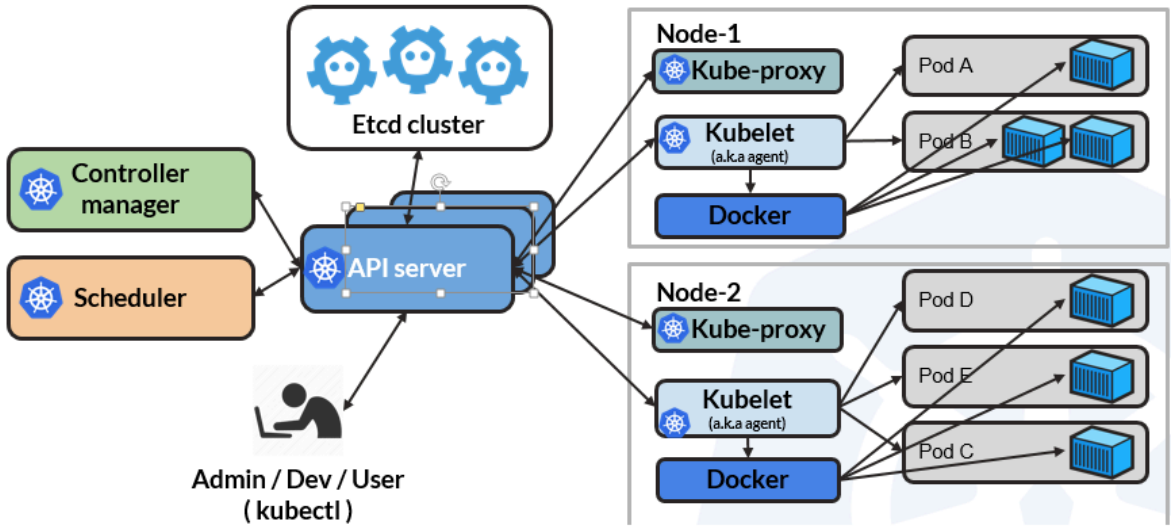
Despliegue con Minikube

Es una tecnología de gestión de contenedores desarrollada por GOOGLE (posteriormente hecha de código abierto en 2015) para gestionar aplicaciones contenedorizadas (orquestación).

¿Para qué?

1. Descubrimiento de servicios y balanceo de carga
2. Reversiones automáticas
3. Autocuración
4. Autoescalado
5. Actualizaciones Canary y actualizaciones rodantes
6. Código abierto y impulsado por la comunidad
7. Alta disponibilidad

Diagrama de Arquitectura de Componentes



Componentes Básicos

Componentes en los Nodos del Gestor:

- **Controller manager**: Ejecuta varios controladores para ayudar a mover el estado en ejecución al estado deseado.
- **Node Controller**: Responsable de notar y responder cuando los nodos se caen.
- **Replication Controller**: Responsable de mantener el número correcto de pods para cada objeto de replicación en el sistema.
- **Endpoints Controller**: Llena el objeto Endpoints (es decir, une Servicios y Pods).
- **Service Account & Token Controllers**: Crea cuentas predeterminadas y tokens de acceso a la API para nuevos espacios de nombres.

- **Scheduler**: Observa los pods recién creados que no tienen un nodo asignado y selecciona un nodo para que se ejecuten en él.
- **API Server**: El front-end del plano de control de Kubernetes. Está diseñado para escalar horizontalmente. Todos los demás componentes se comunican con este.
- **Etcd Cluster**: Almacén de respaldo de clave/valor para los datos del clúster. Almacena el estado del clúster (qué nodos existen en el clúster, qué pods deberían estar ejecutándose, en qué nodos se están ejecutando y mucho más) en cualquier momento dado.

Componentes en los Nodos de Trabajo:

- **Kubelet**: Agente que observa continuamente el servidor de API. Asegura que los contenedores estén ejecutándose en un pod.
- **Kube-proxy**: Un servicio proxy que se ejecuta en cada nodo de trabajo para manejar el subneteo individual del host y exponer servicios al mundo exterior. Realiza el reenvío de solicitudes a los pods/contenedores correctos a través de las diversas redes aisladas en un clúster.

