Reto Técnico Ingeniero Cloud

Preguntas teóricas:

1. ¿Cuál es la diferencia entre nube pública, privada e hibrida?

   Nube Pública: Es un servicio ofrecido por terceros proveedores, accesible a cualquier persona. Los recursos se comparten entre varios clientes y es accesible mediante internet.

   Nube Privado: Es un entorno dedicado exclusivamente a un único cliente. Puede alojarse internamente dentro de la empresa o gestionarse por un proveedor externo. Ofrece mayor control y seguridad.

   Nube Híbrida: Combina elementos de nubes públicas y privadas, permitiendo transferir datos y aplicaciones entre ambas. Esta solución ofrece flexibilidad, lo que permite a las empresas aprovechar la nube pública para tareas menos sensibles y mantener datos críticos en un entorno privado. El objetivo de esta nube es aprovechar las fortalezas de ambos enfoques para optimizar el rendimiento, la seguridad y los costos.

2. Describa tres prácticas de seguridad en la nube.

   Cifrado de datos: Mecanismo que permite proteger la información gracias a la capacidad de transformar los datos legibles a ilegibles, de esta forma los datos serán utilizables solo si tenemos la clave de cifrado.

   Control de Acceso basado en roles RBAC: Principio para que las personas solo tengan los permisos que su rol necesite para acceder a recursos específicos, esto brinda una segregación basada en el uso real de los recursos en la nube.

   Monitoreo y Auditoria: Herramientas y proceso para realizar supervisiones sobre las actividades realizadas en la nube, ambas de la mano permiten identificar cualquier actividad no planificada, esto facilita la detección y reacción rápida ante posibles amenazas o incidentes de seguridad.

3. ¿Qué es la IaC, y cuáles son sus principales beneficios?, mencione 2 herramientas de iaC y sus principales características.

   La infraestructura como código (IaC) es un enfoque para gestionar la infraestructura utilizando código y automatización en lugar de procesos manuales, permite a los equipos implementar y gestionar recursos rápidamente, así como destruirlos si la fase de implementación lo requiere.

   Principales beneficios:

   a. Control de Versiones
   b. Automatización
   c. Escalabilidad
   d. Flexibilidad
   e. Integración con automatizaciones CI/CD
   f. Recuperacion anti-desastres
   g. Auditoria

   Terraform: Enfocado en provisión y gestión de infraestructuras.

   a) **Sintaxis declarativa**: Utiliza HCL para describir el estado deseado de la infraestructura, lo que permite a los usuarios definir los recursos necesarios sin tener que escribir instrucciones paso a paso.
   b) **Estado Persistente:** Almacena el estado actual de la infraestructura en un archivo de estado, lo que le permite rastrear cambios y administrar dependencias, asegurando actualizaciones correctas.
   c) **Proveedores y extensibilidad:** Admite múltiples proveedores de nube y le permite crear sus propios proveedores para integrarlos con otros sistemas, lo que la convierte en una herramienta versátil para administrar diferentes entornos.

   Ansible: Enfocado en configuración y gestión de sistemas (automatización de tareas y configuraciones)

   a) **Simplicidad y legibilidad:** Ansible utiliza YAML para escribir playbooks, lo que hace que las configuraciones sean comprensibles y accesibles incluso para personas sin conocimientos técnicos profundos. Esto facilita la colaboración entre los equipos de desarrollo y operaciones.
   b) **Arquitectura sin agentes:** Ansible solo utiliza SSH para conectarse (WinRM para windows). Esto simplifica la implementación y la administración, lo que minimiza la sobrecarga y mejora la seguridad.
   c) **Idempotencia:** Ansible proporciona idempotencia, lo que significa que ejecutar un playbook varias veces dará como resultado el mismo estado del sistema. Esto ayuda a prevenir errores y mantener la estabilidad en la producción.

