



**Universidad de Guadalajara**  
**Centro Universitario de Ciencias Exactas e Ingenierías**

## **MicroProfile, Quarkus y Docker**

**Alumno:** Bryan De Anda Reyes

**Código de alumno:** 216195537

**Materia:** Computación Tolerante a Fallas (D06)

**Horario:** lunes y miércoles (11:00 – 13:00)

**Carrera:** INCO

**Profesor:** Michel Emanuel López Franco

## **Instrucciones**

El objetivo de esta actividad es realizar una investigación sobre MicroProfile, Quarkus y Docker, después de investigar la definición de estas herramientas, se generará un ejemplo utilizando Docker.

## **Desarrollo**

### **¿Qué es MicroProfile?**

MicroProfile especifica una colección de APIs y tecnologías Java EE que juntas forman la línea de base para crear Microservicios, que tiene como objetivo proporcionar la portabilidad de la aplicación en múltiples entornos de ejecución.

es una excelente manera de crear pequeñas aplicaciones de Java que se pueden implementar de forma rápida y sencilla en servicios como Azure Web Apps for Containers.

### **¿Qué es Quarkus?**

Quarkus es un marco de Java integral creado en Kubernetes para las compilaciones originales y las máquinas virtuales Java (JVM), el cual permite optimizar esta plataforma especialmente para los contenedores y para los entornos sin servidor, de nube y de Kubernetes.

Se diseñó para que funcione con las bibliotecas, los marcos y los estándares de Java conocidos, como Eclipse MicroProfile y Spring (ambos se presentaron como parte de una sesión en el evento Red Hat Summit 2020), así como también Apache Kafka, RESTEasy (JAX-RS), Hibernate ORM (JPA), Infinispan, Camel y muchos más.

La solución de inyección de dependencias de Quarkus se basa en la inyección de dependencias y contextos (CDI), e incluye un marco de extensión para ampliar las funciones y configurar, iniciar e integrar un marco en las aplicaciones. Dado que agregar una extensión es tan sencillo como incorporar una dependencia, puede optar por esa opción o utilizar las herramientas de Quarkus.

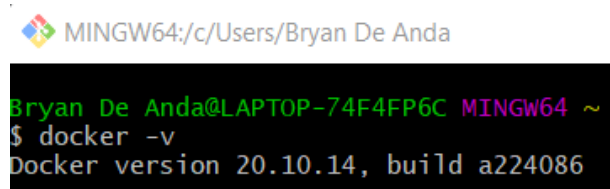
Además, proporciona la información correcta a GraalVM (una máquina virtual universal para ejecutar aplicaciones escritas en varios lenguajes, incluidos Java y JavaScript) para la compilación propia de las aplicaciones.

## ¿Qué es Docker?

Docker es una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.

## Ejemplo

Para el desarrollo de esta actividad, primero fue necesario realizar la instalación de docker de su sitio oficial, y para verificar que se instaló correctamente, desde el PowerShell se escribe la línea de comandos “docker -v” la cual nos mostrara la versión de Docker instalada.



```
MINGW64:/c/Users/Bryan De Anda  
Bryan De Anda@LAPTOP-74F4FP6C MINGW64 ~  
$ docker -v  
Docker version 20.10.14, build a224086
```

Ahora bien, para que docker funcione correctamente también es necesario instalar el subsistema de Windows para Linux (WSL) en la versión 2, si todo esto se hizo correctamente en la aplicación de docker de nuestro computador debe abrirse sin ningún inconveniente.

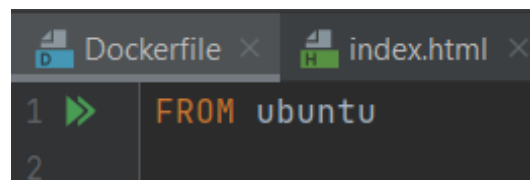
Hablando a grandes rasgos de lo que hace el ejemplo, es meter nuestra aplicación en un contenedor, que en este caso la aplicación es un código HTML que muestra una portada con la información del curso y lo que va a hacer este contenedor es empaquetar

la aplicación con todas las partes necesarias, por lo que, todas las bibliotecas y demás dependencias se empaquetaran.

