Integración continua Front

1) Preparar ambientes en front.

Se crean 4 ambientes (environment) dentro de la MFSicof (Local, Dev, QA, Prod). Dentro de ellos se ejecutan los comandos correspondientes de ambiente:

Dev: npm run build:dev QA: npm run build:qa Prod: npm run build:prod Local es el definido por defecto si no se le coloca un ambiente y se deja tal cual:

Local: npm run start o npm start

2) Preparar Docker. Construir la imagen Docker:

- docker build –build-arg TYPE=qa -t mfsicof .

Nota: TYPE cambia en los pipelines de despliegue en Jenkins dependiendo del ambiente que apunte. Las opciones son dev, qa y prod.

Ejecutar el contenedor Docker:

- docker run -d -p 80:80 mfsicof

Nota: Se define el puerto, donde el primero es el externo y el segundo es el interno del contenedor.

Al hacer Docker Build se ejecuta la siguiente receta:

```
# Usa una imagen base de Node.js

FROM node:20-alpine AS build-step

# Establecer el límite de memoria para Node.js

ENV NODE_OPTIONS=–max_old_space_size=4096

ARG TYPE

# Crea un directorio /app en la imagen

RUN mkdir -p /app

# Establece el directorio de trabajo

WORKDIR /app

# Copia los archivos de tu proyecto al directorio de trabajo

COPY package.json /app

# Instala las dependencias del proyecto

RUN npm install

# Copia el resto de los archivos de tu proyecto al directorio de trabajo
```

COPY . /app

Construye la aplicación Angular

RUN npm run build:${TYPE}

Configura la imagen de producción de Nginx

FROM nginx:alpine

Copia los archivos generados de la compilación de Angular a la carpeta de Nginx

COPY –from=build-step /app/dist/mfsicof /usr/share/nginx/html

Expone el puerto 80 para que se pueda acceder a la aplicación desde el navegador

EXPOSE 80

Comando para iniciar Nginx cuando se ejecute el contenedor

CMD [“nginx”, “-g”, “daemon off;”]

3) Crear proyecto en GitLab.

Se crea un proyecto en GitLab con el siguiente repositorio:
http://10.1.140.120/ada-microservices-ecosystem/frontends/mfsicof.git

Se crea un webhook para el frontend con la siguiente URL:

Para clonar el proyecto una vez tengas permisos de Git:

code git clone http://10.1.140.120/ada-microservices-ecosystem/frontends/mfsicof.git Para instalar el ambiente se recomienda consultar la guía de primeros pasos.

Se recomienda consultar la guía de flujo Git.

4) Preparar Jenkins.

Adicional a la configuración de contenedores para Jenkins es necesario instalar jq. jq es una herramienta de línea de comandos para procesar JSON. Asegúrate de que esté instalada en el sistema donde Jenkins está ejecutando el script. En sistemas basados en Debian/Ubuntu, puedes instalar jq con:

1. sudo apt-get install jq
2. Configurar el Script en Jenkins:
3. Accede a la configuración del proyecto en Jenkins.
4. En la sección “Build”, añade o edita el paso “Execute shell”.
5. Copia y pega el script anterior en el campo de texto del shell.
6. Guardar y Probar:
7. Guarda la configuración del proyecto.

5.1 Acceder a Jenkins: Inicia sesión en tu instancia de Jenkins.

5.2 Crear un nuevo proyecto: Ve a “New Item”. Selecciona “Freestyle project” y da un nombre a tu proyecto. Haz clic en “OK”.

5.3 Configurar el repositorio Git: En la sección “Source Code Management”, selecciona “Git”. Ingresa la URL de tu repositorio GitLab y las credenciales necesarias.

5.4 Configurar el webhook de GitLab:

- En GitLab, ve a tu proyecto.
- Navega a “Settings” > “Webhooks”.
- Añade una nueva URL de webhook apuntando a tu Jenkins (e.g., *http://your-jenkins-url/gitlab-webhook//). * Selecciona los eventos que deseas que disparen el webhook, como “Push events”. **5.5 Añadir un paso de ejecución de shell:** En la sección “Build”, haz clic en “Add build step” y selecciona “Execute shell”. Copia y pega el siguiente script. # Definir la URI del host Docker DOCKER_HOST_URI=“tcp:*172.17.0.1:2375”

Exportar la variable DOCKER_HOST

export DOCKER_HOST=$DOCKER_HOST_URI

Extraer la versión y el nombre del archivo package.json

TAG_NAME=$(jq -r '.version' package.json) NAME=$(jq -r '.name' package.json)

Modificar el nombre para insertar un guion después de 'mf'

CONTAINER_NAME=$(echo $NAME | sed 's/^mf/mf-/')

Quitar el guion del nombre del repositorio

REPOSITORY_NAME=“ecosystemuser/${NAME}”

Imprimir los nombres para visualización

echo “REPOSITORY_NAME: $REPOSITORY_NAME” echo “CONTAINER_NAME: $CONTAINER_NAME”

Construir la imagen Docker con el argumento de construcción

docker build –no-cache –build-arg TYPE=prod -t $REPOSITORY_NAME:$TAG_NAME .

Etiquetar la imagen con latest

docker tag $REPOSITORY_NAME:$TAG_NAME $REPOSITORY_NAME:latest

Empujar la imagen con la etiqueta latest a Docker Hub

docker push $REPOSITORY_NAME:$TAG_NAME docker push $REPOSITORY_NAME:latest

if [“$(docker ps -aq -f name=$CONTAINER_NAME)”]; then

```
  docker stop $CONTAINER_NAME
  docker rm -fv $CONTAINER_NAME
```

fi

Ejecutar el nuevo contenedor con la imagen “latest”

docker run -d -p 8095:80 –name $CONTAINER_NAME $REPOSITORY_NAME:latest

←Regresar