Conceptos de Kubernetes

Concepto	Descripción
Node	Máquina en el clúster
Docker	Ayuda en la creación de contenedores que incluyen aplicaciones y sus binarios.
Pods	Un Pod es el bloque de construcción básico de Kubernetes: la unidad más pequeña y simple en el modelo de objetos de Kubernetes que se crea o despliega, y también es un grupo de contenedores (1 o más). Solo los contenedores del mismo pod pueden compartir almacenamiento compartido.
Service	Es una abstracción que define un conjunto lógico de Pods y una política para acceder a ellos.
Jobs	Crea pod(s) y asegura que un número especificado se complete exitosamente. Cuando se completa el número especificado de ejecuciones exitosas de pods, el trabajo se considera completo.
Cronjob	Programador de trabajos en Kubernetes (K8s).
Replicasets	Asegura cuántas réplicas de un pod deben estar ejecutándose.
Namespaces	Separación lógica entre equipos y sus entornos. Permite a varios equipos (Desarrollo, Producción) compartir el clúster de Kubernetes proporcionando un espacio de trabajo aislado.
Deployment	Estado deseado de pods para actualizaciones declarativas.

DaemonSet	Asegura que un pod en particular se ejecute en algunos o todos los nodos.
PersistentVolume	Almacenamiento persistente en el clúster con un ciclo de vida independiente.
PersistentVolumeClaim	Solicitud de almacenamiento (para un PersistentVolume) por un usuario.
Ingress	Un Ingress es una colección de reglas que permiten que las conexiones entrantes lleguen a los servicios del clúster.

Pasos:

Paso 1: Configurar el Proyecto Spring Boot

Usa Spring Initializr para generar un proyecto con las siguientes dependencias:

- Spring Web
- Spring Data JPA
- Spring Security
- H2 Database (para desarrollo), MySQL, PostgreSQL, NoSQL
- Lombok

Paso 2. Configurar la estructura del proyecto

Crea los paquetes

- 🔄 **`controller`**,
- 🛠️ **`service`**,
- 📦 **`repository`**,
- 🧩 **`model`**,
- 📤 **`dto`**.

Paso 3: Crear Dockerfiles

Contenido Típico

- **`FROM`**: Especifica la imagen base a partir de la cual se construirá la nueva imagen.
- **`COPY`** o **`ADD`**: Copia archivos y directorios desde el host al sistema de archivos de la imagen.
- **`RUN`**: Ejecuta comandos en la imagen, como instalar paquetes.
- **`CMD`** o **`ENTRYPOINT`**: Especifica el comando que se ejecutará cuando se inicie un contenedor a partir de la imagen.
- **`EXPOSE`**: Indica el puerto en el que el contenedor escuchará en tiempo de ejecución.
- **`ENV`**: Establece variables de entorno.

Dockerfile para Spring Boot

```
# Usa una imagen base de Maven para construir el proyecto
FROM maven:3.8.4-openjdk-17 AS build
WORKDIR /app

# Copia el archivo pom.xml y descarga las dependencias
COPY src-api/pom.xml .
RUN mvn dependency:go-offline

# Copia el resto del código del proyecto y compila la aplicación
COPY src-api/src ./src
RUN mvn package -DskipTests

# Usa una imagen base de OpenJDK para ejecutar la aplicación
FROM openjdk:17-jdk-slim
WORKDIR /app

# Copia el JAR compilado desde la etapa anterior
COPY --from=build /app/target/API-VG-REVIEWS-0.0.1-SNAPSHOT.jar
app.jar

# Expone el puerto 8080
EXPOSE 8080

# Define la entrada para ejecutar la aplicación
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Dockerfile

Dockerfile para React

```
FROM node:14 as build
WORKDIR /app COPY . .
RUN npm install
RUN npm run build FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
XPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

K8S

Paso 4: Crear el Entorno Kubernetes en Minikube

Instalar Minikube y Kubectl

Sigue las instrucciones en **kubectl** y Minikube para instalarlos.

