...
...
```

Al igual que el contenedor de `mongo`, el container del `backend` se iniciara cada vez que Docker lo haga, se agrega la dependencia al contenedor de la base de datos, `mongodb-egresados` y se mapea el puerto `7667` del host (servidor remoto) la puerto `7667` del contenedor. Con la dependencia se logra que primero se debe iniciar el contenedor de la base de datos antes que el backend.

Declaramos el ultimo contenedor que debe ser creado el cual tiene como dependencia el contenedor del backend, similar al anterior se construye utilizando el contexto local `./frontend` y un `Dockerfile` similar al anterior.

```
version: '3.5'
```

```
services:
  mongodb-egresados:
    ...
  back-egresados:
    ...
  front-egresados:
    build:
      context: ./frontend
      dockerfile: Dockerfile
      container_name: front-egresados
    restart: always
    depends_on:
      - back-egresados
    ports:
      - 7666:7666
```

Se realiza un mapeo del puerto `7666` host al `7666` servidor remoto.

```
# Dockerfile front-egresados
FROM node:18

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 7666

RUN npm run build
CMD [ "npm", "run", "start" ]
```

Adicionalmente y para evitar subir archivos no deseados propios del proceso de desarrollo, en cada contexto backend y frontend, se crean deben crear dos archivos `.dockerignore`. En el caso actual los archivos `.dockerignore` para backend y frontend serán los siguientes:

Backend

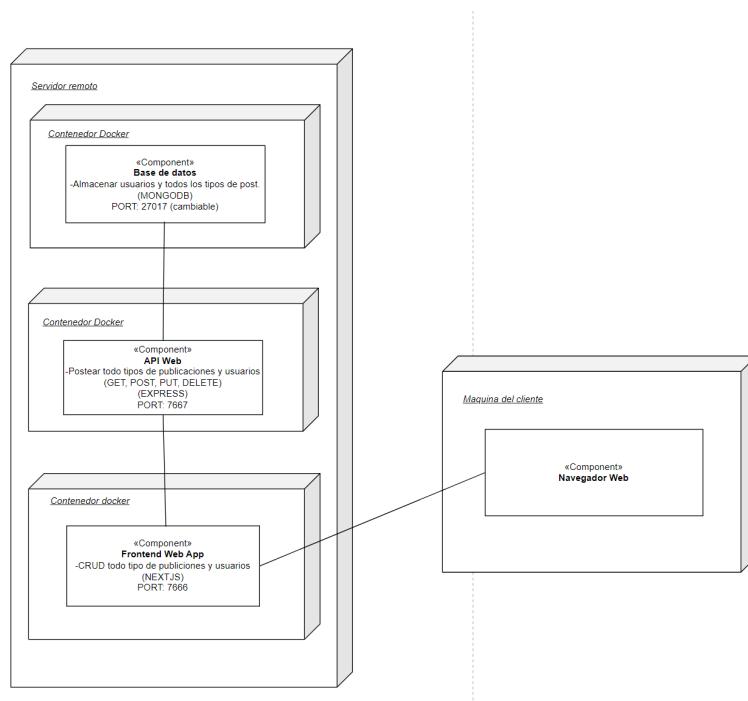
```
./dist/
./node_modules
./__test__
.gitignore
README.md
```

Frontend

