Preguntas Teóricas:

1. Diferencia entre nube pública, privada e híbrida
   - Nube Pública: Servicios ofrecidos por terceros proveedores a través de Internet, accesibles para cualquier usuario. Los recursos (servidores, almacenamiento, redes) se comparten con otros clientes en un entorno multiinquilino. Ejemplos: AWS, Google Cloud, Azure.

   - Nube Privada: Infraestructura dedicada exclusivamente a una organización, ya sea gestionada internamente o por un proveedor externo. Ejemplo: Azure Stack.

   - Nube Híbrida: Combinación de nubes públicas y privadas, permitiendo la portabilidad de datos y aplicaciones entre ellas.

2. Tres prácticas de seguridad en la nube
   1. Cifrado de Datos: Asegurar que los datos en tránsito y en reposo estén cifrados para protegerlos de accesos no autorizados. Esto garantiza la confidencialidad y la integridad de la información.

   2. Gestión de Identidades y Accesos (IAM): Implementar políticas estrictas de control de acceso y autenticación multifactor (MFA) para garantizar que solo los usuarios autorizados accedan a los sistemas y datos críticos.

   3. Monitoreo y Auditoría Continua: Utilizar herramientas de monitoreo para detectar y responder a actividades sospechosas y realizar auditorías regulares de seguridad. Esto ayuda a detectar comportamientos sospechosos o posibles brechas de seguridad.

3. ¿Qué es la IaC y cuáles son sus principales beneficios?
La Infraestructura como Código (IaC): Práctica de gestionar y aprovisionar la infraestructura a través de archivos de configuración legibles por máquina en lugar de configuraciones manuales. Los principales beneficios de la IaC incluyen:

   - Beneficios:
     - Automatización: Reducción de errores humanos y mayor consistencia.
     - Versionamiento: Capacidad de rastrear y revertir cambios en la infraestructura.
   - Herramientas de IaC:
     - Terraform:
       - Características: Proveedor agnóstico, permite la gestión de infraestructura en múltiples nubes.
     - Ansible:
       - Características: Basado en YAML, no requiere agentes, ideal para configuración y orquestación.

4. Métricas esenciales para el monitoreo de soluciones en la nube
   1. Disponibilidad: Tiempo de actividad y tiempo de inactividad de los servicios.
   2. Rendimiento: Latencia, tiempo de respuesta y throughput.
   3. Uso de Recursos: CPU, memoria, almacenamiento y red.
   4. Seguridad: Intentos de acceso fallidos, vulnerabilidades detectadas y eventos de seguridad.

5. ¿Qué es Docker y cuáles son sus componentes principales?
Docker es una plataforma de contenedores que permite empaquetar aplicaciones y sus dependencias en contenedores aislados y portables.

- Componentes Principales:
  - Docker Engine: Motor que permite la creación y ejecución de contenedores.
  - Docker Images: Plantillas inmutables que contienen todo lo necesario para ejecutar una aplicación.
  - Docker Containers: Instancias en ejecución de imágenes de Docker.
  - Docker Hub: Repositorio en línea para compartir y almacenar imágenes de Docker.

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.

Solución:

Previo al diseño, se requiere información adicional para una solución óptima:

- Caracterización de la aplicación web: Definir el tipo de aplicación (e-commerce, red social, etc.) y sus patrones de uso para determinar los requerimientos de escalabilidad, rendimiento y latencia.
- Especificación de la base de datos: Detallar el tipo de datos a almacenar (transaccionales, relacionales, NoSQL) y el volumen estimado para la selección del sistema de gestión de base de datos (DBMS) adecuado.
- Requerimientos no funcionales: Identificar las necesidades de seguridad, cumplimiento normativo (GDPR, HIPAA), disponibilidad y tolerancia a fallos.