Iniciar Minikube

```
minikube start --driver=docker
minikube config set driver docker
```

Construir imágenes Docker dentro de Minikube

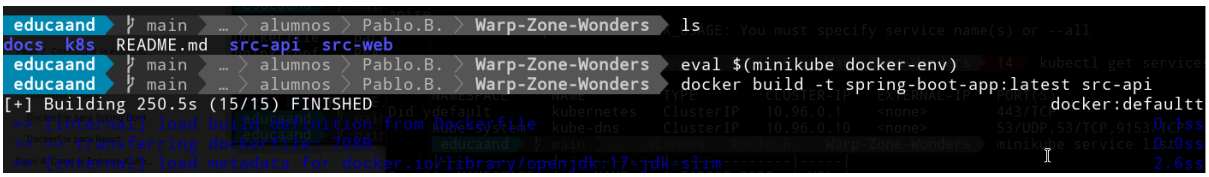
```
eval $(minikube docker-env)
docker build -t spring-boot-app:latest path/to/spring-boot-project
docker build -t react-app:latest path/to/react-project
```

Paso 5: Crear los Archivos de Configuración de Kubernetes

```
my-spring-boot-frontend-project/
│
├── backend/
│   ├── src/
│   ├── pom.xml
│   ├── Dockerfile
│   └── ... (otros archivos del backend)
│
├── frontend/
│   ├── src/
│   ├── package.json
│   ├── Dockerfile
│   └── ... (otros archivos del frontend)
│
├── k8s/
│   ├── mysql-deployment.yaml
│   ├── spring-boot-deployment.yaml
│   ├── react-deployment.yaml
│   └── ... (otros archivos de configuración de Kubernetes)
│
└── README.md
```

K8S

Crear Deployment y Service para Spring Boot



docker build -t your-docker-repo/spring-boot-app:latest -f src-api/**Dockerfile .**

Crear un archivo de configuración para MySQL

mysql-deployment.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:5.7
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "root_password"
        - name: MYSQL_DATABASE
          value: "prueba"
        ports:
        - containerPort: 3306
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
```

```
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  ports:
  - port: 3306
    targetPort: 3306
  selector:
    app: mysql
```

Crea un archivo `spring-boot-app` `spring-boot-deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spring-boot-app
  labels:
    app: spring-boot-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: spring-boot-app
  template:
    metadata:
      labels:
        app: spring-boot-app
    spec:
      containers:
      - name: spring-boot-container
        image: spring-boot-app:local
        ports:
        - containerPort: 8080
        env:
        - name: SERVER_PORT
          value: "8080"
        - name: SPRING_PROFILES_ACTIVE
          value: "demo"
        - name: SERVER_ERROR_WHITELABEL_ENABLED
          value: "false"
        - name: JWT_SECRET
          value: "3Vs6wFZNv7T75lC6P/KHhBulb5HIG6e8KUdLskt+ssw="
        - name: SPRING_DATASOURCE_URL
          value: "jdbc:mysql://mysql-service:3306/prueba"
        - name: SPRING_DATASOURCE_USERNAME
```

```
          value: "root"
        - name: SPRING_DATASOURCE_PASSWORD
          value: "root_password"
        - name: SPRING_JPA_HIBERNATE_DDL_AUTO
          value: "update"
        - name: SPRING_JPA_SHOW_SQL
          value: "true"
        - name: SPRING_JPA_PROPERTIES_HIBERNATE_DIALECT
          value: "org.hibernate.dialect.MySQL8Dialect"
---
apiVersion: v1
kind: Service
metadata:
  name: spring-boot-service
spec:
  type: NodePort
  selector:
    app: spring-boot-app
  ports:
    - port: 8080
      targetPort: 8080
      nodePort: 30000
```

Deployment para Angular

/your-angular-project
  /dist
    /your-angular-app
      index.html
      main.js
      ...
  /nginx
    nginx.conf
  Dockerfile

Dokerfie

```
# Build stage
FROM node:14 AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build --prod

# Production stage
FROM nginx:alpine
COPY --from=build /app/dist/your-angular-app /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
```

