Memoria del Proyecto

Equipo 3:

ALEJANDRO PEREZ-CHIRINOS RODRIGUEZ

ADRIAN DOMINGUEZ MOTA

DANIEL EDUARDO CALEIRAS MARTIN

MIGUEL ESCRIBANO BUZON

SOFIA COVADONGA BROTTON RUIZ

Indice

1. Introducción

Esta memoria descriptiva documenta la infraestructura y el despliegue de una aplicación de gestión de talleres de mecánica rápida basada en una arquitectura orientada a servicios. El objetivo principal es consolidar los conceptos relacionados con la infraestructura y el despliegue de la aplicación.

En la primera práctica, se especificó la API para uno de los servicios de la aplicación (recambios). En esta, se estudiará y planificará la infraestructura necesaria para integrar todos los servicios, considerando dos enfoques de despliegue: Docker Compose y Kubernetes.

2. Infraestructura y despliegue

En esta sección, se detalla la infraestructura necesaria para la integración de los subsistemas de la aplicación, así como los enfoques de despliegue utilizando Docker Compose y Kubernetes.

2.1 Despliegue con Docker Compose

El despliegue de los servicios se realiza utilizando Docker Compose, que permite definir y gestionar múltiples contenedores. Cada servicio se despliega en su propio contenedor Docker y se configuran las comunicaciones entre ellos utilizando networks específicos.

2.1.1 Esquema de despliegue en Docker Compose

En el esquema de despliegue presentado, los servicios se comunican entre sí a través de las redes internas definidas por Docker Compose. No es necesario exponer los puertos internos en el archivo de configuración. La comunicación se realiza mediante las rutas y redirecciones configuradas en el servicio Caddy. Esto permite un acoplamiento suelto y mayor seguridad al evitar la exposición directa de puertos externos.

- Caddy
  - Network: proxy_network
- Clientes Service
  - Network: clientes_network
  - Puertos: 4010
- Vehículos Service
  - Network: vehiculos_network
  - Puertos: 4010
- Trabajos Service
  - Network: trabajos_network
  - Puertos: 4010
- Notificaciones Service
  - Network: notificaciones_network
  - Puertos: 4010

- Facturas Service
    - Network: facturas_network
    - Puertos: 4010

- Recambios Service
    - Network: recambios_network
    - Puertos: 8000
- Logs Service
    - Network: logs_network
    - Puertos: 4010

2.1.2 Comunicación entre los Servicios

La comunicación entre los servicios se establece a través de redes internas definidas por Docker Compose. Cada servicio se conecta a su red específica para asegurar la comunicación adecuada. A continuación se detalla la comunicación entre los servicios:

- El servicio Caddy actúa como proxy y redirige las solicitudes entrantes a los servicios correspondientes utilizando las siguientes rutas:
    - http://127.0.0.1/clientes: Redirige a la especificación Swagger del servicio de Clientes.
    - http://127.0.0.1/vehiculos: Redirige a la especificación Swagger del servicio de Vehículos.
    - http://127.0.0.1/trabajos: Redirige a la especificación Swagger del servicio de Trabajos.
    - ...
    - http://127.0.0.1/recambios: Redirige a la documentación del servicio de Recambios.
- Las peticiones a los servicios se realizan utilizando la URL base http://127.0.0.1/api/v1, seguida del endpoint específico de cada servicio. Por ejemplo, una petición al servicio de Clientes se realizaría como http://127.0.0.1/api/v1/clientes.

2.1.3 Justificación de las Decisiones Tomadas

Las decisiones tomadas para este esquema de despliegue se basan en los siguientes factores:

- Modularidad: Cada servicio se despliega en su propio contenedor, lo que garantiza la independencia y el aislamiento de los servicios. Esto permite escalar y mantener los servicios de manera individual, sin afectar a los demás componentes del sistema.
- Comunicación entre Servicios: Se utilizan redes internas específicas para cada servicio, lo que facilita la comunicación entre los servicios y evita la exposición innecesaria de puertos externos. Esto permite un acoplamiento suelto y una mayor flexibilidad para agregar, modificar o eliminar servicios sin afectar a otros componentes del sistema.
- Proxy Reverso: El uso de Caddy como proxy reverso proporciona una capa adicional de seguridad y gestión de solicitudes, permitiendo redirigir las solicitudes entrantes a los servicios correspondientes según la ruta especificada en el Caddyfile.

- En relación al enfoque de despliegue con Docker Compose, es importante mencionar que se implementó la API de recambios utilizando Python con FastAPI y MongoDB. A diferencia de los demás servicios mencionados en el despliegue, los cuales son simulaciones (mocks) del backend de cada servicio, el servicio de recambios se encuentra completamente funcional, con lógica de negocio y persistencia de datos.

