**Prueba teórica o de conocimientos técnicos**

El conjunto de escenarios presentados a continuación pretender entender un poco su capacidad de resolución de problemas o incidentes, no hay respuesta mala o incorrecta.

No obstante, debe explicar detalladamente y justificar las soluciones ofrecidas a cadauno de los casos expuestos de manera escrita.

**1. Soluciones para publicar la página web y los Web Services en el mismo puerto y URL (HTTPS - 443):**

Para mí una de las estrategias más recomendadas, que ayudan a mejorar los servicios, es agregar o utilizar un Reverse Proxy (como Nginx o Apache HTTP Server)

Esto permite que el servicio web o aplicación funcionen los dos bajo el mismo dominio, y por el puerto 443, este proxy reverso se realiza con NGINX. De la siguiente manera :

* **Nginx o Apache HTTP Server** se configura para escuchar en el puerto 443 (HTTPS).
* El tráfico HTTPS entrante se redirige de acuerdo con el **path** o la URL. Por ejemplo:
  + Si la solicitud es para la página web (https://mipruebaFinaktiva.com.co/), el proxy redirige el tráfico a **Tomcat en el puerto 8080**.
  + Si la solicitud es para el Web Service (https://mipruebaFinaktiva.com.co/servicio), el proxy redirige el tráfico al puerto **8443** en el mismo servidor o en otro servidor que aloje los Web Services.

De esta forma, todo el tráfico HTTPS se maneja por el puerto 443, pero se direcciona a los diferentes puertos internos de acuerdo con el **path**.

**Ventajas**:

* Permite centralizar la gestión de SSL (certificados de seguridad).
* Flexible, puedes modificar las reglas de redirección fácilmente.
* Buen rendimiento si se configura correctamente.
* Se pueden agregar conten security pólices al proxy reverso usando nginx

Existen una alternativo y es utilizando virtual host con Apache:

Otra opción es configurar el Apache Tomcat para que sirva tanto la página web como los servicios web sobre el mismo puerto 443, utilizando diferentes contextos. Esto se hace a través de la configuración de un virtual host en Tomcat y habilitando el conector HTTPS.

1. En el archivo server.xml de Tomcat, configurar el conector HTTPS para escuchar en el puerto 443.
2. Luego, dentro de la configuración de Tomcat, se pueden configurar diferentes contextos para cada servicio:
   * El contexto raíz (/) puede apuntar a la página web.
   * El contexto /servicio puede apuntar a los Web Services.

De esta manera, el tráfico HTTPS entrante en el puerto 443 será dirigido al contexto correcto dentro de Tomcat.

Ventajas:

* Menos infraestructura adicional que implementar.
* Útil si no se quiere agregar un proxy adicional.

**2. Razones por las cuales el cliente no puede acceder a los servicios, pero internamente sí:**

**Razón 1: Configuración de DNS**

Es posible que el cliente esté utilizando un servidor DNS que no resuelve correctamente el dominio mipruebaFinaktiva.com.co. Para confirmar esto, se puede probar desde otro dispositivo o usar herramientas como nslookup o dig para verificar si la resolución DNS es correcta.

Solución: Verifique que los registros DNS de mipruebaFinaktiva.com.co estén correctamente configurados y apunten a la IP pública del servidor.

**Razón 2: Problemas con el firewall**

Si el servidor tiene un firewall activado, es posible que esté bloqueando las conexiones externas al puerto 443, pero permita el tráfico interno. Verifique las reglas del firewall para asegurarse de que el tráfico entrante en el puerto 443 está permitido.

Solución: Abra el puerto 443 en el firewall para permitir las conexiones externas.

**Razón 3: Problemas con la configuración del servidor web o proxy**

Si se está utilizando un reverse proxy (como Nginx o Apache), podría haber un problema de configuración que permita el acceso interno pero no externo. Es importante revisar si existe algún log o error en el servidor web, y con este validar una posible solución según el análisis. , adicional es importante también validar si no existe algún problema de configuración de servidor o bloqueo de firewall.

**Razón 4: Restricciones geográficas o de red**

Es importante validar si existe algún WAF o restricción de ips para acceder a el servicio, o alguna restricción geográfica en la red, validar políticas del firewall y configuraciones aplicadas al servicio o portal web.

**Razón 5: Certificados SSL**

Es posible que el servidor o servicio no tenga un certificado SSL valido y esto puede que este generando problemas de carga al momento de acceder al sitio web o servicio, se debe de validar los certificados utilizados y validar la vigencia, si estos se encuentran vencidos se debe de actualizarlos y validar si el servicio es funcional de nuevo.

**3. Algoritmos de balanceo de carga:**

**Round Robin**

Este es uno de los algoritmos más comunes. Consiste en distribuir las solicitudes de forma equitativa entre los servidores backend. Cada nueva solicitud se envía al siguiente servidor de la lista, en un ciclo continuo.

**Uso recomendado**: Es ideal cuando los servidores tienen capacidades similares y no se requiere un balanceo más avanzado.

**Round Robin Weighted**

Este algoritmo es similar al Round Robin, pero asigna un "peso" a cada servidor. Los servidores con mayor peso reciben más solicitudes que los servidores con menor peso.

**Uso recomendado**: Útil cuando tienes servidores con diferentes capacidades (por ejemplo, uno más potente que otro) y deseas distribuir el tráfico de acuerdo con el rendimiento esperado de cada servidor.

**Least Connections**

Este algoritmo envía la nueva solicitud al servidor con la menor cantidad de conexiones activas en ese momento. Esto es útil para balancear la carga de manera dinámica, basándose en el número de conexiones activas.

**Uso recomendado**: Es ideal cuando las solicitudes pueden durar más o menos tiempo, y se quiere asegurar de que los servidores con menos carga reciban más tráfico.

**Source IP Hash**

El tráfico se distribuye entre los servidores según una "hash" de la IP de origen. Esto significa que la misma IP de origen siempre será enviada al mismo servidor backend, lo que puede ser útil para sesiones persistentes.

**Uso recomendado**: Es útil cuando se necesita mantener la sesión de un cliente en el mismo servidor, como en aplicaciones de comercio electrónico donde la consistencia de la sesión es importante.

**URI (Uniform Resource Identifier)**

Este algoritmo distribuye el tráfico según el URI de la solicitud, lo que significa que las solicitudes que se refieren a la misma URL serán enviadas al mismo servidor.

**Uso recomendado**: Es útil para aplicaciones donde diferentes rutas o servicios deben ser manejados por servidores específicos. Por ejemplo, un servidor para la página principal y otro para los servicios web.

Gracias por la oportunidad, me parecido muy divertido el reto y la investigación que se debe de realizar.