Prueba de Arquitectura:

Diseñe la arquitectura para una aplicación que recibirá 40.000 peticiones por minuto, con una BD de My SQL, y tendrá una interfaz para dispositivos móviles Y un sitio web en java, el sistema deberá permitir inscripciones de personas por un formulario web y una app en Android.

Dibuje el esquema de arquitectura especificando numero de servidores, (de BD, backend, Procedimientos almacenados, Apis, Frontend, etc) especifique cada ítem.

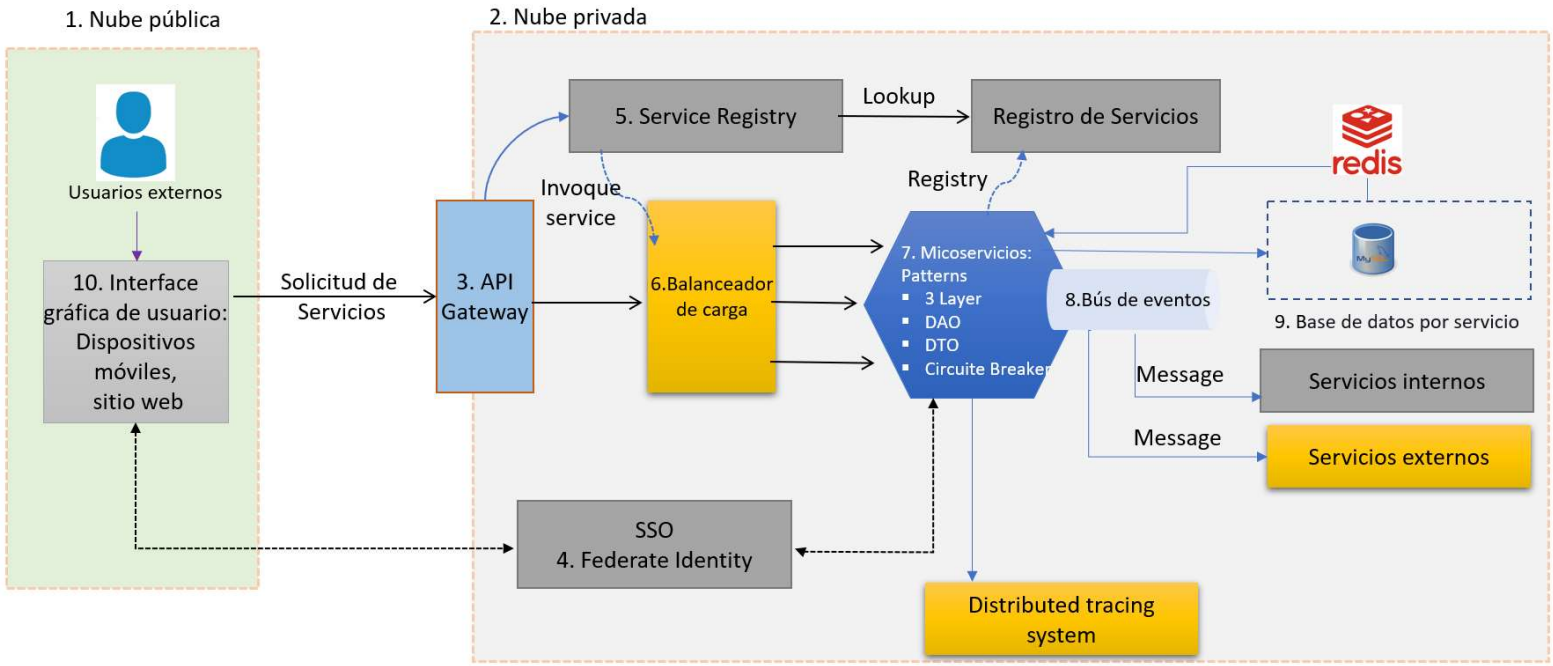
Especifique y justifique si usara Apirestful o webservice, y si estos deben contener consultas o procesos lógicos, especificando el servidor donde se alojarían.

Especifique si utilizara separación de los servicios por api, aplicaciones distribuidas, bróker de mensajería o similares).

Deben explicar todas las decisiones que tomen, Beneficios y desventajas, Y el xq recomiendan esas decisiones.

Que seguridad perimetral recomiendan

Prueba Arquitectura



Sustentación:

Objetivo:

Estructurar a alto nivel el sistema propuesto.

Diseñar una arquitectura de software agnóstica y moderna.

Garantizar un alto desempeño y baja latencia.

Aprovechar los beneficios que ofrecen la implementación de microservicios débilmente acoplados.

Componentes:

1. **Nube pública**: para las aplicaciones para los diferentes canales ofrecidos para los usuarios.

2. **Nube privada**: con varias zonas de disponibilidad con los microservicios requeridos por la solución. Al estar en una red privada impide que cualquiera pueda acceder a los servicios de la red. Esta alberga todos los Microservicios que en su conjunto conforman toda la infraestructura que hace posible el funcionamiento de la aplicación web.

3. **API Gateway:** es la compuerta que permite controlar que recursos o servicios de la infraestructura estarán disponibles en la red pública, para los clientes internos/externos, de esta forma solo se exponen los recursos necesarios.

4. **Federated Identity**: **Seguridad basada en Tokens** implementado el **Single Sign On – SSO,** así cuando un usuario quiera acceder a la aplicación web, tendrá que autenticarse en el SSO enviando su nombre de usuario y contraseña, obtiene un token cómo respuesta que tendrá que guardar en un lugar seguro. Dicho token deberá ser enviado en las posteriores invocaciones al API Gateway para autenticarse.