El primer paso para nuestra aplicación es en un archivo Docker que tiene que ser llamado “Dockerfile”, y para generar el contenedor se necesitan tres cosas indispensables, la primera es el Dockerfile que la se menciono, la segunda es una imagen docker y por ultimo el contenedor en si.

Lo que hace el Dockerfile es generar imágenes docker, la imagen es una plantilla para ejecutar contenedores y el contenedor es el proceso en ejecución real donde tenemos nuestro proyecto empaquetado.

Lo primero que tenemos que hacer es definir en el Dockerfile el sistema operativo con el que se va a trabajar, por esta razón, en este ejemplo se importo ubuntu para manejarlo linux.



Después, se genera la imagen, para esto, en la terminal se escribe el comando “docker build -t ubuntu\_img” que generará la imagen con la información guardada en el archivo “Dockerfile”.

En la siguiente imagen podemos apreciar como se esta generando la imagen deseada.

```
PS C:\Users\Bryan De Anda\PycharmProjects\pythonProject3> docker build -t ubuntu_img .  
[+] Building 4.7s (2/3)  
=> [internal] load build definition from Dockerfile  
=> => transferring dockerfile: 50B  
=> [internal] load .dockerignore  
=> => transferring context: 2B  
=> [internal] load metadata for docker.io/library/ubuntu:latest
```

También se muestra como terminó de crear la imagen satisfactoriamente.



Por otro lado también contamos con el comando `docker ps -a`, el cual nos da acceso a los contenedores creados y nos señala su estatus, ya sea si se encuentra en ejecución o detenido.

```
PS C:\Users\Bryan De Anda\PycharmProjects\pythonProject3> docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3d67e963d7af	38e4aaa93203	"bash"	8 minutes ago	Exited (137) 29 seconds ago		zen_haibt
d714d0ebe38c	apache/airflow:2.2.5	"/usr/bin/dumb-init ..."	2 days ago	Exited (137) About an hour ago		airflow-docker-airflow-triggerer-1
f4f52b99d4c6	apache/airflow:2.2.5	"/usr/bin/dumb-init ..."	2 days ago	Exited (137) About an hour ago		airflow-docker-airflow-worker-1
7c446e15466c	apache/airflow:2.2.5	"/usr/bin/dumb-init ..."	2 days ago	Exited (137) About an hour ago		airflow-docker-flower-1
26746ee0430c	apache/airflow:2.2.5	"/usr/bin/dumb-init ..."	2 days ago	Exited (137) About an hour ago		airflow-docker-airflow-webserver-1
e27267a814e3	apache/airflow:2.2.5	"/usr/bin/dumb-init ..."	2 days ago	Exited (137) About an hour ago		airflow-docker-airflow-scheduler-1
23360e7d4880	apache/airflow:2.2.5	"/bin/bash -c 'funct...	2 days ago	Exited (0) 46 hours ago		airflow-docker-airflow-init-1
c37d61672c98	postgres:13	"docker-entrypoint.s..."	2 days ago	Exited (0) About an hour ago		airflow-docker-postgres-1
4796693d2be3	redis:latest	"docker-entrypoint.s..."	2 days ago	Exited (0) About an hour ago		airflow-docker-redis-1

En la siguiente imagen se muestra una parte del código HTML.

```
1020 </head>
1021
1022 <body lang=ES-MX link="#0563C1" vlink="#954F72" style='tab-interval:35.4pt;
1023 word-wrap:break-word'>
1024
1025 <div class=WordSection1>
1026
1027 <p class=MsoNormal align=center style='text-align:center'><b><span
1028 style='font-size:18.0pt;line-height:105%;font-family:"Times New Roman",serif'>Universidad
1029 de Guadalajara <o:p></o:p></span></b></p>
1030
1031 <p class=MsoNormal align=center style='text-align:center'><b><span
1032 style='font-size:18.0pt;line-height:105%;font-family:"Times New Roman",serif'>Centro
1033 Universitario de Ciencias Exactas e Ingenierías<o:p></o:p></span></b></p>
1034
1035 <p class=MsoNormal align=center style='text-align:center'><b><span
1036 style='font-size:18.0pt;line-height:105%;font-family:"Times New Roman",serif'><o:p>&nbsp;</o:p></span></b></p>
1037
1038 <p class=MsoNormal align=center style='text-align:center'><b><span
1039 style='font-size:18.0pt;line-height:105%;font-family:"Times New Roman",serif'><o:p>&nbsp;</o:p></span></b></p>
1040
1041 <p class=MsoNormal align=center style='text-align:center'><b><span
```

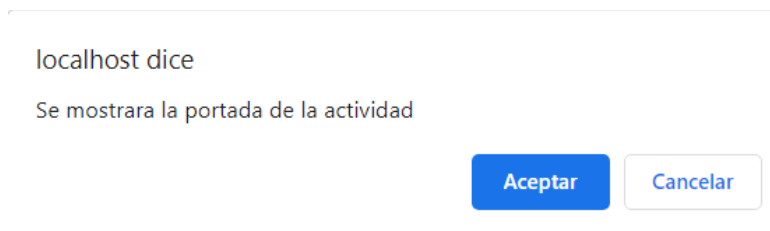
Para el contenedor que empaquetara el código HTML, fue necesario escribir el siguiente código en el Dockerfile para generar la imagen adecuada, definiendo en qué sistema operativo se utilizará y demás aspectos necesarios.

```
FROM nginx:1.19.0-alpine
COPY . /usr/share/nginx/html
```

Después de la creación de la imagen que se le asignó el nombre de “nginx\_img” se ejecuta el siguiente comando en la terminal “docker run -it -p 80:80 NAME\_IMAGE”, indicando los puertos adecuados.

