

Manual Técnico

Para el despliegue de la aplicación es necesario la configuración de los siguientes archivos.

Configuración Dockerfile



Un DockerFile es un documento de texto que contiene todos los comandos que queramos ejecutar en la línea de comandos para armar una imagen. Esta imagen se creará mediante el comando `docker build` que irá siguiendo las instrucciones. Se muestran las instrucciones utilizadas dentro del archivo.

FROM

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.

Sintaxis

FROM <imagen>

FROM <imagen>:<tag>

- Ejemplo
 - FROM centos:latest

El tag es opcional, en caso de que no la especifiquemos, el daemon de docker asumirá latest por defecto.

MAINTAINER

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

Sintaxis

MAINTAINER <nombre> <Correo>

- Ejemplo
 - MAINTAINER Jason Soto "jason_soto@jsitech.com"

RUN

RUN tiene 2 formatos:

- RUN <comando>, esta es la forma shell, `/bin/sh -c`
- RUN ["ejecutable", "parámetro1", "parámetro2"], este es el modo ejecución

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, ej: RUN ["/bin/bash", "-c", "echo prueba"]

ENV

ENV tiene 2 formas:

- ENV <key><valor> , variable única a un valor
- ENV <key><valor> ... , Múltiples variables a un valor
-

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. Ej: `docker run --env <key>=<valor>`

ADD

ADD tiene 2 formas:

- ADD <fuente> ..<destino>
- ADD ["fuente", ... "<destino>"]

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

- Ejemplo:
 - ADD ./prueba.sh /var/tmp/prueba.sh

EXPOSE

Esta instrucción le especifica 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

- Ejemplo:
 - EXPOSE 80 443
 - docker run centos:centos7 -p 8080:80

CMD

CMD tiene tres formatos:

- 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

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.

Ejemplos:

1. Modo Shell
 - CMD echo "Esto es una prueba"

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"]

A continuación, se muestra la configuración de nuestro Dockerfile



```
1 FROM node:14.16.0
2
3 WORKDIR /usr/src/app
4
5 COPY package*.json ./
6
7 RUN npm install
8
9 COPY . .
10
11 EXPOSE 5010
12
13 CMD [ "npm", "start" ]
```


Configuración de Kubernetes:

Para el despliegue utilizaremos kubernetes y la configuración la haremos por medio de archivos YAML

Archivo de configuración.

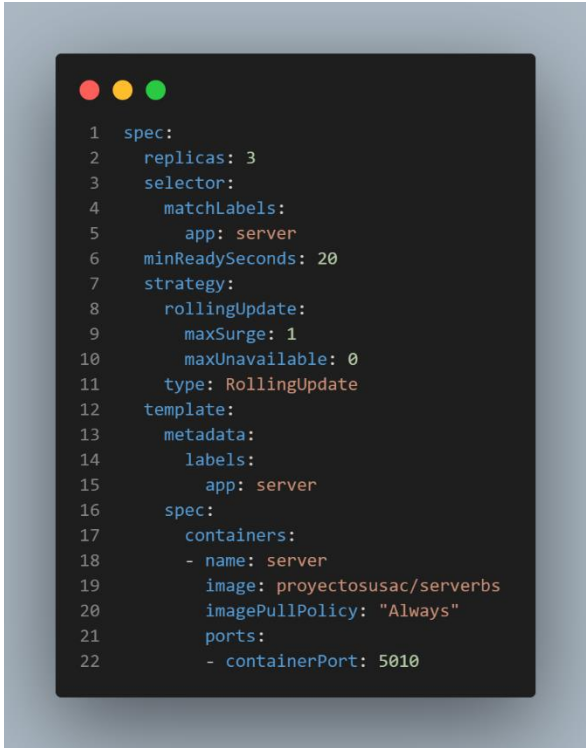
El archivo se divide en tres partes

1. **Metadata:** información de los componentes que se están creando



```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: server
```

2. **Especificaciones:** En esta sección estará cada una de las configuraciones necesarias que el componente desea solicitar. Dentro de la sección de especificaciones se encuentran los atributos específicos para el tipo de componente que se esta creando.



```
1  spec:
2    replicas: 3
3    selector:
4      matchLabels:
5        app: server
6    minReadySeconds: 20
7    strategy:
8      rollingUpdate:
9        maxSurge: 1
10       maxUnavailable: 0
11     type: RollingUpdate
12  template:
13    metadata:
14      labels:
15        app: server
16    spec:
17      containers:
18      - name: server
19        image: proyectosusac/serverbs
20        imagePullPolicy: "Always"
21        ports:
22        - containerPort: 5010
```

- 3. Estados:** Esta sección será generada automáticamente y agregada al documento por Kubernetes. La forma que funciona es que Kubernetes compara el estado deseado del componente y el estado actual, de este modo y no son iguales Kubernetes sabe que hay algo que debe de arreglar o configurar para que ambos estados se encuentren igual. Esta es la base de la autocuración o autoconfiguración.

A continuación, se muestran los comando utilizados para el RollBack utilizando la política Rolling Update

- **docker build -t proyectosusac/sopes2bs:1.2 .**
- **docker push proyectosusac/sopes2bs:1.2**
- **kubectl rollout history deploy frontend**
 - para ver las versiones del deployment
- **kubectl set image deployment frontend frontend=proyectosusac/sopes2bs:1.0**
 - para crear la imagen
- **kubectl describe deployment frontend**
 - Para verificar la imagen
- **kubectl rollout undo deployment frontend --to-revision=1**
 - Para regresar a una versión anterior