

# PRACTICA 5

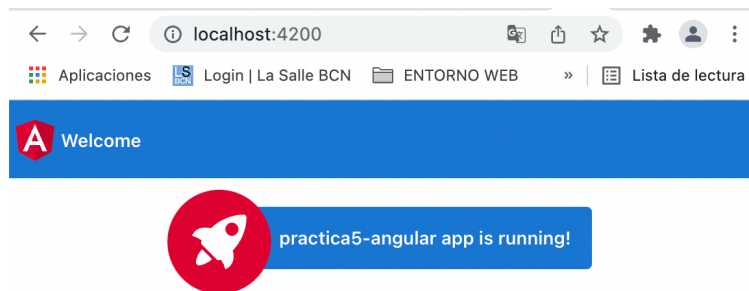
## Índice

<b>PRACTICA 5 .....</b>	<b>1</b>
<b>1.0 BUILD NODE PROJECT .....</b>	<b>2</b>
<b>2.0 TEST NODE PROJECT .....</b>	<b>3</b>
<b>3.0 SUBIR EL PROYECTO A UN REPOSITORIO DE GIT .....</b>	<b>4</b>
<b>4.0 DOCKERIZAR UNA IMAGEN DE ANGULAR .....</b>	<b>5</b>
<b>5.0 DESPLEGAR APLICACIÓN EN KUBERNETES.....</b>	<b>7</b>
<b>6.0 EXPLICACIÓN DEL PROCESO DE DISEÑO E IMPLEMENTACIÓN DE CI/CD.....</b>	<b>8</b>

## 1.0 BUILD NODE PROJECT

Para esta práctica he usado el framework de Angular que permite el desarrollo de aplicaciones web SPA, con el entorno de ejecución Node.js y el gestor de paquetes por defecto npm, que se ejecuta desde la línea de comandos y maneja las dependencias de la aplicación.

Para ver el contenido de esta aplicación, vamos a lanzar el comando `ng serve` desde el directorio de la aplicación para compilar el código y exponerlo en el puerto 4200 (puerto por defecto de angular). Hacemos una petición a `localhost:4200` para ver el contenido de la aplicación que, siendo muy sencilla, nos devuelve el mensaje “practica5-angular app is running!”. Este paso solo lo vamos a hacer para comprobar que la aplicación compila correctamente y tener una primera visión de cómo es la aplicación.



Una vez comprobado que funciona correctamente, pasamos a la primera fase del proceso de CI/CD que sería construir el proyecto. Para ello se lanza el comando **ng build** que crea la carpeta `dist` dentro del proyecto con el contenido estático que compone la aplicación.

```
arianegutierrez@MacBook-Pro-de-Ariane practica5-angular % ng build
✓ Browser application bundle generation complete.
✓ Copying assets complete.
✓ Index html generation complete.

Initial Chunk Files | Names | Size
main.96373f2cde2213b3.js | main | 181.17 kB
polyfills.ec754e06a0702b33.js | polyfills | 36.21 kB
runtime.f67a42c6437ed9f2.js | runtime | 1.06 kB
styles.ef46db3751d8e999.css | styles | 0 bytes

Initial Total | 218.44 kB
```

## 2.0 TEST NODE PROJECT

Los test unitarios nos permiten probar pequeñas partes del código sin que nuestra aplicación esté encendida. En este caso vamos a usar el framework de Jasmine que sirve para realizar las pruebas unitarias de código JavaScript. Y la herramienta de Karma que es sobre la cual corren las pruebas de Jasmine.

Por defecto angular crea test unitarios para cada componente de la aplicación con la extensión “spec.ts”. Como esta aplicación es muy sencilla, utilizaremos el test del único componente que es parte de la aplicación como prueba unitaria. En este caso sería el siguiente, que solo prueba que el componente se ha creado correctamente:

```
import { TestBed } from '@angular/core/testing';
import { RouterTestingModule } from '@angular/router/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [
        RouterTestingModule
      ],
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  });

  it('should create the app', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
  });

  it('should have as title \'practica5-angular\'', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app.title).toEqual('practica5-angular');
  });


  it('should render title', () => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.nativeElement as HTMLElement;
    expect(compiled.querySelector('.content span')?.textContent).toContain('practica5-angular app is running!');
  });
});
```

En el apartado *describe* ponemos una descripción al grupo de test que vamos a crear, y con cada *it* vamos a establecer un test en particular. Dentro de cada test está el apartado *expect* para definir como debe comportarse ese test en concreto.

