**GUIA PARA DOCKERIZAR PLATAFORMA DORY**

En este documento, aprenderás cómo aprovechar la potencia de Docker para empaquetar y distribuir tu aplicación web y API REST de manera eficiente y portátil. Docker es una herramienta que facilita la creación de entornos aislados y reproducibles, lo que te permitirá garantizar la consistencia entre los distintos entornos de desarrollo, pruebas y producción.

En esta guía, te proporcionaremos una explicación paso a paso de los conceptos básicos de Docker, cómo crear imágenes personalizadas para tu WebApp y API REST, y cómo gestionar tus contenedores de forma efectiva. Con Docker, podrás simplificar el despliegue de tu aplicación, reducir problemas de dependencias y asegurar una ejecución fluida en cualquier entorno compatible.

Continuemos esta guía paso a paso para Dockerizar la plataforma Dory:

**Paso 1: Instalación de Docker**

Antes de comenzar, asegúrate de tener Docker instalado en tu sistema.

* Para Windows y macOS: Dirígete al sitio web oficial de Docker y descarga la versión adecuada para tu sistema operativo. Sigue las instrucciones de instalación proporcionadas por el instalador.
* Para Linux: Dependiendo de la distribución de Linux que estés utilizando, los pasos de instalación pueden variar.

**Paso 2: Preparación del Proyecto**

Antes de dockerizar la plataforma Dory, es fundamental tener un proyecto estructurado y listo para trabajar. Asegúrate de que todos los archivos y dependencias necesarias estén correctamente organizados y funcionando en tu entorno local.

**DORY API REST**

Para descargar el proyecto de la API- REST, clona el repositorio del proyecto en un directorio de tu computadora. Recuerda que para este paso debes tener Git instalado en tu sistema.

* git clone <https://github.com/doryteam1/dory-api-rest.git>

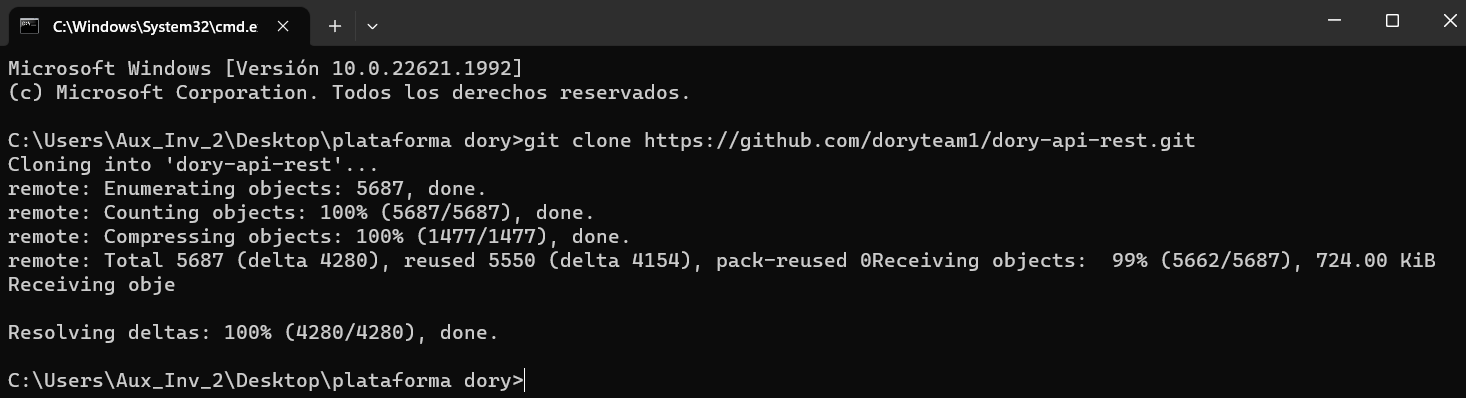


Figura 1. Clonando Dory API-REST

Al terminar la ejecución del comando anterior dejara como resultado una carpeta llamada dory-api-rest con el código fuente del proyecto.

Ahora abriremos el proyecto con el editor de nuestra preferencia, en esta guía utilizaremos Visual Studio Code, para esto nos ubicaremos dentro de la carpeta del proyecto y procedemos a abrirlo con el editor de la siguiente manera:

* cd dory-api-rest
* code .

El comando “code .” abre el directorio donde no encontramos ubicados actualmente en visual studio code por lo que veras la siguiente vista con la estructura del proyecto.

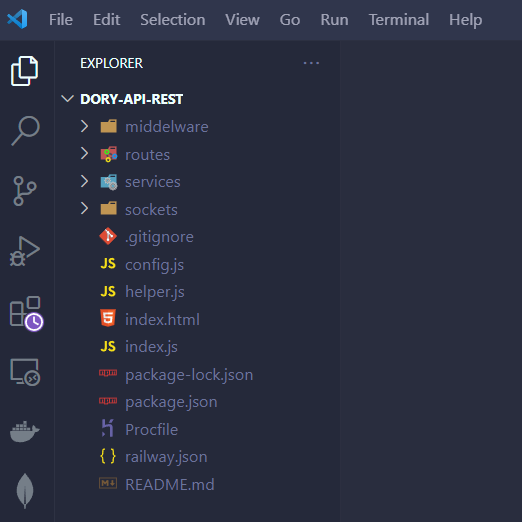


Figura 2. Código fuente de la API-REST

Ahora debemos comprobar el funcionamiento de la app de forma local, pero antes de colocar en marcha la aplicación debemos crear un archivo con las variables de entorno donde se definirán variables necesarias para que la aplicación se ejecute correctamente, algunas de estas variables son confidenciales y es por eso que no se incluye directamente.

