**Universidad de las Américas**

**Facultad de Ingeniería y Ciencias Aplicadas**

**Ingeniería en Sistemas de Computación e Informática**

**Implementación de una guía de buenas prácticas de Integración Continua en proyectos de desarrollo en el área de tecnología de una Empresa Privada.**

Edison P. Mosquera León, Carlos X. Ruiz Águila, David F. Martínez Bravo

**DOCKER**

**Objetivo:**

Ayudar a los usuarios de la empresa a saber para qué sirve Docker y Dockerfile en   
, al conocer para que funcione y cuál es la función de este contenedor los usuarios d ellos próximos proyectos, sabrán poder implementarlo de mejor manera o explicarlo para otros equipos.

Docker: La mejor explicación que podemos dar de Docker es una herramienta para empaquetar software en contenedores que va aislado es decir que puede funcionar independientemente del sitio donde este, es decir por ejemplo una máquina virtual de java que ejecuta esta mismas en millones de dispositivos.

Para implementarlo con nuestros proyectos debemos tener en cuenta que cada contenedor esta estandarizado, generalmente una aplicación está en un solo contenedor y todos estos contenedores están dentro de un gestor para que funcionen. Docker se encarga de crear estos contenedores para que se puedan usar por alguien más. Siempre lo que este dentro del contenedor debe estar ordenado y hay que aprovechar al máximo el tamaño del contenedor.

La razón por la que aprendemos Docker es para que la aplicación sea productiva y es muy probable que actualmente la aplicación que se integre termine corriendo en una nube. Al estar en la nube puede tener múltiples servicios y el que más nos interesaría es el CaaS.

CaaS es para desplegar, gestionar contenedores y usualmente va con un gestor de alguna nube. Lo principal de esto es crear el contenedor ligero y en Docker la acción se llama dockerizar.

**DockerFiles**

Para hablar bien de los DockerFiles, los Docker build es un comando que se ejecutara como un instructivo del archivo. Esta manda todo el contexto del directorio al Daemon, esto es muy importante.

El Dockerfile es un archivo o documento que contiene comandos que se ejecutan para armar una imagen. Mientras se crea mediante los comandos del Docker build que sigue las instrucciones de este mismo. Es mejor colocar como practica en un directorio limpio y agregar los archivos necesarios en ese directorio cuando sea necesario.

Ahora el comando dockerbuild crea una imagen que sigue las instrucciones del dockerfile que se encuentra en el directorio presente. La creación de la imagen es ejecutada por el deamon de Docker.

Lo que corre las instrucciones en un dockerfile line por línea es el Docker Daemon y los resultados se muestran en la pantalla. En cada instrucción es ejecutada en nuevas imágenes, los resultados de las instrucciones por las muestras id de las imágenes permiten que Daemon haga una limpieza automática de las imágenes intermedias.

Docker daemon va creando imágenes intermedias mientras la creación de la imagen, si por alguna razón la creación de la imagen falla, ya sea por un comando mal digitado en el archivo, o lo que sea, cuando corregimos el Dockerfile, este no iniciará todo el proceso reiteradamente, sino, que hará uso de las imágenes intermedias, y continuará la creación en el punto donde falló.

El archivo debe esta nombrado Dockerfile

Aquí el formato de las líneas del dockerfile

* **MAINTAINER**

Esta instrucción nos permite configurar datos del autor que genera la imagen.

|  |
| --- |
| MAINTAINER <nombre> <Correo> |

* **ENV**

ENV tiene 2 formas:

Esta instrucción configura las variables de ambiente, estos valores estarán en los ambientes de todos los comandos que sigan en el DockerFile. Pueden por igual ser sustituidos en una línea.