5. **Registro de servicios y autodescubrimiento:** se requiere de un servidor centralizado dónde se registren los servicios para garantizar la conexión a éstos, conociendo IP y puerto, debido a que al balanceo de cargas se despliegan múltiples instancias de un mismo componente.

6. **Balanceador de carga**: para atender la demanda de manera segura exponiéndose a Internet solo el balanceador la carga se distribuye entre el clúster de servidores disponibles, con una estrategia de escalado horizontal.

7. **Microservicios:** con los servicios encargados de resolver y administrar el workflow/procesos del negocio, éstos pueden ser desarrollado utlizando diferentes patrones arquitectónicos como:
    - 3 capas: capa de presentación, capa de negocio y capa de datos
    - Data Access Object – DAO: separar la lógica de datos de la de servicios.
    - Data Transfer Object – DTO: objetos de transferencia de datos.
    - Circuite Braker para tratar los errores del procesamiento de solicitudes.

Debido a que los Microservicios involucrados operarán en una arquitectura distribuida, desplegados de forma independiente y desacoplados entre sí, y serán accedidos por medio de protocolos de acceso remoto se analizará el uso de SOAP, RESTful, EDA.

- **SOA**

El uso de SOA se justifica en aplicaciones que están pensadas para ser integradas con otras aplicaciones.

Ventajas	Deventajas
Reduce el acoplamiento	Performance: la comunicación por la red y la distribución de los servicios agregar una latencia significativa en la comunicación.
Fácil de probar	No se lleva con grandes volúmenes por petición.
Reutilización	Gobernanza requiere esfuerzos grandes.
Agilidad	Más puntos de falla debido a que es una arquitectura distribuida.
Escalabilidad	
Interoperabilidad	

Razones para no usar SOA debido:

- Para aplicaciones que tiene que procesar o enviar masivas cantidades de datos. Es importante diferenciar entre grandes volúmenes de datos y grandes volúmenes de peticiones, con SOA podemos procesar grandes volúmenes de mensajes, pues podemos clusterizar los servicios para atender muchas peticiones, pero es diferente cuando estas peticiones tienen en solitud o respuesta una masiva cantidad de datos, este último es un inconveniente para SOA pues puede poner en jaque a la arquitectura.

- Debido a que tenemos como objetivo máximo desempeño la arquitectura SOA no es la ideal porque agrega muchas capas que hace que el procesamiento de los datos no sea tan rápido.

- Importante considerar el costo en administración y gobierno, por lo que para empresas pequeñas o que no tiene procesos maduros puede llegar a ser un obstáculo en lugar de una ventaja.

- **RESTful**

Continuando con el objetivo de definir la arquitectura agnóstica de una aplicación con baja latencia se analizará la opción de usar la arquitectura de Representational State Transfer REST, lo cual tiene como ventaja la manera como se transfieren los Recursos, datos ó cualquier cosa que pueda ser recuperada por una referencia de hipertexto (URL), los cuales son enviados en bruto para que el cliente reciba la información, la procese y genere la vista a partir de los datos proporcionados por el servidor, de esta forma, varias aplicaciones ya sean Web, Móviles, escritorio, o en la nube pueden solicitar esa misma información y mostrarla en los formatos requeridos. Utilizando el protocolo de comunicación HTTP.

Ventajas	Deventajas
No está casado con una tecnología.	Siendo una arquitectura distribuida el performance puede verse afectado.
Escalabilidad	Los puntos de fallo pueden ser altos debido a la cantidad de componentes.
Es posible mantener un bajo acoplamiento entre los componentes.	La gobernalidad implica gran esfuerzo, al existir varios equipos de trabajo puede existir duplicación de servicios.
Los **recursos** de REST pueden ser reutilizado por otros componentes y representar los datos de diferentes maneras.	
Fácil de probar.	
Puede ser utilizado para crear integraciones con diferentes dispositivos, incluso con empresas o proveedores externos, que requieren información de nuestros sistemas para opera con nosotros.	

- **EDA**

Para cumplir con el objetivo de tener una arquitectura moderna y escalable se propone combinar la arquitectura dirigida por eventos EDA y RESTFul async implementadas con Microservicios, pudiéndose implementar aplicaciones asíncronas y distribuidas. Durante la operación los Microservicios lanzan eventos entre sí y se produce una reacción de cada uno de ellos, los procesa y posiblemente generará nuevos eventos para que otros componentes los ejecuten.

8. **Bús de eventos:** responsable de administrar todos los eventos de la arquitectura.
   Se propone una Topología con Broker con el objetivo de contar con una una mejor agilidad y poder responder más rápido a los cambios en el flujo de ejecución.

9. **Base de datos por servicio:** Con un acoplamiento flexible definido por la arquitectura de Microservicios, cada uno de éstos puede almacenar y recuperar información de forma independiente de su propio respositoris de datos. Al implementar el patrón de base de datos por servicio, se puede elegir los almacenes de datos más apropiados (por ejemplo, bases de datos relacionales o no relacionales) para su aplicación y requisitos comerciales.

10. Interface gráfica de usuario: Para la interface gráfica de usuario se propone usar una framework crossplatform que permite una desarrollo único para los diferentes canales, como Blazor, Piranha.