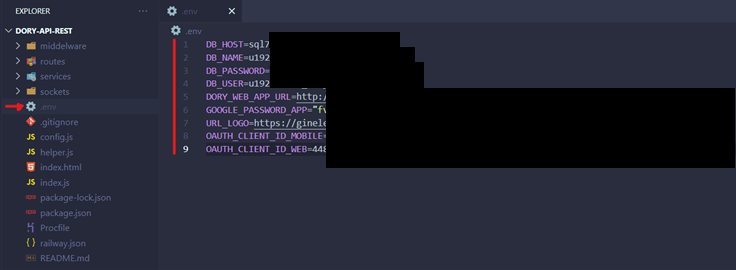


Figura 3. Agregando archivo “.env” con variables de entorno de la API- REST

Las variables definidas en esta guía son para el entorno de pruebas, para el entorno de producción se deben modificar.

Hasta esta parte ya tenemos el proyecto configurado y listo para la puesta en marcha, para ejecutar la aplicación instalaremos las dependencias y ejecutaremos la app con los siguientes comandos

* npm install
* npm run start

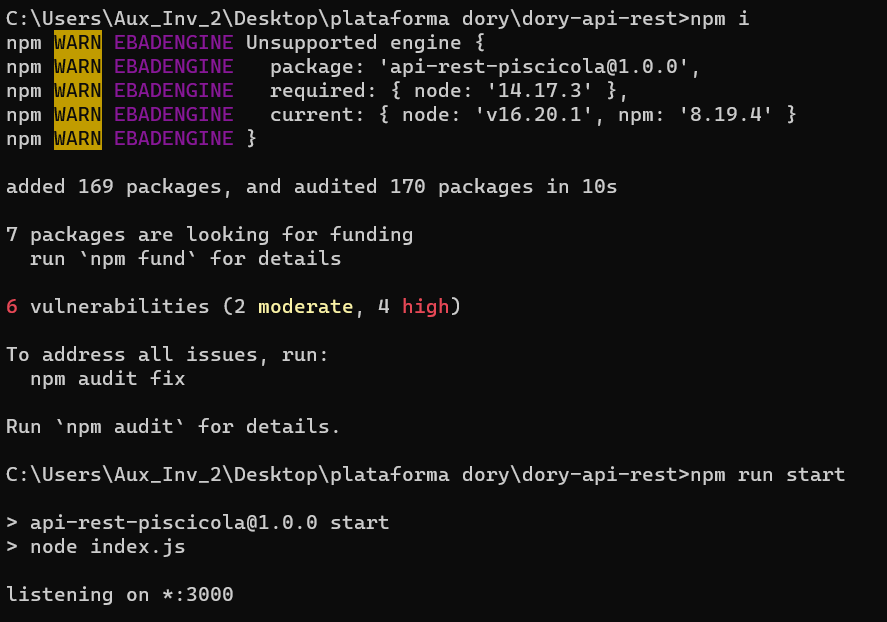


Figura 4. Instalación de dependencias y ejecución de la App

Ahora verificaremos el funcionamiento accediendo a la siguiente ruta desde el navegador <http://localhost:3000> y deberíamos tener el siguiente resultado



Figura 5. API- REST corriendo en el puerto 3000

**DORY WEB APP**

Para descargar el proyecto de la aplicación web, clona el repositorio del proyecto en un directorio de tu computadora.

* git clone <https://github.com/doryteam1/dory-web-app.git>

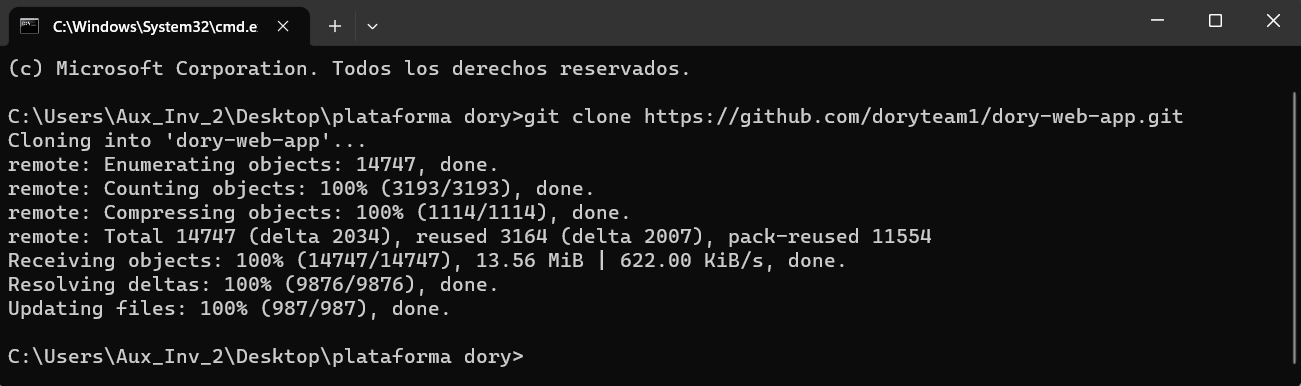


Figura 6. Clonando aplicación web Dory

Al terminar la ejecución del comando anterior dejara como resultado una carpeta llamada dory-web-app con el código fuente del proyecto.

Ahora abriremos el proyecto con Visual Studio Code, para esto nos ubicaremos dentro de la carpeta del proyecto y procedemos a abrirlo con el editor de la siguiente manera:

* cd dory-web-app
* code .

