El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

CICLO IV:

Desarrollo de Aplicaciones Web

MisionTIC2022

El futuro digital es de todos | MinTIC

UNIVERSIDAD DEL NORTE
Vigilada Mineducación

Sesión 17:
Desarrollo de Aplicaciones Web

Manejo de Contenedores

MisionTIC2022

Objetivos de la sesión

Al finalizar esta sesión estarás en capacidad de:

1. Diseñar un sistema de gestión de aplicaciones web a través del concepto de contenedores.
2. Configurar contenedores para realizar el despliegue de aplicaciones web.

Despliegue

- Hoy en día con las aplicaciones web modernas no se requiere de que un servidor contenga todos los archivos de la interfaz web o del Front-end.
- Por otro lado, lo que actualmente se utiliza son almacenamientos de archivos estáticos como los buckets de AWS, los blob storage de Azure, entre otros.
- Esto hace que el cliente descargue los archivos para que el mismo genere la interfaz web tomando en cuenta que nuestra aplicación es una Single Page Application (SPA).
- Frameworks como React, o Angular generan este tipo de web apps que no recargan la página web.

Despliegue

- Es posible que se requiera que la interfaz web cuente con meta-elementos en el documento HTML.
- Por ejemplo, esto puede ser debido a que nuestra aplicación es un e-commerce el cual queremos que aparezca en la barra de búsqueda de los posibles usuarios en su navegadores web.
- Esto con el fin de atraer usuarios y ganar visibilidad.
- A este proceso de aparecer en los resultados de búsqueda de los navegadores web se le conoce como Search Engine Optimization (SEO).
- Para incluir estos meta-elementos debemos generar nuestra aplicación del lado del servidor.

Despliegue

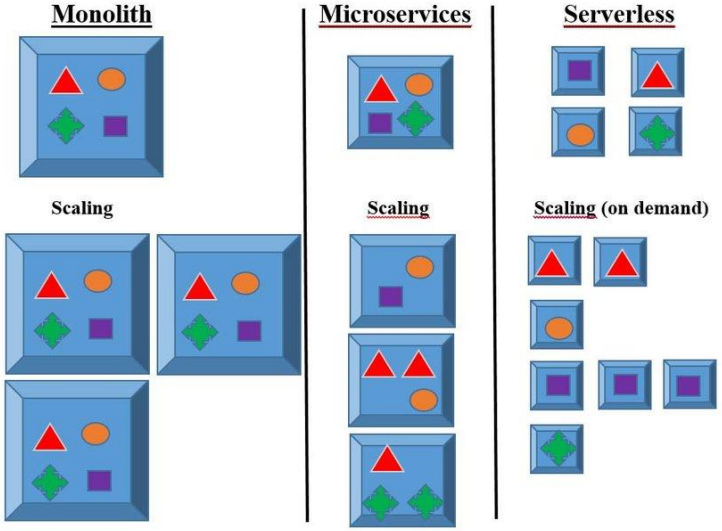
- Esto se debe a que estos meta encabezados no pueden ser generados por el cliente desde los resultados de la búsquedas.
- Esto es obvio puesto que en los resultados de búsquedas no se descarga ninguna clase de archivo, esto ocurre únicamente al acceder a la URL de la SPA.
- Por otro lado, las páginas generadas del lado del servidor o Server Side Rendered (SSR), generan estos meta-elementos incluso para los bots que se encargan de visitar sitios web.
- Esto nos permite tener SEO y miniaturas al compartir un link o que incluso contenido de nuestra aplicación web tenga su propia miniatura, entre otras cosas.

Despliegue

- Cabe resaltar que también existen alternativas para generar sitios por medio de SSR en el mundo de Node.js como lo sería Next.js o Nuxt.js.
- Esto consiste en tener un servidor que se encargue de generar el contenido de la vista que estamos solicitando.

Despliegue

- Por otro lado, también tenemos el despliegue del Back-end. Donde nuestra aproximación al problema del despliegue varía según nuestra arquitectura:



Tomado de Alex Kaplunovich en Research Gate

- **Monolito**: Todo escala de manera lineal.
- **Microservicios**: Escalado por servicio o módulos con los que cuente nuestro Back-End.
- **Serverless**: Escalado por uso de cada recurso en nuestro Back-End.

