Vamos a plantear el siguiente sistema empresarial sobre el que justificar las respuestas

Sistema empresarial: Plataforma de gestión clínica distribuida

1. Descripción general

Se trata de una plataforma empresarial dedicada a la **gestión integral de pacientes y servicios clínicos** para hospitales y clínicas privadas. Está compuesta por:

- **Aplicación web (frontend):** acceso para pacientes y personal médico.
- **Backend REST API:** gestiona la lógica de negocio, autenticación, historiales, citas, recetas, etc.
- **Base de datos SQL (MariaDB):** almacena toda la información crítica (datos personales, historias clínicas, logs de acceso, etc.).

2. Requisitos técnicos

- **Alta disponibilidad (HA)**: tolerancia a fallos y continuidad 24/7.
- **Consistencia de datos fuerte**: información médica y legalmente sensible.
- **Seguridad**: cifrado, control de acceso, cumplimiento de normativas (GDPR).
- **Escalabilidad horizontal**: número creciente de clínicas y pacientes.
- **Multiregión**: nodos en Europa, América y Asia, con acceso distribuido.

3. Arquitectura base

- Orquestador: **Kubernetes**
- Contenedores Docker:
    - `frontend-app` (React o Angular)
    - `backend-api` (Java Spring Boot, Node.js o Django)
    - `mariadb-db` con Galera
- Almacenamiento distribuido: **CephFS** o **Longhorn**
- Copias de seguridad: Snapshots y backup remoto

Arquitectura de almacenamiento persistente

1. Orquestador de contenedores

- Utilizamos **Kubernetes**, lo cual permite gestionar contenedores de forma distribuida y programática.

2. Base de datos

- Se implementa **MariaDB en modo clúster Galera**, que requiere almacenamiento persistente con consistencia transaccional.

3. Tipo de almacenamiento persistente

- Usamos **volúmenes persistentes (PV)** en Kubernetes mediante **Persistent Volume Claims (PVC)**.

4. Proveedor de almacenamiento

- Se configura el almacenamiento distribuido con **CephFS** (sistema de archivos distribuido POSIX, altamente disponible).

Detalles técnicos del almacenamiento

Volúmenes persistentes (PVC)

- Cada pod de MariaDB accede a su volumen mediante un `StatefulSet`, que asegura que los volúmenes sean **estables y únicos por nodo**.
- Cada PVC está respaldado por CephFS, desacoplado del ciclo de vida de los contenedores.

StorageClass personalizada

- Se define una `StorageClass` de tipo `ReadWriteOnce` con réplica automática y cifrado.
- Se configura el **provisionamiento dinámico**, que permite que Kubernetes cree nuevos volúmenes automáticamente cuando se desplieguen nuevos nodos.

Replicación de datos a nivel de almacenamiento

- CephFS se configura con **réplica 3x**, es decir, cada bloque de datos se almacena en tres nodos distintos del clúster.
- Esta redundancia garantiza **alta disponibilidad** incluso si se pierde un nodo o disco.

Seguridad y cumplimiento

- **Cifrado en reposo** activado en CephFS usando LUKS o nativo del backend (según proveedor).
- **Cifrado en tránsito** mediante TLS en todas las comunicaciones entre pods, nodos y volúmenes.
- **RBAC y control de acceso** en Kubernetes para limitar qué servicios pueden montar volúmenes.

Disponibilidad y multizona

- El clúster de Kubernetes está desplegado en **modo multizona** (por ejemplo, 3 zonas de disponibilidad dentro de una misma región).
- Ceph replica los datos entre zonas, lo que permite tolerar el fallo de una zona completa sin pérdida de disponibilidad.

Gestión de fallos

- Si un nodo con MariaDB cae, Kubernetes reubica el pod en otro nodo **y vuelve a montar automáticamente su volumen PVC**.
- Ceph garantiza que los datos están disponibles en otros nodos sin necesidad de intervención manual.
- Esto asegura la **resiliencia y continuidad operativa** de la base de datos.