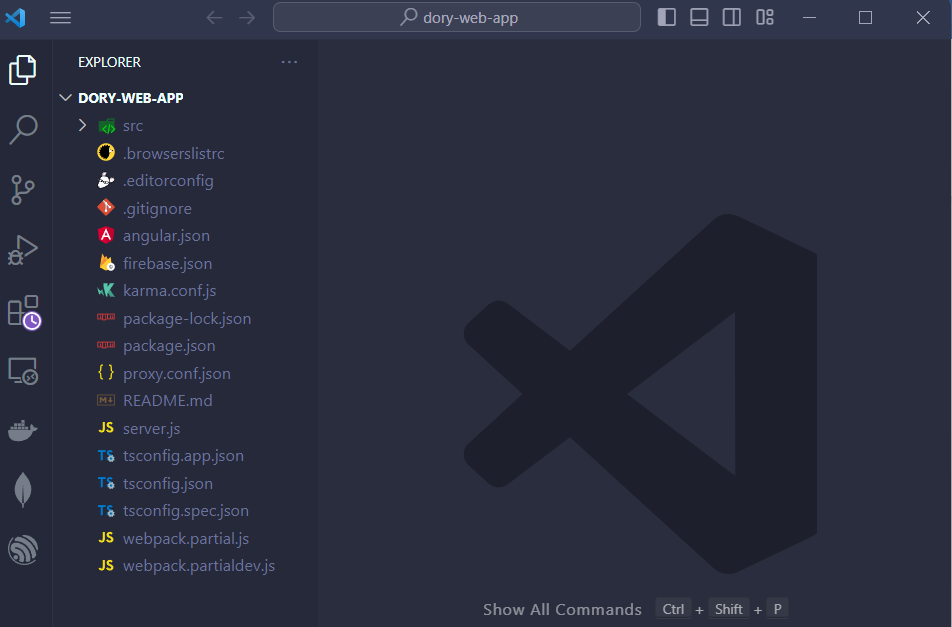
****

Figura 7. Código fuente de la aplicación web Dory

Ahora comprobaremos el funcionamiento de la app de forma local, igual que con la app anterior antes de colocar en marcha la aplicación debemos crear un archivo con las variables de entorno donde se definirán variables necesarias para que la aplicación se ejecute correctamente, algunas de estas variables son confidenciales y es por eso que no se incluye directamente.

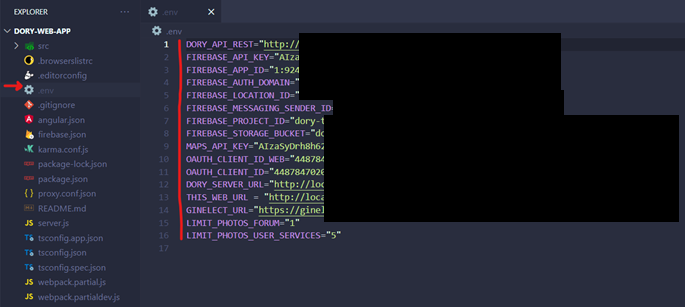


Figura 8. Agregando archivo “.env” con variables de entorno de la app web

Las variables definidas en esta guía son para el entorno de pruebas, para el entorno de producción se deben modificar.

Hasta esta parte ya tenemos el proyecto configurado y listo para la puesta en marcha, para ejecutar la aplicación instalaremos las dependencias y ejecutaremos la app con los siguientes comandos

* npm install –force
* npm run ini

Ahora verificaremos el funcionamiento accediendo a la siguiente ruta desde el navegador <http://localhost:4200> y deberíamos tener el siguiente resultado