Despliegue

- Por esto mismo, nuestra solución puede variar según la complejidad de nuestra aplicación.
- Si sabemos que no vamos a tener necesidad de escalar nuestra aplicación y tenemos requerimientos sencillos, normalmente escogemos el monolito ya que es la solución más fácil de implementar.
- Por otro lado, si nos preocupa escalar correctamente nuestra aplicación, entonces debemos decantarnos por microservicios o serverless.

Despliegue

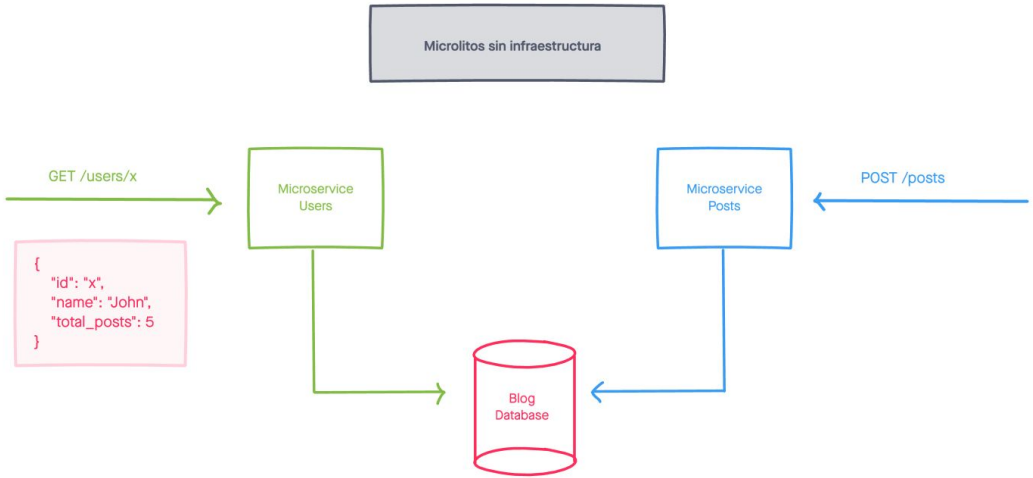
- La filosofía de microservicios es popular ya que busca dividir para conquistar.
- Esto nos permite encapsular nuestros requerimientos en módulos los cuales convertiremos en microservicios.
- La estrategia con la que se decide la división de cada módulo varía según los requerimientos del proyecto en cuestión.
- Por lo general, se utilizan los módulos a nivel de recursos como criterio de división.

Despliegue

- Finalmente, serverless lo que busca es dividir cada recurso de forma única para poder realizar su escalado de manera independiente según la demanda en tiempo real de los mismos.
- Estos desarrollos, por lo general son los que más se diferencian a nivel de framework ya que solo se usan frameworks en específico como lo sería serverless.
- Para estos despliegues normalmente se utilizan tecnologías como:
  - AWS Lambda.
  - Azure Serverless.
  - Google Cloud Functions.
  - Entre otros.

Despliegue

- Para este caso tomaremos en cuenta un despliegue con la metodología de microservicios.
- Como podemos detallar, tenemos varios servicios por máquina o nodo de lo que vendría siendo nuestro Back-End:



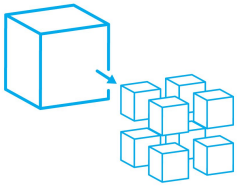
Tomado de [mangelsnc@Medium]

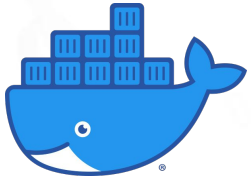
Despliegue

- Donde es posible que en una misma máquina se tengan varios micro servicios como lo son el de usuarios y posts en el ejemplo mostrado.
- Esto nos lleva a la duda de realmente cómo estructurar ambos microservicios para que sus entornos de ejecución no entren en conflicto entre sí.
- Para esto se propone la idea o el concepto de contenedores.

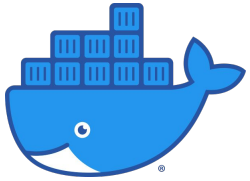
Contenedores

- El concepto de contenedores, nace de ofrecer un entorno de ejecución donde un conjunto de aplicaciones trabajen en entre ellas sin entrar en conflicto.
- Así mismo, se pueden generar, instalar y ejecutar todas las dependencias de este entorno sin que afecte directamente a otra aplicación en la máquina en cuestión.
- En pocas palabras, generamos un entorno de ejecución virtual con las necesidades exactas que requiere nuestra aplicación.
- Para la creación de estos contenedores nos apoyaremos en la tecnología Docker.



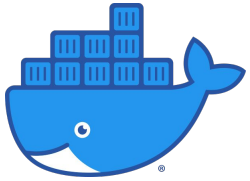


Contenedores - Docker



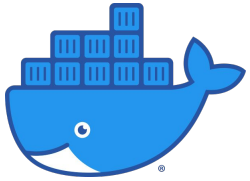
- Docker es una plataforma que se encarga de ofrecernos un entorno virtualizado donde podemos interactuar con contenedores.
- Con Docker podremos:
  - Definir la configuración de nuestro microservicio.
  - Componer un grupo de servicios y desplegarlos en conjunto.
  - Compartir nuestros conjuntos como un contenedor o imagen que es el concepto con el cual Docker las maneja.

Contenedores - Docker



- Es importante tener en cuenta las siguientes guías de instalación de Docker para su sistema operativo correspondiente:
  - Windows.
  - Mac.
  - Linux.
- Para los usuarios Windows es importante tener instalado Windows Subsystem for Linux o WSL.

Contenedores - Docker



- Por otro lado, Docker nos permite definir un archivo por cada microservicio que contemplemos, para optimizar lo que sería la creación de su contenedor.
- Este archivo se conoce como **`Dockerfile`**.
- Así mismo, Docker nos permite definir otro archivo de configuración donde especificamos el funcionamiento de nuestro contenedor y con qué otros contenedores o servicios está relacionado.
- Este archivo se le conoce como **`docker-compose.yml`**.

Contenedores - Kubernetes

- Kubernetes es una plataforma que nace de la necesidad de organizar nuestros contenedores de Docker.
- Fue desarrollada y es mantenida por Google.
- Este nos permite orquestar a los contendores para realizar tareas de monitores con ellos y tener un mejor control sobre los mismos.
- Es una herramienta flexible ante cualquier entorno de despliegue, la cual se considera como el siguiente paso a la construcción de nuestra aplicación.
- Para más detalles, ver este artículo en medium.

Node - Docker

- Para hacer una corta demostración de cómo trabajar en conjunto con tecnologías como lo son Docker y Node, estaremos utilizando el proyecto realizado en la sesión 15.
- Los cambios realizados sobre el repositorio los encontraran sobre la rama “add-docker”.
- Para esto, primero empezaremos tomando en cuenta que estaremos utilizando babel.
- Esto es importante, ya que esto implica que nuestro servidor o flujo de despliegue ha de compilar nuestro código para ser compatible con diferentes versiones de node.
- Esto con el fin de tener la mejor experiencia como desarrollador contando con versiones más recientes de JS en nuestras máquinas a diferencia de la nuestros contenedores.

Node - Docker

- Primero agregaremos el archivo “Dockerfile” en nuestro directorio raíz:

```
FROM node:14-alpine AS builder
ENV NODE_ENV build
WORKDIR /home/node
COPY . /home/node
RUN npm run build \
    && npm prune --production

FROM node:14-alpine
ENV NODE_ENV production
WORKDIR /home/node
COPY --from=builder /home/node/package*.json /home/node/
COPY --from=builder /home/node/node_modules/ /home/node/node_modules/
COPY --from=builder /home/node/build/ /home/node/build/

CMD ["node", "build/index.js"]
```

Node - Docker

- En este caso, definimos nuestra imagen de origen:
  - Creamos un entorno de compilado con Node en la versión 14.
  - Definimos la variable de entorno ***NODE_ENV***.
  - Definimos nuestro folder de trabajo bajo ***/home/node***.
  - Ejecutamos nuestro comando ***build*** y finalmente removemos las dependencias de desarrollador con el comando ***prune***.

Node - Docker

- En este caso, definimos nuestra imagen de origen:
  - Creamos otro entorno, el de ejecución, con Node en la versión 14.
  - Clonamos el Package.json, dependencias y código compilado de nuestro entorno de compilado.
  - Finalmente, ejecutamos el comando **`node build/index.js`**.
- Con esto finalmente hemos terminado de definir nuestra imagen de Docker.
- Por lo que, podemos crear una imagen y levantar un servicio con Docker.

Node - Docker

- Finalmente, ejecutamos el comando “**docker build .**”:

```
$ docker build .
[+] Building 22.4s (13/13) FINISHED
 => [internal] load build definition from Dockerfile                                                                0.1s
 => => transferring dockerfile: 556B                                                                                0.0s
 => [internal] load .dockerignore                                                                                   0.1s
 => => transferring context: 2B                                                                                     0.0s
 => [internal] load metadata for docker.io/library/node:14-alpine                                                   2.9s
 => [auth] library/node:pull token for registry-1.docker.io                                                         0.0s
 => [internal] load build context                                                                                   3.0s
 => => transferring context: 99.07MB                                                                                2.9s
 => [builder 1/4] FROM docker.io/library/node:14-alpine@sha256:7bcf853eeb97a25465cb385b015606c22e926f548cbd117f85b7196df8aa0d29   0.0s
 => CACHED [builder 2/4] WORKDIR /home/node                                                                         0.0s
 => [builder 3/4] COPY . /home/node                                                                                 4.0s
 => [builder 4/4] RUN npm run build     && npm prune --production                                                  11.9s
 => CACHED [stage-1 3/5] COPY --from=builder /home/node/package*.json /home/node/                                   0.0s
 => CACHED [stage-1 4/5] COPY --from=builder /home/node/node_modules/ /home/node/node_modules/                      0.0s
 => CACHED [stage-1 5/5] COPY --from=builder /home/node/build/ /home/node/build/                                    0.0s
 => exporting to image                                                                                              0.0s
 => => exporting layers                                                                                             0.0s
 => => writing image sha256:88cef320db12cb7c0bfbc64e03e831faa2cdc98dcb2713e20d30bf4835f5c93b                        0.0s
```

- Como podemos observar se construye nuestro contenedor según lo que definimos en el Dockerfile.

Node - Docker

- Podemos verificar nuestras imágenes de Docker creadas ejecutando el comando "**`docker images`**":

```
$ docker images
REPOSITORY                                TAG       IMAGE ID       CREATED          SIZE
express-unittesting-template_calc-api     latest    88cef320db12   2 hours ago      120MB
```

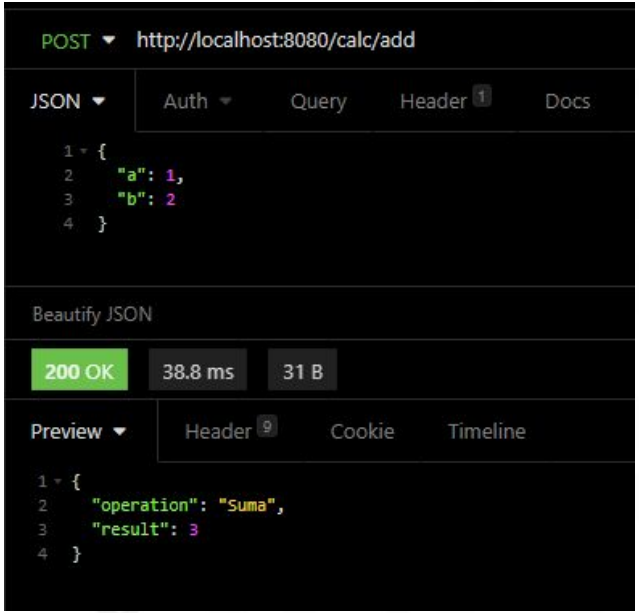
- Como podemos observar, encontramos la imagen que acabamos de crear.
- Finalmente si ejecutamos el comando "**`docker run -p 8080:8000 express-unittesting-template_calc-api`**"

```
$ docker run -p 8080:8000 express-unittesting-template_calc-api
Sevidor esperando por peticiones en localhost:8000
```

- Esto ejecutará nuestro contenedor enlazando nuestro puerto 8080 con el puerto 8000 del contenedor.

Node - Docker

- Por lo que si hacemos una solicitud a **`localhost:8080`**, como lo sería **`/calc/add`**:



- Este recurso se comunicará directamente con nuestro contenedor de Docker a través del puerto 8080, tal como se le específico con la configuración ***-p***.
- Este es un proceso manual para la ejecución de nuestros de contenedores, por ende como alternativa veremos el uso del comando **`docker-compose`**.

Node - Docker

- Para poder utilizar el comando ***docker-compose*** necesitamos configurar un archivo *"**docker-compose.yml**"*, para este trabajaremos con la versión 3 de **docker-compose**:

```
version: '3'

services:
  calc-api:
    build: .
    ports:
      - '8080:8000'
    environment:
      - PORT=8000
```

- Con esto definimos nuestro contenedor como un servicio dentro un listado.
- Configuramos como se compila y usamos "." para referirnos a nuestro Dockerfile.
- Adicionalmente, configuramos los puertos y variables de entorno.

Node - Docker

- Finalmente para realizar nuevamente el despliegue de nuestro contenedor o imagen de docker procedemos con el comando "**`docker-compose up`**":

```
$ docker-compose up
Starting express-unittesting-template_calc-api_1 ... done
Attaching to express-unittesting-template_calc-api_1
calc-api_1  | Sevidor esperando por peticiones en localhost:8000
```

- Como podemos ver, se despliega nuestro servicio y Docker se encarga de proveernos información al respecto.
- Por último, notamos que docker se encarga de levantar servicio a servicio.
- Y se encarga de informarnos del correcto funcionamiento de los mismos.

El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

Ejercicios de práctica

Mision TIC 2022

El futuro digital es de todos MinTIC

UNIVERSIDAD DEL NORTE
Vigilada Mineducación

Referencias

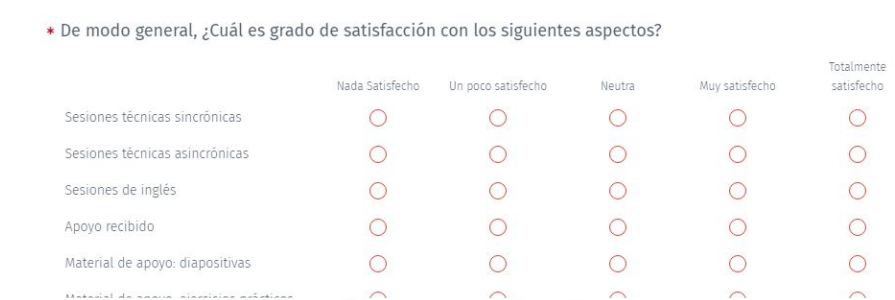
- https://medium.com/the-programmer/kubernetes-fundamentals-for-absolute-beginners-architecture-components-1f7cda8ea536
- https://docs.docker.com/get-docker/
- https://medium.com/all-you-need-is-clean-code/microservicios-50a585b1bbff
- https://www.loginradius.com/blog/async/production-grade-development-using-docker-compose/
- https://www.cloudbees.com/blog/using-docker-compose-for-nodejs-development

Mision TIC 2022

El futuro digital es de todos | MinTIC

UNIVERSIDAD DEL NORTE
Vigilada Mineducación

Seguimiento Habilidades Digitales en Programación



* De modo general, ¿Cuál es grado de satisfacción con los siguientes aspectos?

	Nada Satisfecho	Un poco satisfecho	Neutra	Muy satisfecho	Totalmente satisfecho
Sesiones técnicas sincrónicas	○	○	○	○	○
Sesiones técnicas asincrónicas	○	○	○	○	○
Sesiones de inglés	○	○	○	○	○
Apoyo recibido	○	○	○	○	○
Material de apoyo: diapositivas	○	○	○	○	○

Completa la siguiente encuesta para darnos retroalimentación sobre esta semana ▼▼▼

https://www.questionpro.com/t/ALw8TZlxOJ

MisionTIC2022

El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

¡GRACIAS

POR SER PARTE DE ESTA EXPERIENCIA DE APRENDIZAJE!

Mision TIC 2022