Escalabilidad

- Si la carga de usuarios o de datos aumenta, Kubernetes puede escalar horizontalmente los nodos.
- Los volúmenes CephFS son **dinámicamente expandibles**, sin necesidad de downtime ni migraciones manuales.

Conclusión

Para este sistema empresarial clínico en contenedores:

- Se utiliza almacenamiento persistente gestionado por Kubernetes mediante **PVC y CephFS**, con **alta disponibilidad, cifrado y replicación automática**.
- Los volúmenes son resilientes, escalables y seguros.
- La solución está completamente alineada con un entorno de producción real para sistemas críticos como gestión médica.

Replicación sincrónica

Justificación técnica (detallada y razonada):

1. **Consistencia de datos crítica**
    - El sistema maneja información médica, legal y personal.
    - No se puede permitir la existencia de versiones diferentes de una historia clínica en distintos nodos, ni riesgo de pérdida en caso de fallo inmediato tras una escritura.
2. **Sistema de alta criticidad**
    - El backend realiza operaciones transaccionales sensibles: recetas electrónicas, diagnósticos, accesos legales…
    - Es preferible penalizar ligeramente la latencia para garantizar integridad.
3. **Topología multizona pero regional**
    - Aunque el sistema puede estar distribuido en varias zonas de disponibilidad, se despliega **dentro de una misma región geográfica primaria (ej. Europa-Centro)** para minimizar la latencia.
    - Esto permite que la replicación sincrónica sea viable sin comprometer el rendimiento general.
4. **MariaDB Galera Cluster lo soporta nativamente**
    - Utilizamos MariaDB con **Galera Cluster**, que **replica sincrónicamente entre nodos** de forma integrada y transaccional (multi-master).
    - Las escrituras se realizan en un nodo y se **confirman solo cuando todos los nodos aplican el cambio**, eliminando el riesgo de inconsistencias.
5. **Evita conflictos de replicación**
    - En modelos asincrónicos, los conflictos o "escrituras perdidas" requieren reconciliación manual o mecanismos compensatorios, algo inaceptable en un entorno clínico.
    - La replicación sincrónica evita esta complejidad al asegurar atomicidad global en la escritura.
6. **Alineado con normativas legales**
    - En sectores regulados (salud, protección de datos), se exige trazabilidad y consistencia garantizada.
    - La replicación sincrónica asegura cumplimiento normativo (como el GDPR o la LOPDGDD) al no comprometer la integridad de la información.

Nota técnica: Cómo se implementa

- Se despliega un **MariaDB Galera Cluster** con 3 nodos en modo multi-master.
- La replicación sincrónica se aplica **a nivel de transacción**, no por eventos binarios, lo que elimina el riesgo de inconsistencia parcial.
- Se configura el quorum adecuado para evitar *split-brain* en caso de caída de nodos.

- Todos los nodos comparten el mismo estado lógico de base de datos, incluso si hay múltiples clientes escribiendo desde diferentes zonas.

Conclusión

La política de replicación adoptada es **sincrónica**, ya que:

- Garantiza consistencia fuerte, esencial para un sistema clínico.
- Está soportada nativamente por la arquitectura (Galera + Kubernetes).
- Minimiza riesgos legales y técnicos.
- Es compatible con una infraestructura multizona de baja latencia dentro de la misma región.

Aplicación de una Política de Backup y Failover Automático para Asegurar la Alta Disponibilidad

---

Objetivo

Garantizar que el sistema empresarial (plataforma clínica distribuida) **nunca quede inoperativo** ante fallos y que **los datos estén siempre recuperables**, evitando tanto la pérdida de información como el corte del servicio.