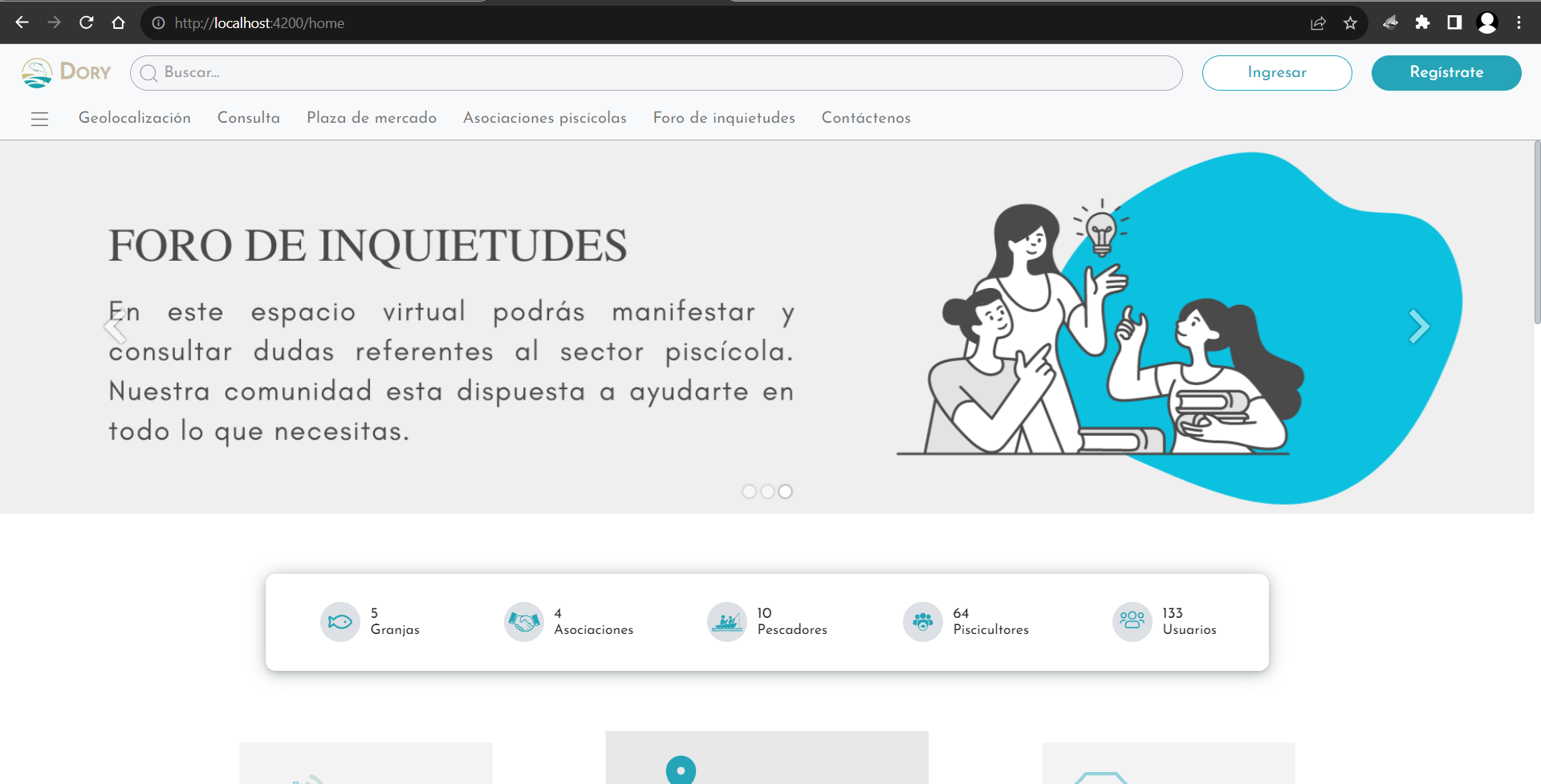


Figura 9. APP WEB corriendo en el puerto 4200

**Paso 3: Creación del Dockerfile**

El Dockerfile es un archivo de texto que contiene instrucciones para construir la imagen de Docker que representará tu aplicación. Aquí aprenderás cómo crear un Dockerfile personalizado para Dory. Te guiaremos a través de cada paso necesario para tener un entorno aislado para tu aplicación.

**DORY API REST**

Antes de crear nuestro Dockerfile lo que haremos será crear una archivo “.dockerignore” , aquí definiremos los archivos que no queremos que se agreguen a la imagen de Docker, este archivo se vería de la siguiente manera:

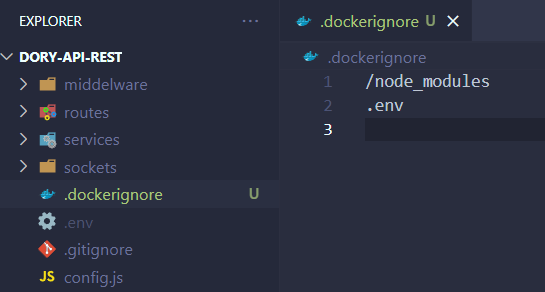


Figura 10. Archivo dockerignore de la API-REST

Con esto lo que hacemos es decirle a Docker que archivos no se tendrán en cuenta para la construcción de la imagen. Ahora pasaremos a ver como quedaría el Dockerfile para la construcción de la imagen:

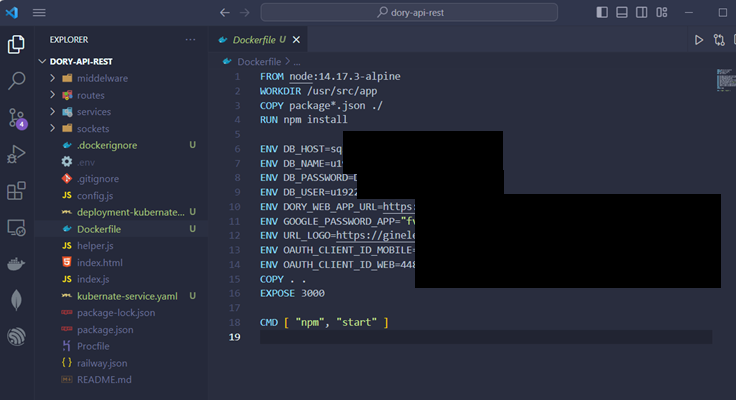


Figura 11. Archivo Dockerfile para la API-REST

1. **FROM node:14.17.3-alpine**: Esta línea indica que estás utilizando la imagen oficial de Node.js en su versión 14.17.3 basada en Alpine Linux como la imagen base.
2. **WORKDIR /usr/src/app**: Establece el directorio de trabajo en **/usr/src/app** dentro del contenedor. Aquí es donde se copiará el código de tu aplicación y donde se ejecutarán los comandos subsiguientes.
3. **COPY package\*.json ./**: Copia los archivos **package.json** y **package-lock.json** (si está presente) desde el directorio local al directorio de trabajo del contenedor.
4. **RUN npm install**: Este comando instala las dependencias de Node.js basadas en los archivos **package.json** y **package-lock.json** que se copiaron en el paso anterior.
5. Definiciones de Variables de Entorno (**ENV** lines): Estas líneas establecen varias variables de entorno que tu aplicación de Node.js podría necesitar. Estas incluyen detalles de conexión a la base de datos, URLs y IDs de cliente de OAuth.
6. **COPY . .**: Este comando copia el resto del código de tu aplicación (incluidos todos los archivos y directorios) desde el directorio local al directorio de trabajo del contenedor.
7. **EXPOSE 3000**: Esto expone el puerto 3000 en el contenedor. Esto no publica el puerto en la máquina host, simplemente documenta que el contenedor escuchará en este puerto.
8. **CMD [ "npm", "start" ]**: Esto especifica el comando predeterminado que se ejecutará cuando el contenedor se inicie. Ejecuta el comando **npm start**, que presumiblemente inicia tu aplicación de Node.js.