Estos valores persistirán al momento de lanzar un contenedor de la imagen creada. Pueden ser sustituida pasando la opción –env en docker run.

|  |
| --- |
| ENV <key><valor> #variable única a un valor |
| ENV <key><valor> … #Múltiples variables a un valor |

* **ADD**

ADD tiene 2 formas:

Esta instrucción copia los archivos, directorios de una ubicación especificada en y los agrega al sistema de archivos del contenedor en la ruta especificada en.

|  |
| --- |
| ADD <fuente> ...<destino> |
| ADD [“fuente”, … “<destino>”] |

* **CMD**

CMD tiene tres formatos:

Solo puede existir una instrucción CMD en un DockerFile, si colocamos más de uno, solo el último tendrá efecto. El objetivo de esta instrucción es proveer valores por defecto a un contenedor. Estos valores pueden incluir un ejecutable u omitir un ejecutable que en dado caso se debe especificar un punto de entrada o ENTRYPOINT en las instrucciones.

|  |
| --- |
| CMD [“ejecutable”, “parámetro1”, “parámetro2”] # este es el formato de ejecución |
| CMD [“parámetro1”, “parámetro2”] # parámetro por defecto para punto de entrada |
| CMD comando parámetro1 parámetro2, modo Shell # Si queremos ejecutar un comando sin un shell, debemos expresar el comando en formato JSON y dar la ruta del ejecutable. Es el formato recomendado. CMD [“/usr/bin/service”, “httpd”, “start”] |

Si lo que queremos es que el mismo ejecutable corra todo el tiempo, lo que necesitamos es un punto de entrada ENTRYPOINT en combinación con CMD. En caso de pasarle un comando mediante docker run, este correrá en vez del especificado por CMD.

* **VOLUME**

Esta instrucción crea un punto de montaje con un nombre especificado y lo marca con un volumen montado externamente desde el host y otro contenedor. El valor puede ser pasado en formato JSON o argumento plano.

El comando docker run inicializará el nuevo contenedor con cualquier data que exista en la ubicación dentro de la imagen base.

|  |
| --- |
| VOLUME [“/var/tmp”] |
| VOLUME /var/tmp |

* **USER**

Esta instrucción configura el nombre de usuario a usar cuando se lanza un contenedor y para la ejecución de cualquier instrucción RUN, CMD o ENTRYPOINT. WORKDIR

|  |
| --- |
| WORKDIR ruta/de/trabajo # Esta instrucción tiene la capacidad de resolver variables de ambiente previamente configuradas mediante la instrucción. |

Esta instrucción configura el directorio de trabajo para cualquier instrucción RUN, CMD, ENTRYPOINT, COPY o ADD en un DockerFile. Puede ser usada varias veces dentro de un DockerFile. Si se da una ruta relativa, esta será la ruta relativa de la instrucción WORKDIR anterior.

**Especificando en la empresa:**

Revisando por cada pipeline creado se debe conocer a fondo para que sirve cada line puesta en el Dockerfile.

El objetivo de este documento es guiar a la empresa y sus equipos a integrar proyectos dentro de Azure DevOps por medio del ejemplo.

Tenemos que ingresar a Azure DevOps y tener lista nuestra organización. Esta organización tendrá los proyectos que realicemos, los ítems de trabajo y los pull requests. Puedes crear cualquier proyecto, pero tienes que primero saber para qué vas a usar Azure DevOps y como esta herramienta puede facilitar tu proyecto, en este caso, estamos implementando un programa basado en programación.