4. ¿Qué métricas considera esenciales para el monitoreo de soluciones en la nube? Aunque depende mucho del escenario y las aplicaciones que se usarán, las métricas mas comunes a considerar son las siguientes:

    a) **Uso de CPU:** Mide el porcentaje de potencia de CPU utilizada por un recurso, ayuda a determinar si una instancia está sobrecargada o tiene recursos infrautilizados.
    b) **Memoria:** Monitorea la cantidad de memoria utilizada por un recurso, esto es importante para identificar cuellos de botella en la memoria.
    c) **I/O de disco:** Incluye métricas como velocidades de lectura y escritura en discos, le permite monitorear el rendimiento del almacenamiento y detectar problemas de latencia.
    d) **Rendimiento de la red:** Mide la cantidad de datos enviados y recibidos por un recurso, ayuda a comprender el uso de la red y detectar posibles cuellos de botella.
    e) **Recuento de solicitudes:** Mide la cantidad de solicitudes realizadas al equilibrador de carga o puerta de enlace API, ayuda a comprender la carga de tráfico y puede proporcionar información sobre la escalabilidad y capacidad de la aplicación.
    f) **Estado de salud de la instancia:** Metrica para supervisar el estado de las instancias en un grupo de Auto Scaling o balanceador de carga, esto nos permite detectar casos donde un grupo de instancias esta con errores y tomar medidas correctivas.
    g) **Errores de solicitud:** Mide la frecuencia de errores en aplicaciones y servicios, como errores 4xx y 5xx, sumamente importante para identificar problemas en la aplicación y la infraestructura.

5. ¿Qué es Docker y cuáles son sus componentes principales?

    Docker es una plataforma de contenedorización de aplicaciones que permite a los desarrolladores empaquetar aplicaciones y todas sus dependencias en contenedores. Estos contenedores están aislados entre sí y pueden operar en cualquier entorno, proporcionando coherencia y portabilidad

    Dentro de los componentes de Docker está:

a) **Docker Engine:** Es el proceso principal que gestiona contenedores e imágenes. Se ejecuta en segundo plano y es responsable de crear, iniciar y detener contenedores.
b) **CLI de Docker:** El cliente Docker es una interfaz de línea de comandos (CLI) que permite a los usuarios interactuar con Docker Engine mediante comandos.
c) **Imagen de Docker:** Una imagen de Docker es una plantilla inmutable que se utiliza para crear contenedores, incluye todos los archivos, dependencias e instrucciones necesarios para ejecutar la aplicación.
d) **Registro Docker:** El registro es un repositorio de imágenes de Docker donde los usuarios pueden cargar y descargar imágenes., un ejemplo es Docker Hub que es un registro público popular con muchas imágenes públicas y oficiales disponibles.
e) **Contenedor:** Un contenedor es una instancia aislada de una imagen que se ejecuta en Docker. Contiene la aplicación y todas sus dependencias, permitiéndole ejecutarse en cualquier entorno sin necesidad de configuración.

6. Caso práctico

Cree un diseño de arquitectura para una aplicación nativa de nube considerando los siguientes componentes:

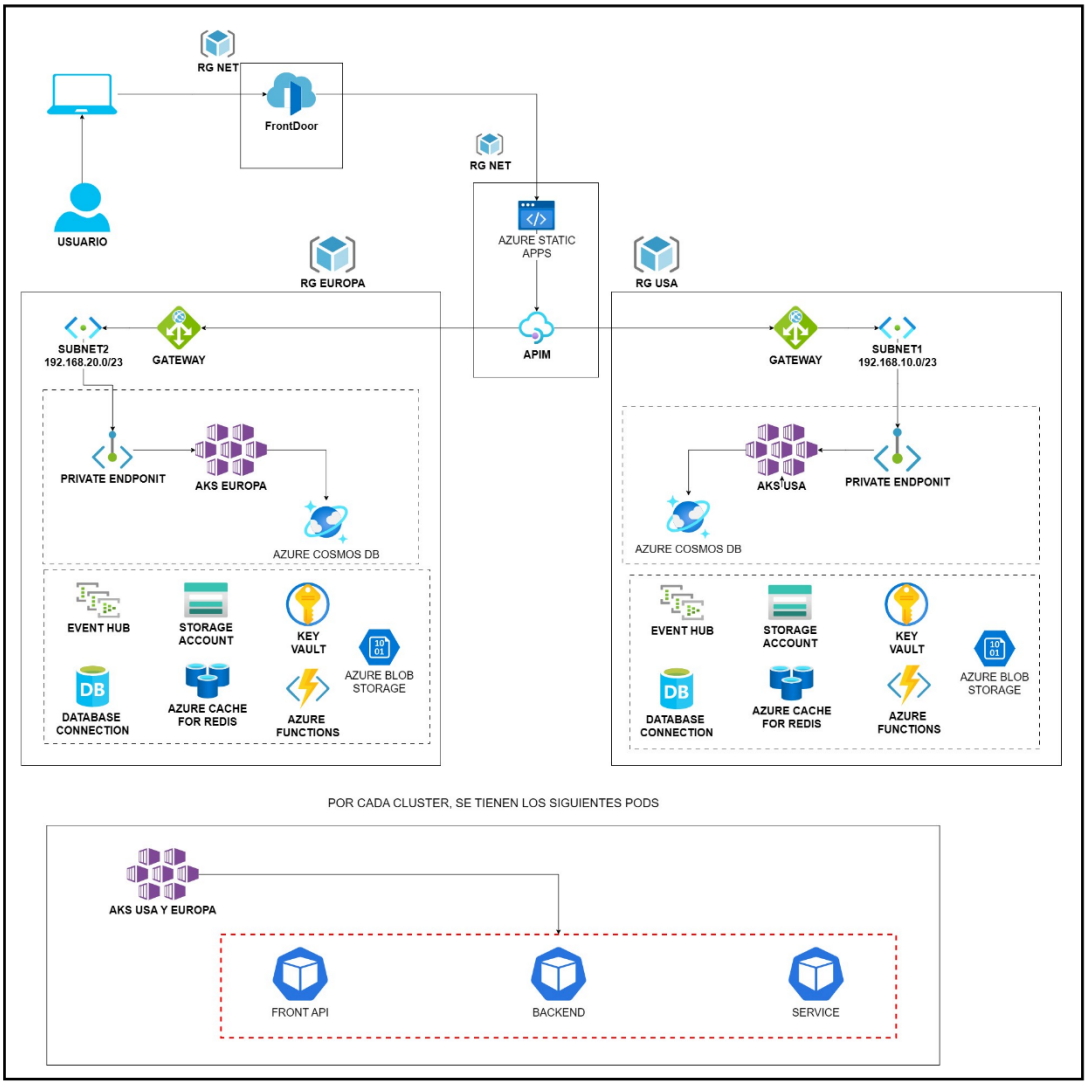
Frontend: Una aplicación web que los clientes utilizarán para navegación.

Backend: Servicios que se comunican con la base de datos y el frontend.

Base de datos: Un sistema de gestión de base de datos que almacene información.

Almacenamiento de objetos: Para gestionar imágenes y contenido estático.

https://drive.google.com/file/d/1-AT-Dzmzas5zL4CVCS9iRt5LFuN1F0tF/view?usp=sharing



Seleccione un proveedor de servicios de nube (Aws, Azure o GCP) y sustente su selección.

Si bien las tres nubes publicas ofrecen los mismos servicios hoy se optó por Azure como proveedor por su amplia gama de servicios que se integran eficientemente, tales como Azure DevOps y AKS facilitan el ciclo de vida de las aplicaciones, desde el desarrollo hasta la implementación.

Además, Azure destaca por su fuerte enfoque en la seguridad y el cumplimiento, las herramientas de gestión de identidades y las opciones de cifrado aseguran que los datos y aplicaciones estén protegidos en un entorno de amenazas complejas.