```
./.next  
./node_modules  
.gitignore  
README.md
```

Estos archivos son lo primero que se lee al ejecutar el comando `docker-compose up` para levantar los contenedores dentro del servidor remoto.

Con lo anterior descrito se sigue una estructura similar a la siguiente



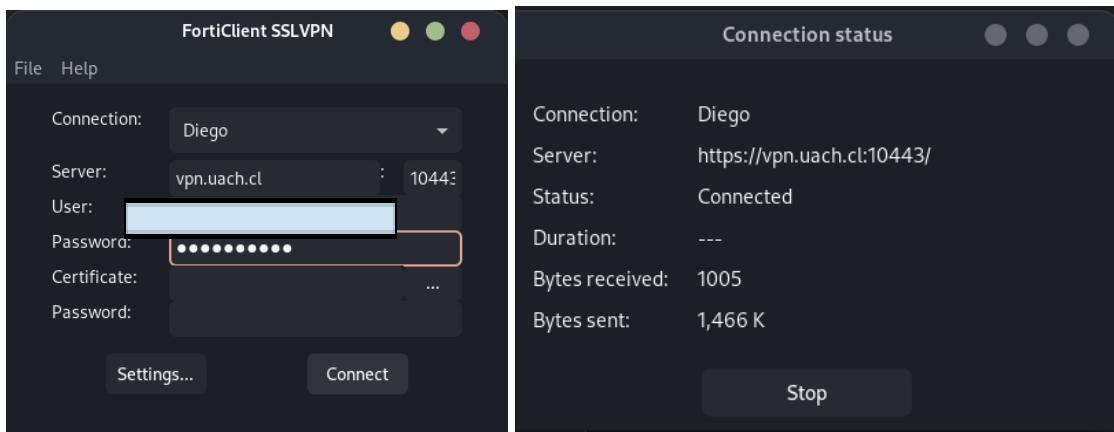
Importante

Para el correcto funcionamiento de la aplicación reemplazar `localhost` por `ip_servidor_remoto`, en la configuración de las variables de entorno (`.env`) de los servicios backend y frontend.

Preparación servidor host

Conexión

Para poder conectarnos al servidor debemos utilizar una vpn, de la Universidad Austral de Chile. [Link descarga](#). **Solo para este caso en específico, lo anterior puede ser omitido de no ser necesaria la vpn.**



Nos conectamos al servidor mediante el protocolo ssh, protocolo de red destinado principalmente a la conexión con máquinas a las que accedemos por línea de comandos. Entonces en un terminal de linux (fedora en este caso) ejecutamos el siguiente comando.

```
$ ssh nombre_usuario@ip_servidor_remoto
```

Donde `nombre_usuario`, es el nombre del usuario en el servidor remoto al cual se desea acceder y `ip_servidor_remoto` es la ip del servidor al cual queremos ingresar. Si es la primera vez que intentamos acceder a este servidor (host) remoto nos solicitará guardar una “llave” en nuestro sistema, con un mensaje similar al siguiente.

```
The authenticity of host 'x.x.x.x' can't be established.
ECDSA key fingerprint is
SHA256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.
Are you sure you want to continue connecting (yes/no)?
```

Al ingresar (yes) guardará la clave del host en tu sistema, y en futuras conexiones no se pedirá confirmar la autenticidad del servidor nuevamente.

Ahora se debe ingresar la clave del usuario, si todo esta correcto, accederemos al servidor donde hospedaremos la aplicación.

```
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.4.0-152-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```

Ahora dentro del servidor crearemos un directorio en la ruta `/home` del servidor donde almacenaremos la aplicación. p.e:

```
$ mkdir proyecto-egresados
```

En otra terminal del sistema, deberemos realizar una copia segura de la carpeta de tu

aplicación hacia el servidor remoto. Esto es sencillo, se realiza con el comando:

```
$ scp app-egresados  
<nombre_usuario>@<ip_servidor_remoto>:/home/<usr>/proyecto-egresados
```

Se solicitará nuevamente la contraseña del usuario y se empezará a copiar la aplicación. En caso de no poder copiarse la aplicación con el comando `scp`, existe la opción de subir el proyecto a un repositorio github y dentro del servidor hacer `git clone` del repositorio.

```
$ foo@foo ~/home/foo/proyecto-egresados/app-egresados git clone  
<url_repositorio_git>
```

El repositorio debe seguir la estructura mencionada en “Preparación aplicación”.

Ahora dentro del terminal nos moveremos dentro de la carpeta `proyecto-egresados/app-egresados`.

```
$ cd proyecto-egresados/app-egresados
```

Antes de levantar los contenedores

Verificamos que los puertos que se utilizarán para los contenedores no estén siendo utilizados por ningún proceso en el servidor.

Ejecutamos lo siguiente:

```
$ netstat -tuln
```

Verificamos que los puertos no se encuentren en la lista.

Active Internet connections (only servers)					
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	0.0.0.0:8080	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:50000	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:8081	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:8180	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.53:53	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:8888	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:5150	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:41823	0.0.0.0:*	LISTEN
...					