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

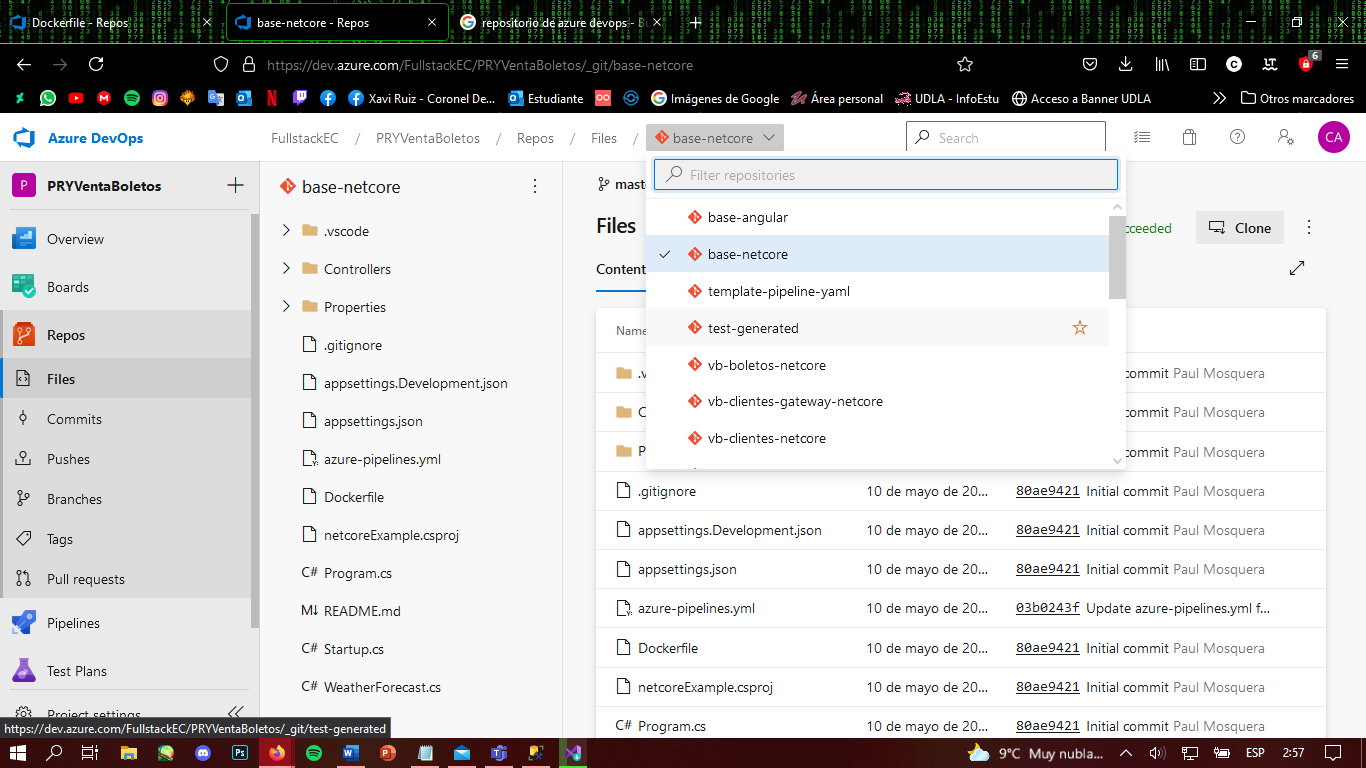
Luego al ingresar al proyecto que ya hayas creado o crearon tus compañeros, podrás visualizar los repositorios, los pipelines, los test Plans y los artefactos.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ahora nos vamos a centrar en la parte del repositorio, donde encontramos los archivos de código fuente, histórico de cambios, ramas, tags, entre sus opciones principales como los pipelines que necesitamos crear para nuestro proyecto.Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Al menos cada uno de estos archivos en el Repos contiene un Docker file. A continuación, te mostraremos que contiene este archivo y cuáles son las funciones de sus códigos a fondo para que tengas una idea de cómo funciona ya que anteriormente explicamos que era Docker y los Dockerfiles.  