Para probar los test debemos ejecutar el comando **ng test**. Y este sería el resultado de esas pruebas:

**Karma v 6.3.9 - connected; test: complete;** DEBUG

Chrome 96.0.4664.55 (Mac OS 10.15.7) is idle

 **Jasmine** 3.10.1 Options

3 specs, 0 failures, randomized with seed 70662 finished in 0.05s

AppComponent

- should create the app
- should have as title 'practica5-angular'
- should render title

### 3.0 SUBIR EL PROYECTO A UN REPOSITORIO DE GIT

Primero vamos a crear un repositorio remoto en GitHub con el nombre “practica5-angular” para así poder vincular nuestro proyecto.

Los siguientes comandos los lanzaremos desde el directorio donde se encuentra el proyecto:

- Creamos la carpeta .git  
`Git init`
- Vinculamos el proyecto con el repositorio remoto  
`Git remote add origin https://github.com/ariane10/practica5-angular.git`
- Creamos el archivo .gitignore para que no suban al repositorio todas las librerías de la carpeta node\_modules y el .angular

`Touch .gitignore`  
`Nano .gitignore`

```
GNU nano 2.0.6  File: .gitignore
/node_modules
.angular
```

Después de vincular el proyecto, todos los cambios que queramos subir se harán con los siguientes comandos:

```
git commit
git push -u origin master
```

## 4.0 DOCKERIZAR UNA IMAGEN DE ANGULAR

### 1) Crear el Dockerfile

Para crear la imagen de docker de la aplicación vamos a crear un Dockerfile multistage dentro del proyecto con el siguiente contenido:

```
# Step 1
FROM node:16.13.0-alpine3.14 as build

WORKDIR /app

COPY package.json /app

RUN npm install

COPY . /app

RUN ng build

# Step 2
FROM nginx:1.17.1-alpine

COPY --from=build /app/dist/practica5-angular /usr/share/nginx/html
```

En el primer paso vamos a utilizar la imagen base de node con la versión 16.13.0 con la versión 3.14 de alpine y le vamos a llamar **build**. Para crear la imagen vamos a crear una carpeta llamada *app* donde se descargarán todas las librerías necesarias del fichero package.json e irá todo el contenido de la aplicación. Para terminar con este paso compilaremos el código.

En el segundo paso vamos a usar la versión 1.17.1 de nginx con alpine para copiar el resultado del paso anterior indicando en el **--form=build** en el destino de nginx y crear la imagen final.

### 2) Crear el .dockerignore

Vamos a crear un fichero .dockerignore para que a la hora de construir la imagen se haga de forma más rápida y omita lo que no es necesario, como por ejemplo la carpeta node\_modules, ya que se creará cuando se lance el comando “npm install”. Omitiremos también otro tipo de ficheros como los comprimidos o los referentes al git. El contenido de ese fichero es el siguiente:

```
.dockerignore
1  .git
2
3  /node_modules
4
5  .gitignore
6
7  *.zip
```

### 3) Construir la imagen

Para construir la imagen a través del dockerfile, usaremos el siguiente comando (situados desde el directorio donde se encuentra el archivo), y le añadiremos el tag angularapp:

```
docker build -t ariane10/angularapp:v1 .
```

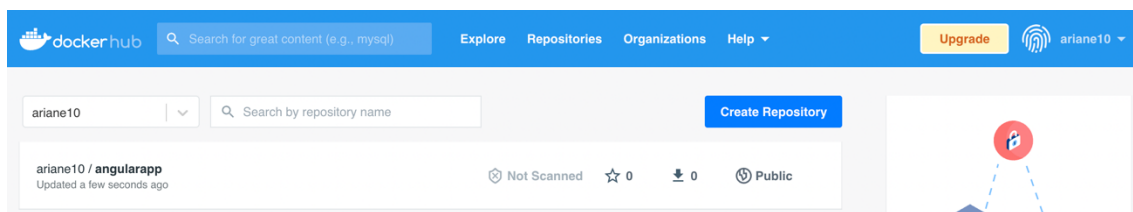
Lanzamos el comando *docker images* para comprobar que se ha creado bien la imagen:

```
[arianegutierrez@MacBook-Pro-de-Ariane practica5-angular % docker images
REPOSITORY          TAG          IMAGE ID       CREATED        SIZE
ariane10/angularapp  v1          5e16a3b55907  3 minutes ago  20MB
jenkins/jenkins     lts-jdk11   3463ba8925cd  8 days ago    431MB
gcr.io/k8s-minikube/kicbase  v0.0.28    c3db8807ea16  2 months ago  981MB
```