En resumen, el esquema de despliegue y las decisiones tomadas garantizan un entorno eficiente y seguro para la ejecución de los servicios, permitiendo la comunicación adecuada entre ellos y asegurando su disponibilidad y escalabilidad. Además, se evita la exposición de puertos externos innecesarios al utilizar redes internas específicas para cada servicio.

2.2 Despliegue con Kubernetes

2.2.1 Recursos utilizados

En Kubernetes, se utilizarán los siguientes recursos para el despliegue de los servicios:

- Services (Servicios): Se crea un servicio para cada servicio de la práctica. Los servicios actúan como puntos de entrada para acceder a los pods que ejecutan los contenedores. Cada servicio se encarga de enrutar las solicitudes entrantes al pod correspondiente.
- Deployments (Despliegues): Se crea un deployment para cada contenedor del Docker Compose. Un deployment define un conjunto de pods replicados que ejecutan los contenedores. Cada deployment se encarga de gestionar la replicación y escalado de los pods.
- Volumes (Volúmenes): Se han utilizado volúmenes para garantizar la persistencia de los datos. Aunque en este caso no hay datos persistentes, los volúmenes permiten almacenar los datos en el host de la máquina, evitando la pérdida de datos en caso de reinicio o reinicio del pod.

2.2.2 Esquema de Despliegue en Kubernetes

En este esquema, cada servicio se despliega mediante un Deployment, que a su vez crea y gestiona uno o varios Pods. Cada Pod ejecuta los contenedores necesarios para el servicio correspondiente.

La comunicación entre los servicios se establece a través de la red interna de Kubernetes y se realiza mediante resolución de nombres de servicio. Cada servicio tiene su propio punto de entrada y los pods se escalan y replican según sea necesario para garantizar la disponibilidad y escalabilidad de los servicios.

Es importante destacar que este esquema se basa en la estructura utilizada en el Docker Compose previo, asegurando la misma organización y separación de servicios. Además, se considera la posibilidad de utilizar volúmenes para la persistencia de datos en el futuro. Este esquema de despliegue en Kubernetes proporciona un entorno modular, escalable y robusto

para los servicios de la práctica, permitiendo una gestión eficiente de los mismos y garantizando su disponibilidad y rendimiento.

2.2.3 Justificación de las Decisiones Tomadas

Las decisiones tomadas para este esquema de despliegue en Kubernetes se basan en los siguientes factores:

- Separación de Servicios: Se crea un servicio para cada servicio de la práctica, lo que permite una mayor modularidad y aislamiento de los componentes.
- Escalabilidad y Replicación: Mediante el uso de deployments, se pueden definir múltiples réplicas de los pods que ejecutan los contenedores. Esto permite escalar horizontalmente los servicios según la demanda y garantiza la disponibilidad en caso de fallos.
- Persistencia de Datos: Aunque en este caso no hay datos persistentes, se menciona la posibilidad de utilizar volúmenes para almacenar datos en el host de la máquina. Esto proporciona una capa adicional de seguridad y evita la pérdida de datos en caso de reinicios o reinicios de los pods.
- El despliegue en Kubernetes sigue la misma estructura y organización que el Docker Compose utilizado previamente. Se define un deployment para cada servicio y se crea un servicio correspondiente para cada deployment. Esto permite una comunicación efectiva entre los servicios mediante la resolución de nombres de servicio y la conexión a través de la red interna de Kubernetes.

En resumen, el esquema de despliegue en Kubernetes utilizando servicios, deployments y volúmenes proporciona un entorno robusto y escalable para los servicios de la práctica. Cada servicio tiene su propio punto de entrada y se puede escalar de manera independiente según sea necesario. Además, se considera la persistencia de datos utilizando volúmenes en caso de requerirse en el futuro.

3. Conclusiones

En este informe se ha detallado la infraestructura y el despliegue de una aplicación de gestión de talleres de mecánica rápida basada en una arquitectura orientada a servicios. Se han considerado dos enfoques de despliegue: Docker Compose y Kubernetes.

A pesar de los avances logrados, es importante mencionar que el despliegue en Kubernetes se encuentra en un estado incompleto en comparación con el despliegue en Docker Compose. Aunque se han definido algunos recursos, como servicios y deployments, no se ha logrado alcanzar el mismo nivel de funcionalidad y configuración. Se han encontrado desafíos técnicos y consideraciones adicionales que aún deben abordarse para lograr un despliegue completo y equivalente al del Docker Compose.