Finalmente, Azure se integra eficazmente con soluciones de Microsoft, como Microsoft 365, facilitando la adopción de la nube y aprovechando inversiones existentes.

a) **Azure Front Door:** Se utiliza como un punto de entrada global para la aplicación, proporciona una única dirección pública para los usuarios, asegurando una distribución eficiente del tráfico entre las regiones (Europa y USA) para optimizar el rendimiento y la disponibilidad.
b) **Resource Groups (RG) por región:** Se agrupan los recursos en Resource Groups por región para facilitar la gestión y el control de los recursos específicos de cada localización.
c) **API Management (APIM):** Se decidió APIM por su funcion de facilitar la publicación, administración, protección y análisis de las APIs, asegurando un control eficiente sobre las versiones de la API y la seguridad.
d) **Azure Static Web Apps:** Usamos este recursod para alojar aplicaciones web front-end estáticas con fácil integración en CI/CD y APIs backend usando Azure Functions. Esto es ideal para aplicaciones ligeras con una capa de presentación optimizada.
e) **Azure Kubernetes Service (AKS):** Se optó por usar AKS en ambas regiones (Europa y USA) para desplegar los contenedores que ejecutan los pods (front API, backend y servicio). Se decidió AKS ya que proporciona una plataforma escalable y administrada para ejecutar microservicios en contenedores, soportando múltiples réplicas y garantizando alta disponibilidad.
f) **Gateways y Subnets:** Permite una separación clara del tráfico interno y externo, así como la conectividad segura entre los recursos de la red interna, como AKS y los endpoints privados.
g) **Azure Cosmos DB y Private Endpoints:** Azure Cosmos DB proporciona una base de datos distribuida a nivel global y se usa para asegurar que los datos sean accesibles de manera rápida y escalable en ambas regiones, minimizando la latencia. Los private endpoints aseguran que el tráfico hacia Cosmos DB permanezca dentro de la red privada de Azure.
h) **Azure Event Hub, Storage Account, Azure Functions, Key Vault, Azure Cache for Redis:** Estos servicios habilitan características específicas como el procesamiento en tiempo real de eventos, almacenamiento de grandes volúmenes de datos, ejecución de funciones bajo demanda, y mejora del rendimiento con Redis.
i) **Pods (Front API, Backend, Service):** El uso de contenedores para la lógica de negocio permite una fácil escalabilidad y administración, con separación clara entre los componentes de la aplicación, asegurando una arquitectura de microservicios ágil y distribuida.

Beneficios de la arquitectura propuesta:

I. **Alta disponibilidad**: Distribución en dos regiones garantiza la continuidad del servicio.
II. **Escalabilidad:** AKS permite escalar horizontalmente las aplicaciones de manera eficiente.
III. **Seguridad:** Private Endpoints y Key Vault protegen los datos y las credenciales.
IV. **Optimización de rendimiento:** Azure Front Door y Redis Cache optimizan el tiempo de respuesta y el rendimiento global de la aplicación.

Componente	Servicio en la imagen
Frontend	Azure Static Web Apps
Backend	AKS (Pods: Front API, Backend, Service)
Base de datos	Azure Cosmos DB
Almacenamiento de objetos	Azure Blob Storage

Para algún otro escenario donde la arquitectura se defina con un proveedor en específico se tomaría en consideración los siguientes servicios.

Azure	AWS	GCP
Azure Static Web Apps	AWS Amplify / Amazon S3 + CloudFront	Firebase Hosting / App Engine
Azure Kubernetes Service (AKS)	Amazon EKS	Google Kubernetes Engine (GKE)
Azure Cosmos DB	Amazon DynamoDB	Google Firestore / Cloud Spanner
Azure Blob Storage	Amazon S3	Google Cloud Storage
Azure API Management (APIM)	Amazon API Gateway	Google Cloud Endpoints / Apigee
Azure Front Door	Amazon CloudFront	Google Cloud Load Balancing
Azure Virtual Network	Amazon VPC	Google VPC
Azure Functions	AWS Lambda	Google Cloud Functions
Azure Key Vault	AWS Secrets Manager / KMS	Google Secret Manager / Cloud KMS
Azure Event Hub	Amazon Kinesis	Google Pub/Sub
Azure Cache for Redis	Amazon ElastiCache for Redis	Google Memorystore for Redis
Azure Monitor / Log Analytics	Amazon CloudWatch	Google Cloud Monitoring / Logging
Azure SQL Database /	Amazon RDS (SQL, PostgreSQL, etc.)	Google Cloud SQL / Cloud Spanner

Diseñe una arquitectura de nube. Incluya diagramas que representen la arquitectura y justifique sus decisiones de diseño (Utilice https://app.diagrams.net/)