Respecto del DIAGRAMA DE ARQUITECTURA en sí:

- Agregar explicaciones que ayuden a entender el **diagrama** → escribir en **prosa**.
    - Para evitar ambigüedades en los dibujos, más aún cuando se usan "cajas" y se consideran supuestos, es mejor describir en **prosa** qué son exactamente esas "cajas".

- Aunque no sea obligatorio, un **DIAGRAMA DE CONTEXTO** adicional puede ayudar a entender el alcance de nuestro sistema y con qué otros sistemas éste interactúa.
    - De esos sistemas externos solamente nos interesa qué les pedimos/enviamos.

- **PUNTOS ÚNICOS DE FALLA** → ya hecho el diagrama, preguntarse: ¿dónde se puede romper? ¿dónde puede fallar? ¿se puede reforzar algo para que no se rompa? ¿tengo algún requerimiento de negocio fuerte en ese sentido o solamente me parece a mí?
    - Temas de seguridad → ¿hay un ***firewall***? ¿hay algo más?
    - Temas de conectividad → el asunto de los proveedores.
    - Entre todos los **puntos únicos de falla** presentes, hay que considerar:
        - Los que están en contacto con un sistema de facturación o algo similar.
        - Los que tienen mayor impacto (en términos de consecuencias) si hay una disrupción del funcionamiento del servicio ofrecido.
        - El RTO, es decir, tiempo de recuperación.
    - Para evitar **puntos únicos de falla**, se debe:
        - Agregar redundancia.
        - Tener copias sincrónicas en *cloud*.
        - Tener *backups* de seguridad.
    - Si en la solución hay un **proveedor**, también somos dependientes de ellos.

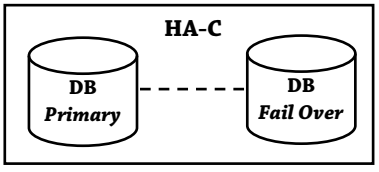
Respecto de la RECUPERACIÓN DE DESASTRES y la CONTINUIDAD DEL NEGOCIO:

- Se debe definir el **desastre** con un escenario posible → es necesario aclarar para qué situación en particular nos vamos a preparar.

- Se debe definir los lineamientos del plan de recuperación de desastres.

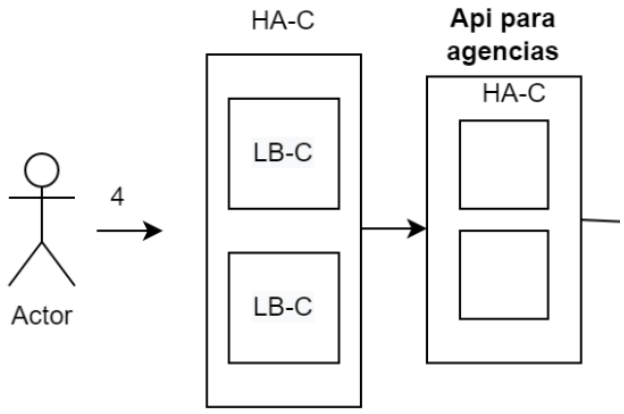
- Tener un *datacenter* de contingencia para garantizar alta disponibilidad es necesario, pero no es suficiente → si ocurre una inundación, deja de haber disponibilidad del sistema.

- **Replicación de DB:** Esquema activo-pasivo con una capa de software que permite, ante la caída de la DB *Primary*, levantar la DB *Fail Over*.



<u>Respecto de los COMPONENTES del DIAGRAMA:</u>

- Si se brinda algún **<u>servicio</u>**, algo/alguien debe consumirlo.
- Si se guarda información en una **<u>BD</u>**, algo/alguien debe leerla.
- Las **<u>API</u>** son reactivas → solamente responden si se les solicita algo.
- Un **<u>balanceador de carga (LB)</u>** reparte la carga entre nodos candidatos a procesar la información, los cuales tienen funcionalidades iguales, pero atienden distintas solicitudes, Detrás del **<u>LB</u>** debe haber varios "receptores" (sean nodos o *clusters*) para que atiendan esas solicitudes → una solicitud es atendida por un único "receptor" (sea un nodo o un *cluster*).
    - En este ejemplo, el **<u>LB</u>** no reparte la carga porque sólo hay un "receptor" (un *cluster* de varios nodos en alta disponibilidad) → las solicitudes no se reparten porque van todas al mismo *cluster*.



    - En este otro ejemplo, el **<u>LB</u>** sí reparte la carga porque hay más de un "receptor" (hay 2 *clusters*, cada uno con varios nodos en alta disponibilidad) → las solicitudes sí se reparten: cada una va a un "receptor" (*cluster*) en particular y no a otro.