FAILOVER AUTOMÁTICO: Alta disponibilidad ante fallos

Cómo se aplica en nuestro sistema

1. Base de datos MariaDB (Galera Cluster)

- Se despliega un clúster Galera de **3 nodos** dentro de Kubernetes.
- Galera utiliza **replicación sincrónica** entre nodos (ya definida en el punto anterior).
- Si un nodo falla, los demás siguen funcionando y aceptando lecturas y escrituras.
- Kubernetes detecta automáticamente el fallo del pod mediante `livenessProbes` y `readinessProbes`, y **recrea el contenedor caído en otro nodo del clúster**.

2. Backend y Frontend

- Se despliegan como **pods con múltiples réplicas**.
- Kubernetes balancea el tráfico con un `Service` tipo `LoadBalancer` y un Ingress Controller (por ejemplo, Traefik o NGINX).
- Si un pod del backend o frontend falla, Kubernetes lo reemplaza automáticamente.
- Si un **nodo físico completo** cae, Kubernetes reprograma todos los pods afectados en otros nodos sanos.

3. Almacenamiento persistente

- Usamos un sistema como **CephFS** o **Longhorn** para que los volúmenes persistentes estén disponibles desde cualquier nodo del clúster.
- Esto permite que un contenedor se reinicie en otro nodo sin perder el acceso a sus datos.

4. DNS y balanceo

- Kubernetes y el Ingress Controller trabajan con un sistema de DNS interno (`CoreDNS`) que actualiza automáticamente las rutas.
- Si un pod cae, el balanceador deja de enrutar tráfico hacia él y lo redirige a los pods activos.

BACKUPS: Copias de seguridad recuperables ante desastre

.

Cómo se aplica en nuestro sistema

1. Frecuencia y retención

- **Diarios**: Backup completo de la base de datos cada noche a las 02:00.
- **Semanales**: Snapshot del sistema completo (base de datos + configuración).
- **Mensuales**: Backup completo almacenado en región secundaria.

  Se conserva:

  - 7 copias diarias.
  - 4 semanales.
  - 3 mensuales.

  Se aplican políticas de rotación automática.

2. Tecnología

- Se utiliza `mariabackup` (binario oficial de MariaDB) dentro de un contenedor que se ejecuta como un **`CronJob` de Kubernetes**.
- Los backups se guardan en un **almacenamiento externo compatible con S3** (como MinIO, AWS S3 o Backblaze).
- Se realiza el **cifrado y firmado GPG** de los archivos para garantizar confidencialidad e integridad.

3. Automatización y monitoreo

- Cada backup genera un log de verificación.
- Se envía una alerta (por correo o webhook) si el backup falla.
- Un contenedor de verificación realiza restauraciones simuladas periódicas en un entorno aislado para validar los backups.

4. Recuperación (Restore)

- Si se detecta corrupción o pérdida total de datos:
  - Se detiene el tráfico a la base de datos.
  - Se recrea el estado de MariaDB restaurando el backup más reciente en los nodos del clúster.
  - Una vez replicado y validado, se reanuda el tráfico.

- Todo este proceso puede ser **automatizado con un pipeline de GitOps** o manualmente ejecutado siguiendo un plan documentado de Disaster Recovery.

Ejemplo práctico de aplicación combinada

- Se cae un nodo físico con un pod de MariaDB → Kubernetes lo detecta y lo reemplaza en otro nodo.
- Se corrompe un volumen → Se reatacha un nuevo PVC desde backup automático de esa mañana.
- Se pierde el clúster completo (incendio del CPD principal) → Se despliega en nueva ubicación, restaurando los backups desde S3, y se retoma el servicio en menos de 1h.

Conclusión

Aplicamos una política de **failover automático** basada en Kubernetes, Galera Cluster y balanceadores, que permite que el sistema siga funcionando ante fallos parciales o completos.

Combinado con una **estrategia sólida de backup cifrado, automatizado, replicado y verificado**, garantizamos que los datos no se pierdan y que el sistema pueda recuperarse ante cualquier incidente mayor, cumpliendo con los requisitos de **alta disponibilidad y continuidad del negocio clínico.**