```
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS C:\Users\Bryan De Anda\PycharmProjects\pythonProject3> docker run -it -p 80:80 nginx_img
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
```

Si intentamos ingresar en un navegador con la dirección “localhost:80”, nos mostrará primero el siguiente mensaje.



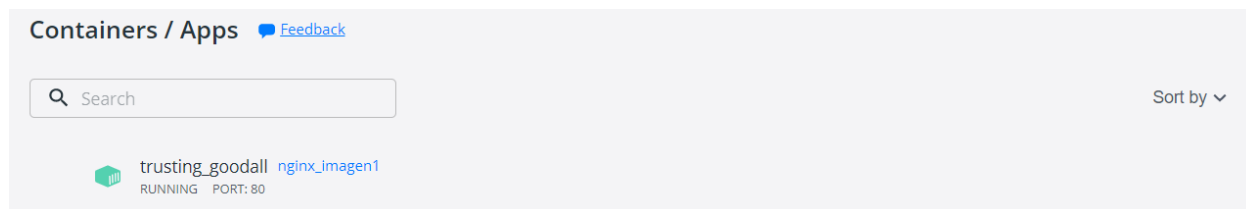
Después de seleccionar “Aceptar” o “cancelar”, podremos observar la portada con información del curso.

**Universidad de Guadalajara**  
**Centro Universitario de Ciencias Exactas e Ingenierías**

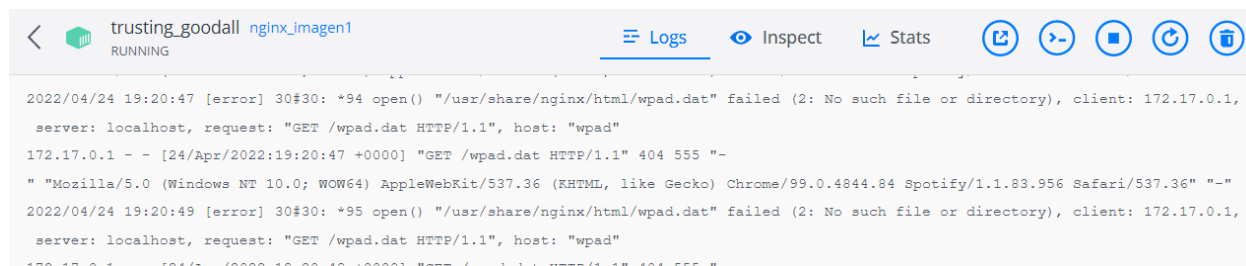
## **MicroProfile, Quarkus y Docker**

**Departamento** de Ciencias Computacionales  
**Calendario:** 2022-A  
**Materia:** Computación Tolerante a Fallas (D06)  
**Horario:** lunes y miércoles (11:00 a 13:00)  
**NRC:** 179961  
**Profesor:** Michel Emanuel López Franco  
**Alumno:** Bryan De Anda Reyes  
**Código de alumno:** 216195537  
**Carrera:** Ingeniería en Computación (INCO)

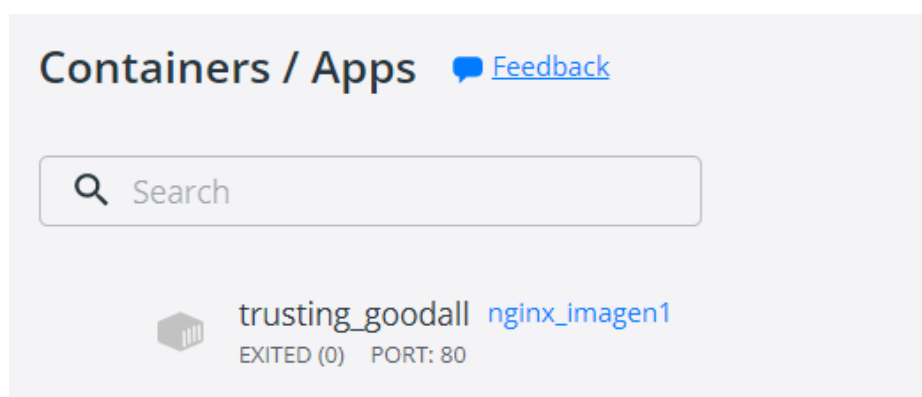
Si entramos directamente a la aplicación de Docker, podemos encontrar los diferentes contenedores que tengamos, ya sea si se encuentran en ejecución o están detenidos, en este caso podemos ver que el que acabamos de generar se encuentra corriendo.



Ahora, si entramos al contenedor podemos apreciar diversa información, como por ejemplo los tiempos en los que estuvo en ejecución.



Por otro lado, la ejecución de este contenedor o cualquiera, puede detenerse desde la línea de comandos o directamente de la aplicación de Docker presionando el botón con el cuadrado negro.



Enlace al código en el repositorio: <https://github.com/BryanDeAnda/Tolerancia-a-fallas-con-MicroProfile-Quarkus-y-Docker.git>



## Conclusión

Esta actividad me pareció muy interesante de realizar al investigar acerca sobre estas herramientas que en lo personal no las conocía y pude darme cuenta de todos los beneficios que estas nos brindan. En cuanto al ejemplo, intenté realizar uno utilizando las diferentes herramientas mencionadas en la actividad, pero se me presentaron muchos problemas, los cuales no me permitían ejecutar el código correctamente, por esta razón, opté por realizar un ejemplo utilizando Docker.

El ejemplo realizado me pareció bastante entretenido e interesante de llevar a cabo, ya que al agregar nuestros códigos en contenedores puede llegar a traer muchos beneficios.

Probablemente en un futuro estas herramientas puedan llegarme a ser de gran utilidad, por eso, considero que es de gran ayuda conocerlas e ir comprendiendo su funcionamiento.

## Bibliografía

C. (2022, 8 abril). *Pasos de instalación manual para versiones anteriores de WSL*. Microsoft Docs. Recuperado 2022, de <https://docs.microsoft.com/es-mx/windows/wsl/install-manual#step-4---download-the-linux-kernel-update-package>

*Contenedores de Docker | ¿Qué es Docker?* | AWS. (s. f.). Amazon Web Services, Inc. Recuperado 2022, de <https://aws.amazon.com/es/docker/>

MicroProfile. (2021, 14 diciembre). *Home*. Recuperado 2022, de <https://microprofile.io/>

*¿Qué es Quarkus?* (s. f.). RedHat. Recuperado 2022, de <https://www.redhat.com/es/topics/cloud-native-apps/what-is-quarkus>