Ahora verificamos que los puertos no estén bloqueados por firewall (UFW):

```
$ sudo ufw status
```

```
Status: active
```

To	Action	From
--	-----	-----
22/tcp	ALLOW	Anywhere
80/tcp	ALLOW	Anywhere
5000/tcp	ALLOW	Anywhere
8000	ALLOW	Anywhere
4040	ALLOW	Anywhere
3308	ALLOW	Anywhere
7667	ALLOW	Anywhere
7667/tcp	ALLOW	Anywhere
7666/tcp	ALLOW	Anywhere
7668/tcp	ALLOW	Anywhere
22/tcp (v6)	ALLOW	Anywhere (v6)
80/tcp (v6)	ALLOW	Anywhere (v6)
5000/tcp (v6)	ALLOW	Anywhere (v6)
8000 (v6)	ALLOW	Anywhere (v6)
4040 (v6)	ALLOW	Anywhere (v6)
3308 (v6)	ALLOW	Anywhere (v6)
7667 (v6)	ALLOW	Anywhere (v6)
7667/tcp (v6)	ALLOW	Anywhere (v6)
7666/tcp (v6)	ALLOW	Anywhere (v6)
7668/tcp (v6)	ALLOW	Anywhere (v6)

Levantando los contenedores

En el servidor y con todas las preparaciones realizadas en la raiz de la aplicación ejecutamos:

```
$ docker-compose up
```

Este comando creara las imágenes y contenedores necesarios para levantar la aplicación.

▼ IMAGES

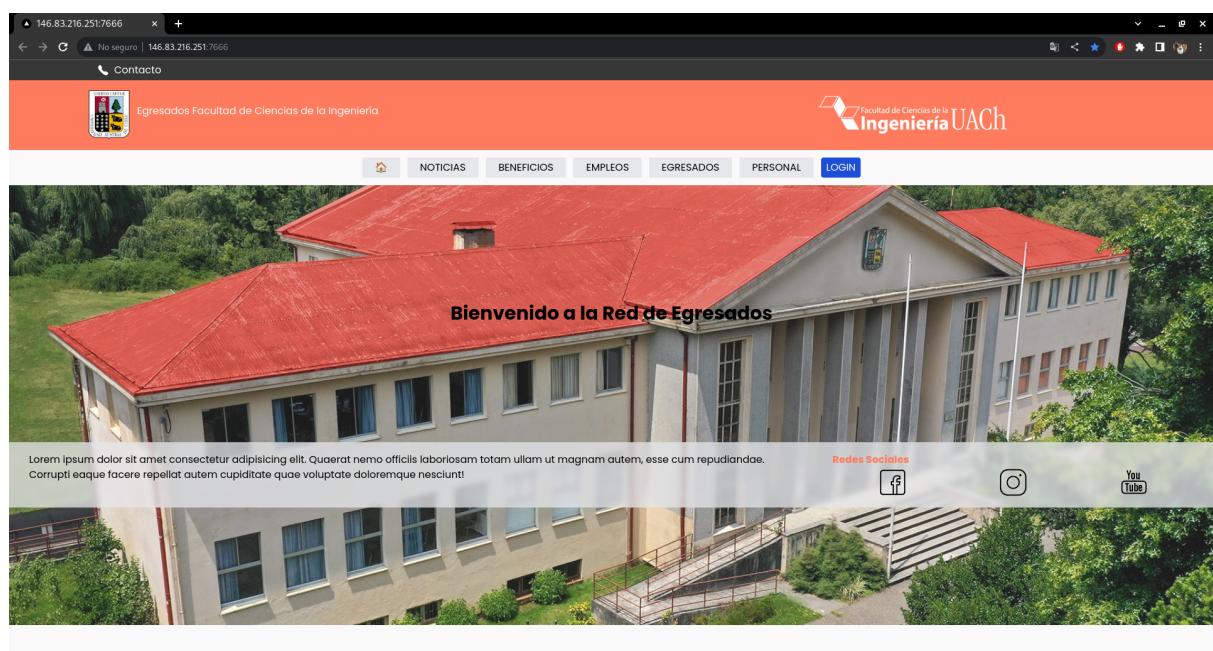
- > app-egresados_front-egresados
- > app-egresados_back-egresados
- > app-egresados_front-egresados

▼ CONTAINERS

- > ~~app-egresados~~
- ▼ app-egresados
 - > app-egresados_front-egresados front-egresados - Up 5 hours
 - > app-egresados_back-egresados back-egresados - Up 5 hours
 - > mongo:4.4.18 mongodb - Up 5 hours

Las imágenes son desechables, es decir, se pueden eliminar una vez creado los contenedores.

Ahora se puede acceder a la aplicación a través del frontend: ip_servidor_host:7666.



Anexo

Docker: <https://www.docker.com/>

Mongo: https://hub.docker.com/_/mongo

Docker-compose: <https://docs.docker.com/compose/>

App-egresados: <https://github.com/GeoCurguan/INFO248-Egresados/tree/docker-final>