Estos archivos en especial están ubicados en la sección de Pipelines, los pipelines contienen el archivo Dockerfile y estos funcionan dependiendo de la optimización de código que nosotros les demos conjuntamente con otras herramientas como SonarCloud.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Bien, nosotros encontramos el documento Dockerfile en el desglose automático de los archivos dentro de Repos, el nombre se visualizará como tal lo mencionamos y contiene varios comandos particulares. Este Dockerfile pertenece solo a este proyecto.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Al seleccionar el archivo Dockerfile, en este caso, como podemos observar en el vb-boletos-netcore, empieza usando el comando:

**FROM**

El comando FROM ejecuta una imagen

|  |
| --- |
| FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS build  FROM alpine:3.9.4 |
| FROM indica la imagen base que va a utilizar para seguir futuras instrucciones. Buscará si la imagen se encuentra localmente, en caso de que no, la descargará de internet.  FROM <imagen>  FROM <imagen>:<tag> #El tag es opcional, en caso de que no la especifiquemos, el daemon de docker asumirá latest por defecto. |

**RUN**

Corre cualquier comando en una capa nueva

|  |
| --- |
| RUN mkdir -p /home/app  RUN dotnet restore  RUN dotnet publish -c Release -r alpine-x64 --self contained true /p:PublishTrimmed=true -o ./publish  # Add some libs required by .NET runtime  RUN apk add --no-cache libstdc++ libintl icu |
| Esta instrucción ejecuta cualquier comando en una capa nueva encima de una imagen y hace un commit de los resultados. Esa nueva imagen intermedia es usada para el siguiente paso en el Dockerfile.  El modo ejecución nos permite correr comandos en imágenes bases que no cuenten con /bin/sh, nos permite además hacer uso de otra shell si así lo deseamos.  RUN <comando> #esta es la forma shell, /bin/sh -c  RUN [“ejecutable”, “parámetro1”. “parámetro2] # este es el modo ejecución |

* **Workdir**

Cambia el directorio por defecto

|  |
| --- |
| WORKDIR /home/app |
| El comando WORKDIR se utiliza para definir el directorio de trabajo de un contenedor Docker en cualquier momento. El comando se especifica en el Dockerfile.  Si el comando WORKDIR no está escrito en el Dockerfile, el compilador de Docker lo creará automáticamente. Por tanto, se puede decir que el comando ejecuta mkdir y cd implícitamente. |

Siempre es útil dar instrucciones o dividir en secciones comentadas para ser más ordenados y encontrar lo que buscamos con más rapidez.

**Copy**

Se menciona cuando se copia archivos o directorios compilados.

|  |
| --- |
| COPY \*.csproj ./  # Copy everything else and build  COPY . ./  COPY --from=build /home/app/publish/ /home/app/ |
| Si desea copiar archivos y directorios dentro de un contenedor Docker desde su máquina local, puede usar la instrucción **COPY** dentro de su *Dockerfile*  COPY <src-path> <destination-path> |

**Expose**

El comando EXPOSE que funciona indicando los puertos en los que va al contenedor.

|  |
| --- |
| EXPOSE 80  EXPOSE 443 |
| Esta instrucción le específica a docker que el contenedor escucha en los puertos especificados en su ejecución. EXPOSE no hace que los puertos puedan ser accedidos desde el host, para esto debemos mapear los puertos usando la opción -p en docker run. |

Recordemos que cada aplicación debe ser guardada en un contenedor, en este caso usamos Docker. Y en la sección de copia podemos ver un ENTRYPOINT

**ENTRYPOINT**

Especifica el ejecutable que usará el contenedor, en este caso en una url.

|  |
| --- |
| ENTRYPOINT ["./vb-boletos-netcore", "--urls", "http://0.0.0.0:80"] |
| ENTRYPOINT tiene 2 formas:  Cualquier argumento que pasemos en la línea de comandos mediante docker run serán anexados después de todos los elementos especificados mediante la instrucción ENTRYPOINT, y anulará cualquier elemento especificado con CMD. Esto permite pasar cualquier argumento al punto de entrada. |
| ENTRYPOINT [“ejecutable”, “parámetro1”, “parámetro2”] #forma de ejecución |
| ENTRYPOINT comando parámetro1 parámetro2 #forma shell |

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Esto es solo un ejemplo de lo que contiene los comandos del Dockerfile en unos de nuestros pipelines. La mayoría de estos tienen las mismas instrucciones, pero cumple diferente funciones.