En resumen, este Dockerfile establece un entorno para construir una imagen Docker para una aplicación de Node.js. Instala las dependencias de la aplicación, configura las variables de entorno, copia el código de la aplicación y define el comando predeterminado para iniciar la aplicación. Una vez construida, esta imagen puede utilizarse para crear contenedores que ejecuten tu aplicación de Node.js con la configuración especificada.

**DORY WEB APP**

Al igual que con la API, lo primero que haremos será crear un archivo “.dockerignore” que se vera de la siguiente manera:

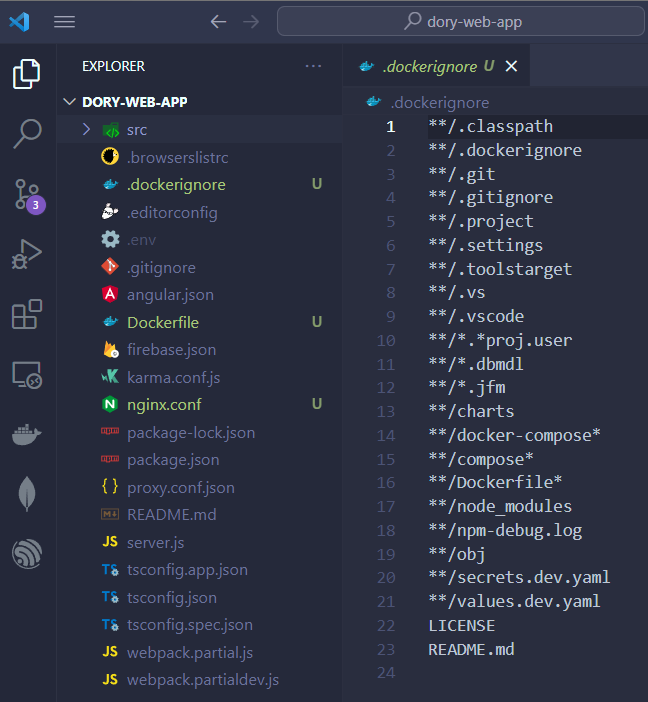


Figura 12. Archivo dockerignore de la APP WEB

Lo siguiente es crear el archivo Dockerfile con las instrucciones para construir la aplicación:

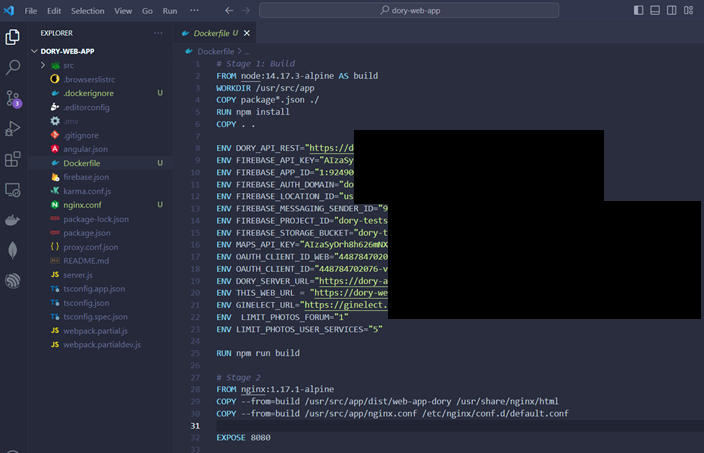


Figura 13. Archivo Dockerfile para la APP WEB

Este Dockerfile se compone de dos etapas. La primera etapa construye la aplicación Node.js y la compila, mientras que la segunda etapa crea una imagen basada en Nginx para servir los archivos estáticos generados por la compilación.

**Stage 1: Build**

1. **FROM node:14.17.3-alpine AS build**: En esta línea, se inicia el primer paso de construcción utilizando la imagen oficial de Node.js 14.17.3 basada en Alpine Linux como base. Se establece un nombre para esta etapa como "build".
2. **WORKDIR /usr/src/app**: Establece el directorio de trabajo en **/usr/src/app** dentro del contenedor. Aquí es donde se llevarán a cabo las operaciones de construcción.
3. **COPY package\*.json ./**: Copia los archivos **package.json** y **package-lock.json** (si existe) del directorio local al directorio de trabajo del contenedor.
4. **RUN npm install**: Ejecuta el comando **npm install** para instalar las dependencias de Node.js basadas en los archivos **package.json** y **package-lock.json** copiados anteriormente.
5. **COPY . .**: Copia todo el contenido del directorio local al directorio de trabajo del contenedor. Esto incluirá los archivos de la aplicación y cualquier otro archivo necesario.
6. Definiciones de variables de entorno (**ENV** lines): Estas líneas establecen varias variables de entorno que la aplicación utilizará para su configuración. Incluyen valores para URLs, claves de API y otros detalles.
7. **RUN npm run build**: Ejecuta el comando **npm run build** para construir la aplicación. Esto generalmente compilará el código fuente de la aplicación y generará los archivos necesarios para su implementación.