### 4) Subir imagen a repositorio privado

Finalmente, subiremos la imagen al repositorio privado de dockerHub

```
docker login
docker push ariane10/angularapp:v1
```



Como esta será la primera versión de la aplicación, el tag que compone el nombre de la imagen será v1, mientras que, en los siguientes despliegues, se ira actualizando dicho tag con la nueva versión. Este tag será un parámetro parte del pipeline, para que en el proceso de CI/CD, el nombre de la imagen se cree dinámicamente a partir de dicho parámetro.

En el pipeline se podrá observar la declaración de ese parámetro de la siguiente manera:

```
parameters {
  string(name: 'TAG', defaultValue: 'v1', description: 'Tag del proceso de CI/CD')
}
```

## 5.0 DESPLEGAR APLICACIÓN EN KUBERNETES

Para desplegar la aplicación en kubernetes vamos a crear un deployment que cree una replica de la imagen publicada en dockerHub. A continuación se puede ver de forma declarativa ese deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: angular-deployment
  labels:
    app: angular
spec:
  replicas: 1
  selector:
    matchLabels:
      app: angular
  template:
    metadata:
      labels:
        app: angular
    spec:
      containers:
        - name: angular
          image: ariane10/angularapp:v1
          ports:
            - containerPort: 80
```

Después creamos un servicio que exponga el pod:

```
apiVersion: v1
kind: Service
metadata:
  name: angular-service
spec:
  type: NodePort
  selector:
    app: angular
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30001
```

Creamos un namespace específico para esta práctica y aplicamos los dos ficheros que acabamos de crear con los siguientes comandos:

```
k create ns practica5
k apply -f deployment.yaml -n practica5
k apply -f service.yaml -n practica5
```

```
[arianegutierrez@MacBook-Pro-de-Ariane practica5-angular % k get all -n practica5
```

NAME	READY	STATUS	RESTARTS	AGE
pod/angular-deployment-58bdd64667-rx9g8	1/1	Running	0	40s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/angular-service	NodePort	10.96.76.103	<none>	80:30001/TCP	34s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/angular-deployment	1/1	1	1	40s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/angular-deployment-58bdd64667	1	<u>1</u>	1	40s

## 6.0 EXPLICACIÓN DEL PROCESO DE DISEÑO E IMPLEMENTACIÓN DE CI/CD

A continuación, se muestra el documento de Jenkins creado para la implementación de CI/CD del proyecto de angular. Todos los pasos explicados anteriormente conforman cada paso del pipeline.

```
pipeline {
  agent any
  parameters {
    string(name: 'TAG', defaultValue: 'v1', description: 'Tag del proceso de CI/CD')
  }
  stages {
    stage('CI: Build node project') {
      steps {
        echo 'ng build'
      }
    }
    stage('CI: Test node project') {
      steps {
        echo 'ng test'
      }
    }
    stage('CI: Subir al repositorio git') {
      steps {
        echo 'git push -u origin master'
      }
    }
    stage('CD: Crear la imagen docker') {
      steps {
        echo 'docker build -t ariane10/angularapp:${params.TAG}'
      }
    }
    stage('CD: Subir a repositorio privado') {
      steps {
        echo 'docker push ariane10/angularapp:${params.TAG}'
      }
    }
    stage('CD: Desplegar la aplicación') {
      steps {
        echo 'kubectl apply -f deployment.yaml'
      }
    }
    stage('CD: Exponer el deployment') {
      steps {
        echo 'kubectl apply -f service.yaml'
      }
    }
  }
}
```

Este seria todo el flujo:

