



**Universidad
del Valle**

Facultad de Ingeniería
Escuela de Ingeniería

Universidad del Valle – Sede Tuluá

INFORME ACADÉMICO

Proyecto Final

AUTOR

Kevin Steven Ramirez Torres 2259371

Juan Manuel Ramirez Agudelo 2259482

Alejandro Sierra Betancourt 2259559

Juan Pablo Castaño 2259487

DOCENTE

Carlos Andres Delgado Saavedra



Tuluá – Colombia

2024–2

Índice

- **Introducción:**
 - Breve descripción del proyecto.
 - Objetivos del proyecto

- **Solución Local:**
 - Descripción detallada de la arquitectura local usando Docker Compose o Kubernetes.
 - Diseño y separación de componentes: backend, base de datos y frontend.
 - Configuración de cada contenedor y cómo estos interactúan.

- **Solución en la Nube:**
 - Replicación de la arquitectura local en la nube eg. Azure.
 - Configuración del balanceador de carga para el backend replicado tres veces.
 - Uso de servicios en la nube, configuración de redes y seguridad.
 - Descripción del flujo para subir la aplicación a la nube

- **Análisis y Conclusiones:**
 - Analizar el rendimiento de la aplicación en ambiente local vs en la nube.
 - Analizar la latencia, escalabilidad y disponibilidad del sistema.
 - Interpretación de posibles retos y cómo fueron abordados.
 - Reflexiones sobre el uso de tecnologías como Docker y Kubernetes.

- **Conclusiones**



Introducción

Descripción del Proyecto:

Lava-Wash es una plataforma web diseñada para ofrecer una experiencia cómoda y eficiente en la programación de servicios de lavado de autos. El proyecto busca resolver problemas comunes en el sector, como largas esperas, procesos manuales y plataformas saturadas de información no relevante.

A través de Lava-Wash, los usuarios pueden:

- Crear una cuenta segura con contraseñas protegidas mediante hashing.
- Agendar citas para el lavado de sus vehículos en horarios disponibles según su conveniencia.
- Recuperar el acceso a su cuenta mediante un enlace de restablecimiento enviado por correo electrónico.

El proyecto no solo beneficia a los clientes finales, sino que también mejora la eficiencia operativa de los lavaderos al automatizar la gestión de reservas, eliminando la necesidad de procesos manuales y aumentando la satisfacción del cliente.

Objetivos del proyecto:

Objetivo General

Desarrollar una plataforma en línea que permita a los usuarios agendar servicios de lavado de autos de manera rápida, eficiente y segura, optimizando tanto el tiempo de los clientes como la gestión de los lavaderos.

Objetivos Específicos



- Diseñar una interfaz amigable y accesible que facilite la navegación y la programación de citas para usuarios con diferentes niveles de habilidades tecnológicas.
- Implementar un sistema de autenticación de usuarios seguro, incluyendo hash de contraseñas y recuperación mediante correo electrónico, para proteger la información personal.
- Incorporar un mecanismo de gestión de horarios que permita a los clientes seleccionar fácilmente la disponibilidad de servicios y evitar tiempos de espera innecesarios.
- Optimizar la experiencia de las pequeñas y medianas empresas mediante la funcionalidad de programación de lavados para flotas de vehículos, reduciendo tiempos de gestión.
- Garantizar la protección de los datos sensibles de los usuarios mediante el uso de protocolos de seguridad como HTTPS y encriptación de datos.




Solución Local:

Descripción detallada de la arquitectura local usando Docker Compose o Kubernetes, Configuración de cada contenedor y cómo estos interactúan.

Descripción de dockerfile de backend

```
backend > Dockerfile > ...
1  # Usa una imagen oficial de Node.js como base
2  FROM node:18
3
4  # Establecer la variable de entorno para la producción
5  ENV NODE_ENV=production
6
7  # Crear y establecer el directorio de trabajo
8  WORKDIR /usr/src/app
9
10 # Copiar el archivo package.json y package-lock.json (si existe) e instalar dependencias
11 COPY package*.json ./
12 RUN npm install
13
14
15 # Instalar netcat-openbsd (nc) para usarlo en el script wait-for-it.sh
16 RUN apt-get update && apt-get install -y netcat-openbsd
17
18 # Copiar el resto del código fuente al contenedor
19 COPY . .
20
21 # Copiar el script wait-for-it.sh
22 COPY wait-for-it.sh /usr/local/bin/wait-for-it.sh
23 RUN chmod +x /usr/local/bin/wait-for-it.sh
24
25 # Exponer el puerto en el que la aplicación se ejecuta
26 EXPOSE 3000
27
28 # Comando para iniciar la aplicación con espera
29 CMD /usr/local/bin/wait-for-it.sh db-mono -- npm start
```

para implementar la base de datos agregamos un archivo wait-for-it.sh

```
backend >  wait-for-it.sh
9
10
11  until nc -z "$host" 3306; do
12      echo "Esperando a que MySQL esté listo en $host:3306..."
13      sleep 2
14  done
15  #instalar el cliente mysql
16  echo "Instalando cliente mysql..."
17  apt-get update
18  apt-get install -y default-mysql-client
19
20
21  # Insertar lavadero.sql a la base de datos
22  echo "Insertando lavadero.sql a la base de datos..."
23  mysql -h db-mono -u root -proot lavadero < /usr/src/app/lavadero.sql
24
25  # Validar si se insertó correctamente
26  mysql -h db-mono -u root -proot lavadero -e "SHOW TABLES;" | cat
27
28
29
30  # Iniciar la aplicación
31  echo "Iniciando la aplicación..."
32  exec $cmd
```

Descripción de dockerfile de frontend

```
1  # Etapa 1: Construcción
2  FROM node:18 AS builder
3
4  # Establecer el directorio de trabajo
5  WORKDIR /app
6
7  # Copiar los archivos de dependencias e instalarlas
8  COPY package*.json ./
9  RUN npm install
10
11 # Copiar el código fuente y compilar la aplicación Angular
12 COPY . .
13 RUN npm run build --prod
14
15 # Etapa 2: Servir la aplicación
16 FROM nginx:alpine
17
18 # Copiar los archivos compilados desde la etapa de construcción
19 COPY --from=builder /app/dist/development/browser /usr/share/nginx/html
20
21 # Copiar el archivo default.conf a la ubicación correcta de Nginx
22 COPY default.conf /etc/nginx/conf.d/default.conf
23
24 # Exponer el puerto 4200
25 EXPOSE 4200
26
27 # Iniciar Nginx
28 CMD ["nginx", "-g", "daemon off;"]
29
```

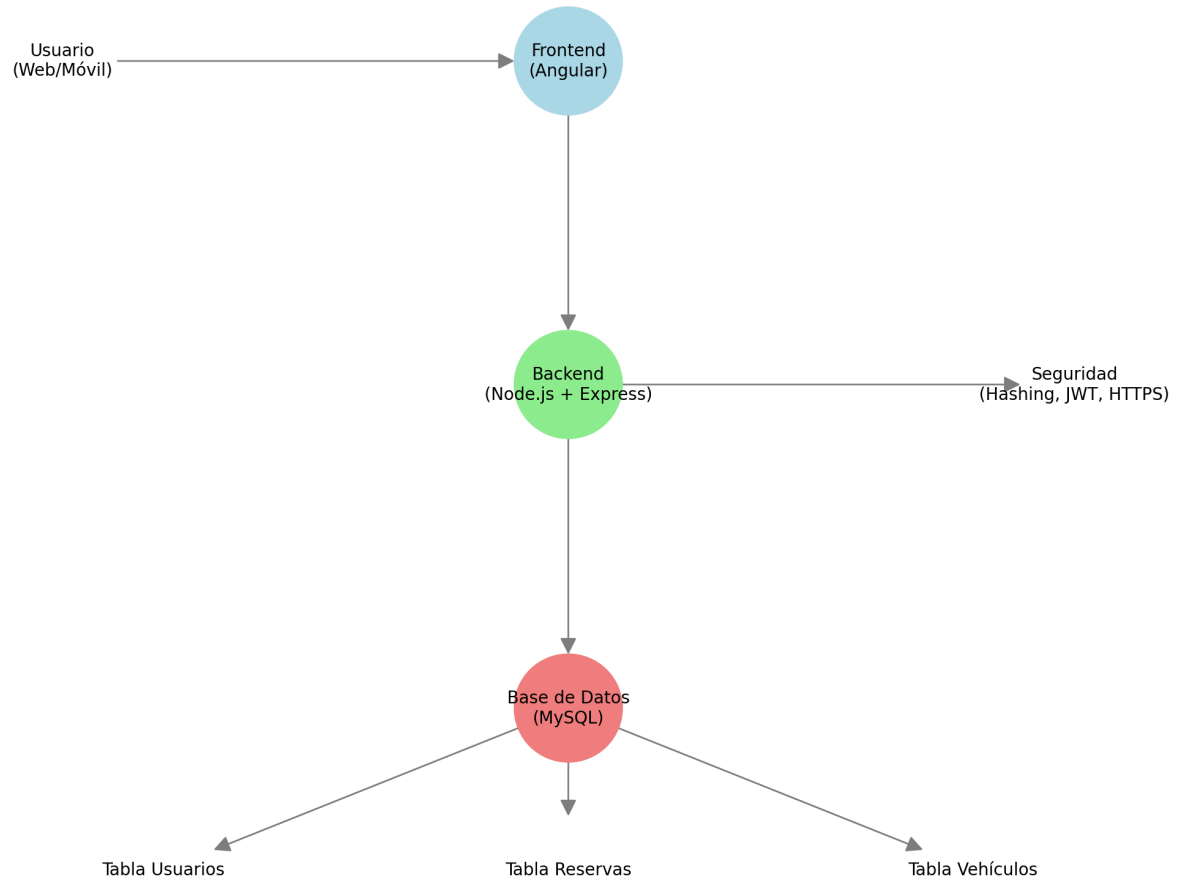
Docker compose

```
docker-compose.yml
1  services:
2    frontendmono:
3      image: angular-app
4      container_name: frontend-mono
5      ports:
6        - "4200:4200"
7      depends_on:
8        - backendmono
9      networks:
10       - my-network
11
12   backendmono:
13     image: backend-mono
14     container_name: backend-mono
15     ports:
16       - "3000:3000"
17     environment:
18       - NODE_ENV=development
19       - PORT=3000
20     volumes:
21       - ./backend:/app/backend
22     networks:
23       - my-network
24     depends_on:
25       - db-mono
26
27   db-mono:
28     image: mysql:9.1.0
29     environment:
30       MYSQL_ROOT_PASSWORD: root
31       MYSQL_DATABASE: lavadero
32     ports:
33       - "3306:3306"
34     volumes:
35       - db_data:/var/lib/mysql
36     networks:
37       - my-network
38
39
40   networks:
41     my-network:
42       driver: bridge
43
44   volumes:
45     db_data: # Declaración del volumen
46
```




Diseño y separación de componentes: backend, base de datos y frontend

Diseño y Separación de Componentes de Lava-Wash



Solución en la Nube:

Replicación de la arquitectura local en la nube la hicimos en digital ocean

```
name: backendmono
spec:
  replicas: 3
  selector:
    matchLabels:
      io.kompose.service: backendmono
  strategy:
    type: Recreate
  template:
    metadata:
      annotations:
        kompose.cmd: kompose convert -f docker-compose.yml
        kompose.version: 1.34.0 (HEAD)
      labels:
        io.kompose.service: backendmono
    spec:
      containers:
      - env:
        - name: NODE_ENV
          value: development
        - name: PORT
          value: "3000"
        image: svenram/proyectoinfra:backendmono-1.0.6
        name: backend-mono
        ports:
```

como se puede observar en la imagen para el backend implementamos 3 replicas y en el archivo backend-mono-deployment.yaml se cargo esta configuración

```
svenram@debian:~/Documentos/ProyectoInfra/infra$ kubectl get services
NAME                TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
backendmono         LoadBalancer  10.245.223.80    24.199.65.57     3000:30160/TCP    23m
db-mono             ClusterIP      10.245.197.37    <none>           3306/TCP          30m
frontendmono        LoadBalancer  10.245.54.78     178.128.134.211  4200:32437/TCP    2m33s
kubernetes           ClusterIP      10.245.0.1       <none>           443/TCP           36m
svenram@debian:~/Documentos/ProyectoInfra/infra$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
backendmono-74b5644447-2g7tw        1/1     Running   0           17m
backendmono-74b5644447-5dj25        1/1     Running   0           17m
backendmono-74b5644447-gv9bl        1/1     Running   0           17m
db-mono-865cdc9f4d-75kbx            1/1     Running   0           32m
frontendmono-56789f848-7dkq9        1/1     Running   0           4m43s
svenram@debian:~/Documentos/ProyectoInfra/infra$
```

estamos trabajando desde el entorno implementado en la herramienta de digitalocean

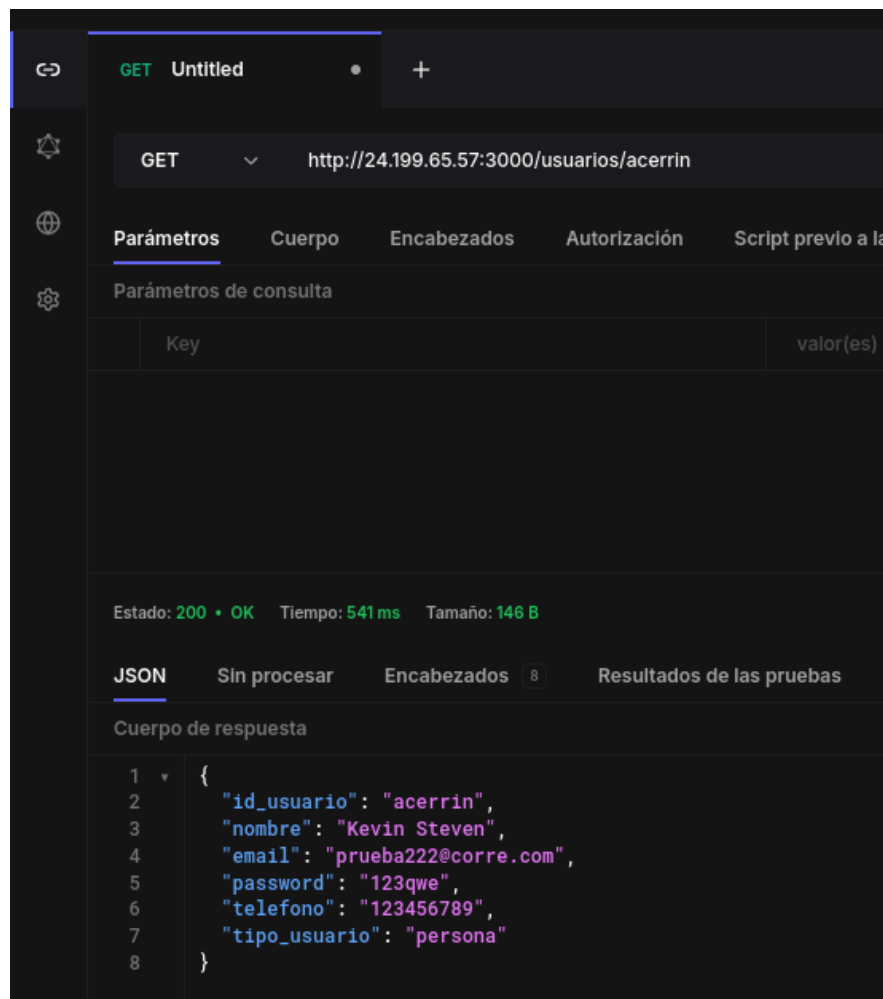


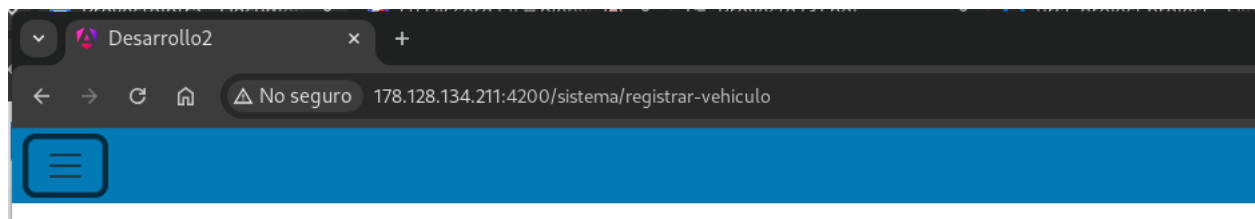
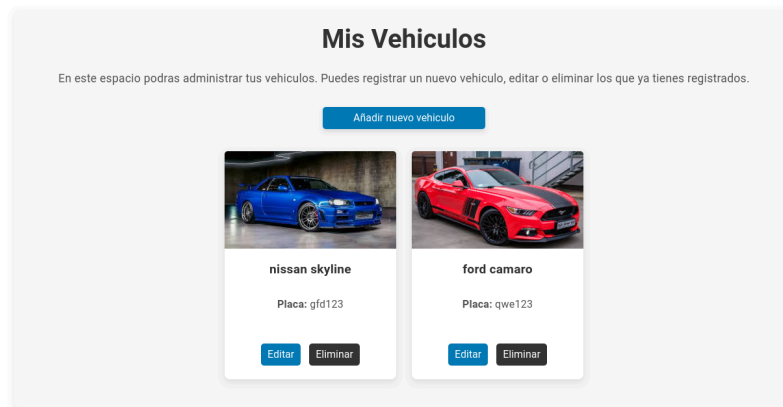
Descripción del flujo para subir la aplicación a la nube.

inicialmente creamos los dockerfiles de nuestro front y back para posteriormente usar el compose, seguidamente montamos las imagenes en docker hub. Una vez con todo esto montado creamos el cluster en digitalocean para montar estas imagenes y luego usamos los deployments, configmaps y services para darle las configuraciones a nuestra app en la nube

Pruebas de uso

prueba de servicio generado para el backend





uso de la ip generada para el frontend



Rendimiento de la aplicación: ambiente local vs nube

Ambiente local:

Ventajas: Control total de recursos, bajos costos iniciales, facilidad para pruebas rápidas y experimentación.

Desventajas: Limitaciones en capacidad de hardware, difícil de escalar, dependiente de la conexión local.

Métricas clave: Tiempo de respuesta de API, uso de CPU/RAM, acceso a base de datos.

En la nube:

Ventajas: Escalabilidad inmediata, alta disponibilidad, infraestructura gestionada.

Desventajas: Dependencia de costos recurrentes, complejidad en la configuración inicial, latencia de red.

Métricas clave: Tiempo de respuesta (incluyendo la latencia de red), rendimiento bajo carga, costos de operación.

Latencia, escalabilidad y disponibilidad del sistema

Latencia:

En ambiente local, la latencia es baja debido a la proximidad del servidor.

En la nube, la latencia subió un poco debido al proveedor y la región seleccionada que no fue Colombia sino USA new yorkk.

La solución que se implementó fue utilizar balanceador de carga para reducir la distancia entre el cliente y los recursos.



Escalabilidad:

Ambiente local: Estamos limitados por el hardware físico ya requiere actualizaciones manuales.

Nube: Uso de escalado horizontal (más instancias) o vertical (más recursos por instancia).

Solución: Configura autoescalado en la nube con métricas como uso de CPU o cantidad de peticiones por segundo.

Disponibilidad:

Local: Dependiente de la estabilidad del servidor físico.

Nube: Garantía de disponibilidad del proveedor

Solución: Configura instancias en varias regiones para mayor redundancia.

Retos e interpretación de cómo abordarlos

Configuración inicial compleja:

ansible es una buena opción para hacer un poco más sencilla todos estos cambios y modificaciones que le hacemos a las imágenes que subimos a docker hub

Reflexiones sobre el uso de Docker y Kubernetes

Docker:

Ventajas:

Entornos consistentes entre desarrollo, pruebas y producción.

Facilidad para escalar en un servidor local o en la nube.

Implementación: Conteneriza tu app Angular, el backend Express y la base de datos SQL.

Configura un archivo docker-compose.yml para orquestar los contenedores.



Kubernetes:

Ventajas:

Ideal para manejar múltiples contenedores en producción.

Escalabilidad y recuperación automática en caso de fallos.

Retos:

Mayor curva de aprendizaje y complejidad inicial.

Puede ser excesivo para proyectos pequeños si no esperas un alto tráfico inicial.

Conclusión:

Para un proyecto como Lava-Wash, inicia con Docker para contenerizar tu app y evalúa Kubernetes si esperas un crecimiento significativo o múltiples microservicios en el futuro.

Conclusiones

El desarrollo y las pruebas iniciales son más ágiles y económicas en un ambiente local. Sin embargo, la nube proporciona ventajas significativas en escalabilidad y disponibilidad, lo que la hace ideal para desplegar la aplicación en producción. La elección depende del nivel de tráfico esperado y los recursos disponibles.

La nube ofrece escalabilidad automática y disponibilidad garantizada (con configuraciones adecuadas), algo que es complejo de implementar en un ambiente local. Configurar autoescalado y redundancia en la nube es esencial para mantener el rendimiento en picos de tráfico.

Docker es una herramienta esencial para garantizar consistencia en los entornos de desarrollo, pruebas y producción. Kubernetes, aunque poderoso, puede ser excesivo para un proyecto pequeño o en etapa inicial. Es más adecuado si se espera un crecimiento significativo o la necesidad de manejar múltiples microservicios en el futuro.