**Stage 2: Deploy**

1. **FROM nginx:1.17.1-alpine**: Inicia la segunda etapa de construcción utilizando la imagen oficial de Nginx 1.17.1 basada en Alpine Linux como base.
2. **COPY --from=build /usr/src/app/dist/web-app-dory /usr/share/nginx/html**:

Copia el contenido del directorio de salida de la compilación (por ejemplo, los archivos generados después de ejecutar **npm run build**) desde la etapa de construcción anterior al directorio de contenido estático de Nginx.

1. **COPY --from=build /usr/src/app/nginx.conf /etc/nginx/conf.d/default.conf**:

Copia el archivo **nginx.conf** desde la etapa de construcción anterior al directorio de configuración de Nginx, lo que permite personalizar la configuración del servidor Nginx.

1. **EXPOSE 8080**: Expone el puerto 8080 en el contenedor. Esto no publica el puerto en la máquina host, simplemente documenta que el contenedor escuchará en este puerto.

En el paso 3 de la etapa de deploy vemos que se utiliza un archivo de configuración para NGINX, este archivo llamado “nginx.conf” debemos crearlo en la raíz del proyecto y quedaría como se muestra a continuación:



Figura 14. Archivo de configuración de NGINX

Este archivo de configuración de Nginx define cómo el servidor Nginx servirá los archivos estáticos de una aplicación web. Configura la compresión Gzip para mejorar el rendimiento, establece el directorio raíz donde se encuentran los archivos, y redirige todas las solicitudes a la raíz a **index.html** para permitir el enrutamiento en el lado del cliente.

**Paso 4: Construcción de la Imagen Docker**

Una vez que hayas creado el Dockerfile, el siguiente paso es construir la imagen de Docker. Esta imagen es la representación de tu aplicación junto con todas sus dependencias y configuraciones necesarias. Aprenderás cómo utilizar el comando **docker build** para construir la imagen.

**DORY API REST**

Para crear la imagen de la aplicación, abre una ventana de comando en la raíz del proyecto y ejecuta el siguiente comando:

* docker build -t dory-api .

El comando “**docker build -t dory-api .”** se utiliza para construir una imagen de Docker utilizando un Dockerfile presente en el directorio actual ("."). A continuación, te explicaré los componentes del comando:

* **docker build**: Este comando inicia el proceso de construcción de una imagen de Docker.
* **-t dory-api**: El flag **-t** se utiliza para etiquetar la imagen resultante con un nombre y, opcionalmente, una etiqueta. En este caso, se etiqueta la imagen con el nombre **dory-api**.
* **.**: Esto representa el contexto de construcción. Es el directorio actual desde donde se ejecuta el comando **docker build**. Todos los archivos y directorios en ese directorio se considerarán parte del contexto y se copiarán al entorno de construcción del contenedor temporal mientras se ejecuta el Dockerfile.

En resumen, el comando **docker build -t dory-api .** toma el Dockerfile presente en el directorio actual y lo utiliza para construir una imagen de Docker llamada "dory-api". El punto (**.**) al final del comando indica que se toma todo el contenido del directorio actual como parte del contexto de construcción para la imagen.

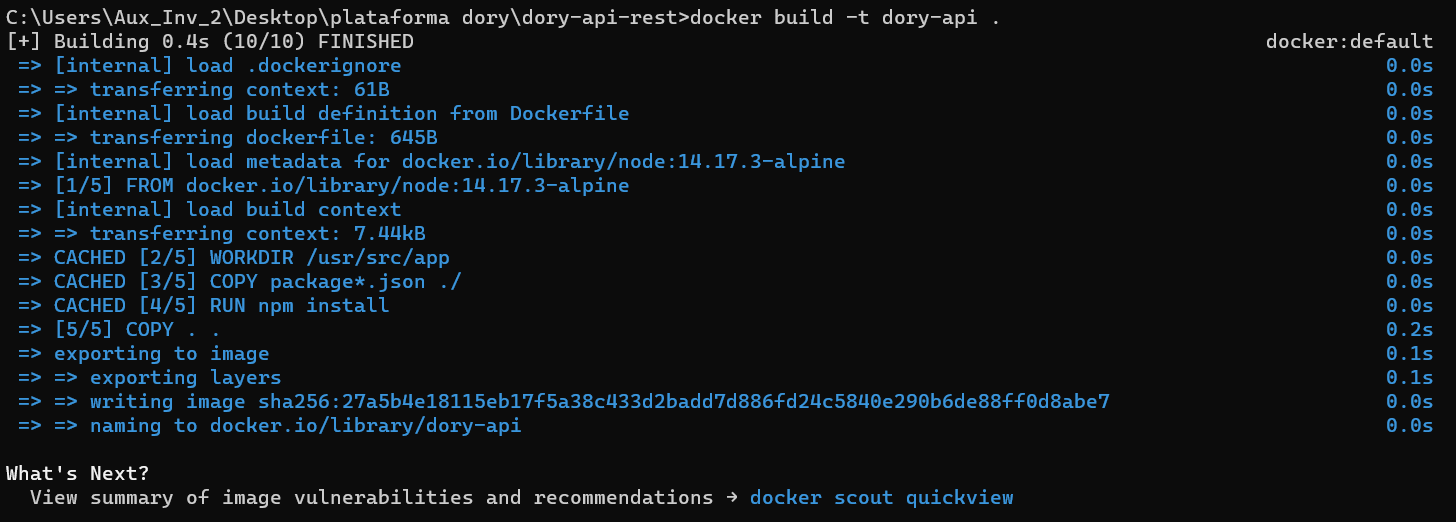


Figura 15. Creación de la imagen de la API-REST

Para ver una lista de imágenes que están almacenadas en tu sistema Docker se utiliza el comando:

* docker images

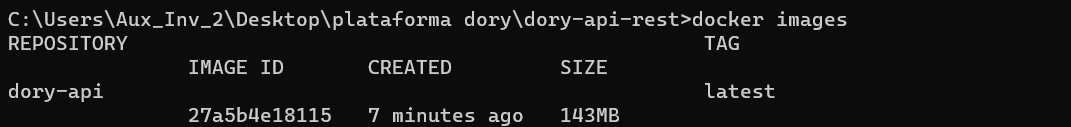


Figura 16. Listando imágenes Docker

**DORY WEB APP**

Abre una ventana de comando en la raíz del proyecto y ejecuta el siguiente comando:

* docker build -t dory-web .

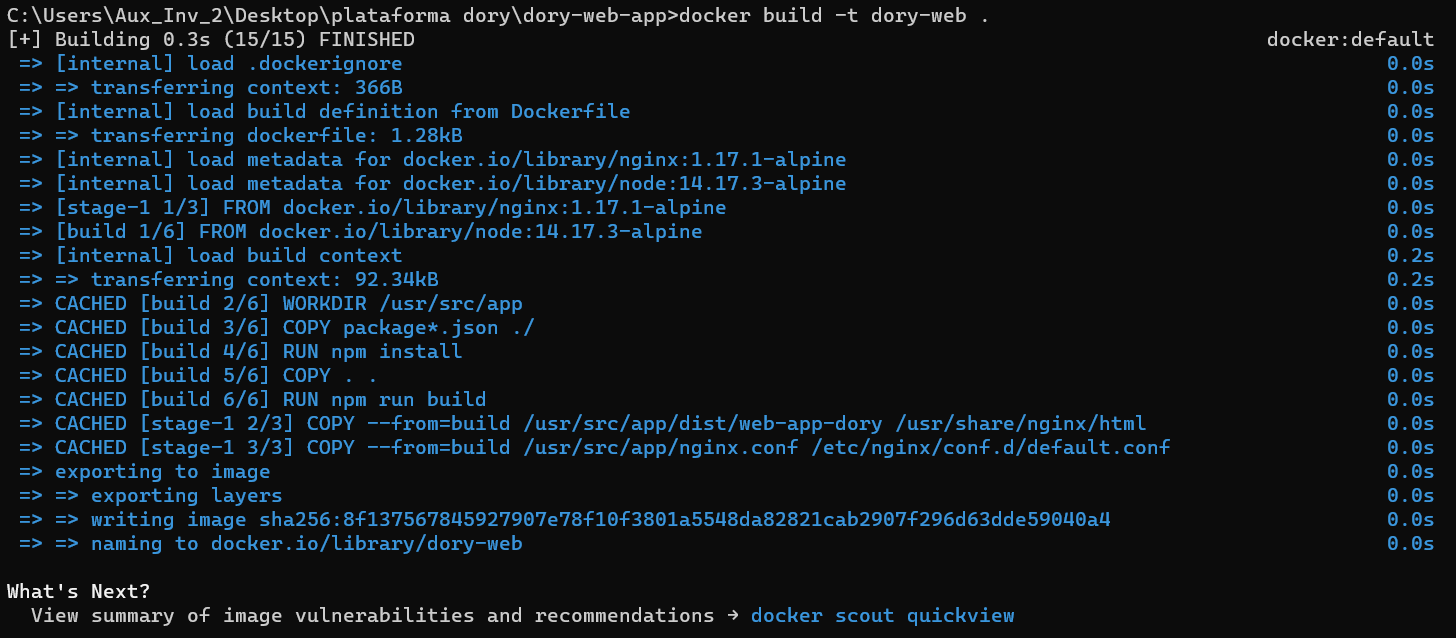


Figura 17. Creación de la imagen para la APP-WEB

Al listar nuevamente las imágenes veras que ahora tendremos dos imágenes correspondientes a las dos aplicaciones.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 18. Imágenes creadas

**Paso 5: Ejecución del Contenedor**

Una vez que tengas tu imagen de Docker creada, es hora de ejecutar un contenedor basado en esa imagen. Aprenderás cómo usar el comando docker run para iniciar el contenedor y acceder a tu plataforma Dory desde el navegador.

**DORY API REST**

Para ejecutar la imagen creada como un contenedor de Docker ejecutaremos el comando:

* docker run -p 8080:3000 -d dory-api

A continuación, desgloso los elementos del comando:

* **docker run**: Inicia un nuevo contenedor a partir de una imagen.
* **-p 8080:3000**: Este flag establece la redirección de puertos del host al contenedor. El puerto 8080 del host se mapea al puerto 3000 dentro del contenedor. Esto significa que las solicitudes que llegan al puerto 8080 del host se redirigen al puerto 3000 del contenedor.
* **-d**: Este flag indica que el contenedor se ejecutará en segundo plano (modo detached).
* **dory-api**: Es el nombre de la imagen que se utilizará para crear el contenedor. El sistema Docker buscará esta imagen localmente y la usará para iniciar el contenedor.