Deployment y Service para React

Crea un archivo `react-deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: react-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: react-app
  template:
    metadata:
      labels:
        app: react-app
    spec:
      containers:
      - name: react-app
        image: react-app:latest
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: react-service
spec:
  type: NodePort
  selector:
    app: react-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
    nodePort: 30002
```

Paso 6: Desplegar en Minikube

- **Construir la imagen Docker**:
  - docker **build** -t your-docker-repo/your-angular-app:latest .
- **Subir la imagen Docker a un repositorio**:
  - docker **push** your-docker-repo/your-angular-app:latest
- **Desplegar en Kubernetes**:
  - kubectl **apply** -f angular-deployment.yaml
- **Verificar el estado del despliegue**:
  - kubectl **get** deployments

- kubectl get pods
- kubectl get services

1. Aplicar las configuraciones de Kubernetes

```
eval $(minikube docker-env)
kubectl apply -f mysql-deployment.yaml
kubectl apply -f spring-boot-deployment.yaml
kubectl apply -f react-deployment.yaml
```

2. Verificar el despliegue

```
kubectl get pods
kubectl get services
```

3. Verificar que la aplicación Spring Boot puede conectarse a MySQL

```
kubectl logs <spring-boot-pod-name>.
```

4. Acceder a la aplicación

Obtener la URL de Minikube para acceder a los servicios:

```
minikube service list

minikube service spring-boot-service
minikube service react-service
```

K8S

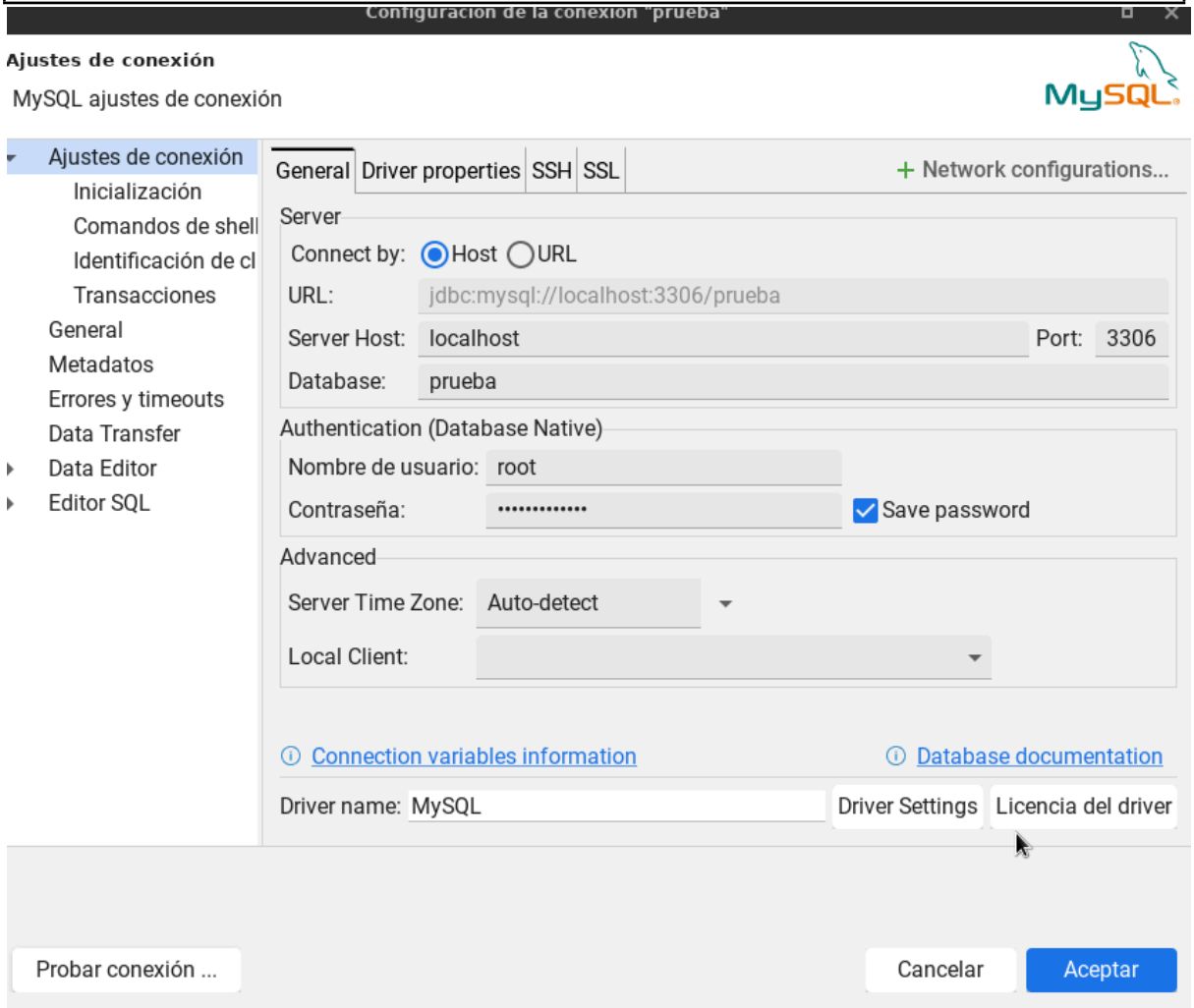
Ver IP servicios:

```
educaand ~ minikube service list
|-------------|---------------------|--------------|---------------------------|
|  NAMESPACE  |        NAME         | TARGET PORT  |            URL            |
|-------------|---------------------|--------------|---------------------------|
| default     | kubernetes          | No node port |                           |
| default     | mysql-service       | No node port |                           |
| default     | spring-boot-service |         8080 | http://192.168.49.2:30000 |
| kube-system | kube-dns            | No node port |                           |
|-------------|---------------------|--------------|---------------------------|
educaand ~ minikube ip
192.168.49.2
```

Conectarse a BBDD:

El comando `kubectl port-forward svc/mysql-service 3306:3306` se utiliza para reenviar puertos de un servicio de Kubernetes a tu máquina local. Aquí, `svc` es una abreviatura para referirse al recurso `Service` en Kubernetes.

```
kubectl port-forward svc/mysql-service 3306:3306
```



K8S

Ver los logs del pod:

Una vez que tengas el nombre del pod, puedes usar `kubectl logs` para ver los logs generados por la aplicación dentro del contenedor:

```
kubectl logs spring-boot-service-xyz
```

Ver los logs de un contenedor específico dentro del pod (si hay varios contenedores)

Si tu pod tiene múltiples contenedores (por ejemplo, un contenedor principal y quizás un sidecar o un contenedor de logging), puedes especificar el nombre del contenedor para obtener los logs específicos de ese contenedo

```
kubectl logs spring-boot-service-xyz -c nombre-del-contenedor
```

Resumen

1. **Configurar Docker para Minikube.**
   a. **`eval $(minikube -p minikube docker-env)`**
2. **Construir la imagen Docker correctamente.**
   a. **`cd /path/to/Tu_Proyecto_Front_Back`**
   b. **`docker build -t spring-boot-app:local -f src-api/Dockerfile .`**
3. **Asegurarse de que el Dockerfile esté correctamente configurado.**
   a. **`docker images`**
4. **Aplicar la configuración de Kubernetes con la imagen correcta.**
   a. **kubectl apply -f k8s/spring-boot-deployment.yaml**
5. **Verificar los pods y logs para identificar problemas adicionales**
   a. **`kubectl get pods`**
   b. **`kubectl get services`**

K8S

Integración con Blockchain (Futuro)

Investigar tecnologías de blockchain adecuadas

- **Explora soluciones como Hyperledger, Ethereum o Corda para validar los documentos.**

Implementar contratos inteligentes

- **Diseñar e implementar contratos inteligentes para la validación de documentos.**

Integrar blockchain con el sistema

- **Añadir lógica en los servicios Spring Boot para interactuar con la blockchain.**