Figura 19. Ejecutando comando docker run -p 8080:3000 -d dory-api

Para comprobar el funcionamiento de la app, accederemos a la url <http://localhost:8080> , el puerto 8080 fue el que indicamos en el comando anterior y puede modificarse de acuerdo a nuestras necesidades.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 20. API-REST corriendo desde un contenedor de Docker

**DORY WEB APP**

Para ejecutar la imagen creada como un contenedor de Docker ejecutaremos el comando:

* docker run -p 4200:8080 -d dory-web

Para comprobar el funcionamiento de la app, accederemos a la url <http://localhost:4200> , el puerto 4200 fue el que indicamos en el comando anterior.

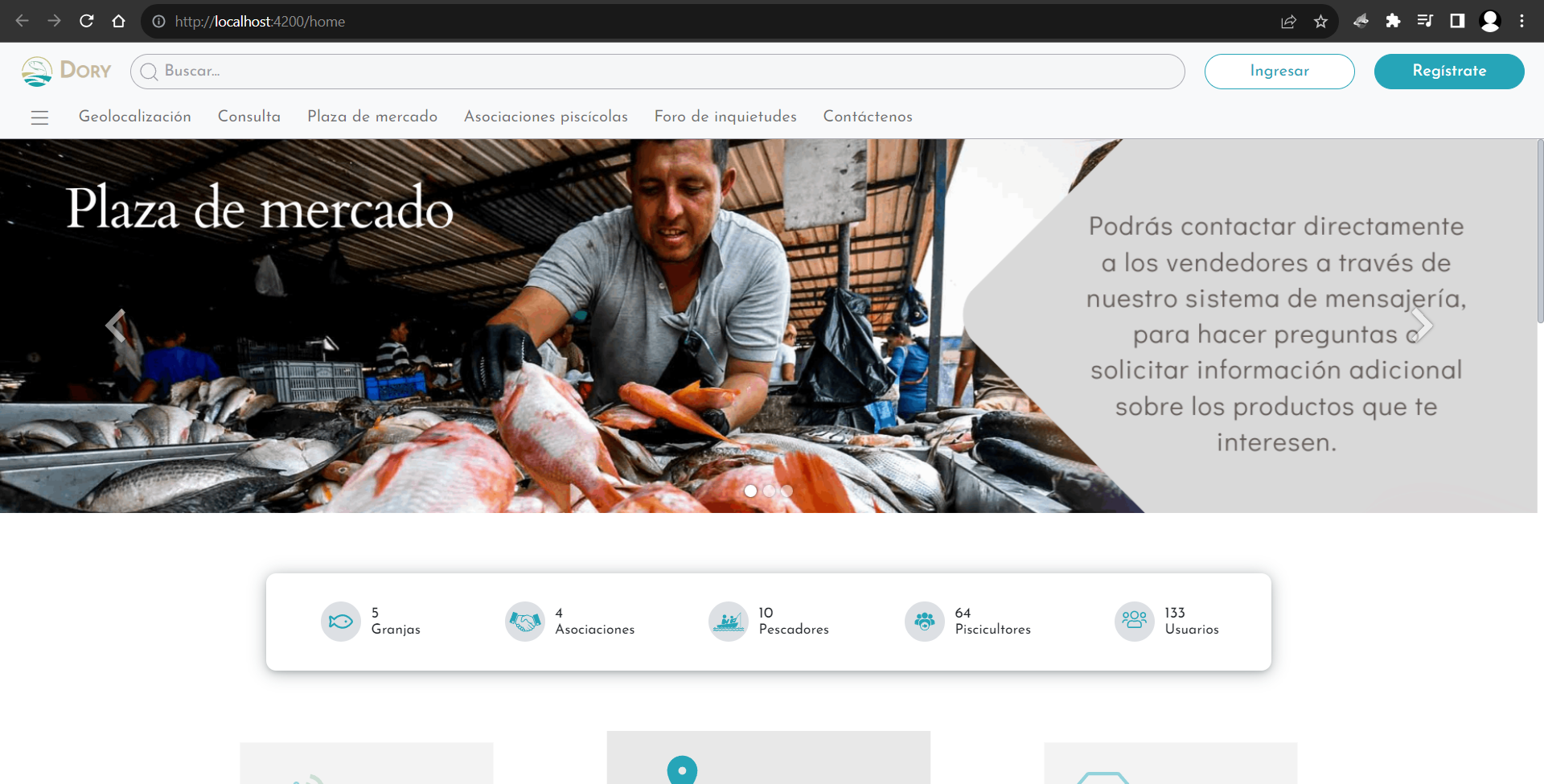


Figura 21. DORY WEB corriendo desde un contenedor de Docker

Para listar lo contenedor corriendo actualmente podemos utilizar el comando

* docker ps

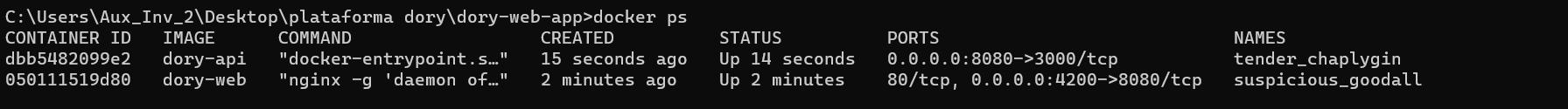


Figura 22. Listando contenedores corriendo en Docker

Para detener el contenedor utilizaremos el comando

* docker stop “container\_id“

el container\_id se coloca sin comillas y se puede obtener de los resultados del comando docker ps.

¡Felicidades! Has completado con éxito la guía para dockerizar la plataforma Dory. Ahora tienes tus aplicaciones web y API REST funcionando en contenedores de Docker, lo que te proporciona portabilidad y aislamiento en diferentes entornos.