Soluciones Enfocadas a la Arquitectura

He preparado 2 modelos de arquitectura el primero enfocado en “microservicios” y el segundo enfocado en “Serverless”

1. Arquitectura basada en Microservicios con Balanceo de Carga:
   - Frontend: Implementación en un servicio de hosting estático (AWS S3, Google Cloud Storage) con distribución a través de una Content Delivery Network (CDN) para optimizar la latencia y el rendimiento.
   - Backend: Descomposición en microservicios independientes, desplegados en contenedores (Docker, Kubernetes) y gestionados por un orquestador de contenedores. Se utiliza un balanceador de carga para distribuir el tráfico entre las instancias de los microservicios, garantizando alta disponibilidad y escalabilidad.

- Base de datos: Implementación de una base de datos distribuida (Amazon DynamoDB, Google Cloud Spanner) para asegurar escalabilidad horizontal, alta disponibilidad y consistencia de datos.
- Almacenamiento de objetos: Utilización de un servicio de almacenamiento de objetos (AWS S3, Google Cloud Storage) para el almacenamiento eficiente y escalable de imágenes y contenido estático.

2. Arquitectura Serverless:
   - Frontend: Similar a la solución 1, con despliegue en hosting estático y distribución mediante CDN.
   - Backend: Implementación mediante funciones serverless (AWS Lambda, Google Cloud Functions) activadas por eventos, optimizando el uso de recursos y la escalabilidad bajo demanda.
   - Base de datos: Utilización de una base de datos serverless (Amazon Aurora Serverless, Google Cloud Firestore) con escalado automático en función de la demanda, simplificando la administración y reduciendo costos.
   - Almacenamiento de objetos: Análogo a la solución 1, utilizando un servicio de almacenamiento de objetos.

Sustentando la Arquitectura Seleccionada.

Ambas arquitecturas expuestas previamente son adecudas para desplegar en Azure (Proveedor de servicio de nube seleccionado), para ambos casos se analizara las ventajas y desventajas.

1. Arquitectura basada en Microservicios con Balanceo de Carga en Azure:
   a. Ventajas:
      i. Escalabilidad y alta disponibilidad: Azure Kubernetes Service (AKS) facilita la orquestación de microservicios en contenedores, permitiendo el escalado automático y la gestión eficiente de recursos. Azure Load Balancer distribuye el tráfico entre las instancias, garantizando alta disponibilidad.
      ii. Flexibilidad y agilidad: La arquitectura de microservicios permite el desarrollo y despliegue independiente de cada componente, facilitando la integración continua y la entrega continua (CI/CD).
      iii. Amplio ecosistema de servicios: Azure ofrece una gama completa de servicios para complementar la arquitectura, como Azure API Management para la gestión de APIs, Azure Monitor para la supervisión y Azure Key Vault para la gestión de secretos.
   b. Desventajas:
      i. Complejidad: La gestión de una arquitectura de microservicios puede ser compleja, requiriendo herramientas y conocimientos específicos para la orquestación, el monitoreo y la gestión de dependencias.
      ii. Costos: Si bien la escalabilidad puede optimizar costos, la complejidad de la arquitectura puede requerir una inversión inicial mayor en infraestructura y herramientas.

Fuente: https://learn.microsoft.com/es-es/azure/architecture/reference-architectures/containers/aks-microservices/aks-microservices

2. Arquitectura Serverless en Azure:
   a. Ventajas:
      i. Simplicidad y eficiencia: Azure Functions permite la ejecución de código sin necesidad de administrar servidores, simplificando el desarrollo y la gestión. El escalado automático se realiza en función de la demanda, optimizando el uso de recursos y reduciendo costos.

ii. Rápido desarrollo: El enfoque serverless acelera el desarrollo y la implementación de aplicaciones, permitiendo a los desarrolladores centrarse en la lógica de negocio.
iii. Integración con otros servicios Azure: Azure Functions se integra fácilmente con otros servicios como Azure Cosmos DB, Azure Event Hubs y Azure Logic Apps, facilitando la creación de soluciones complejas.

b. Desventajas:
   i. Limitaciones de tiempo de ejecución: Las funciones serverless tienen limitaciones en el tiempo de ejecución, lo que puede ser un problema para procesos de larga duración.
   ii. Dependencia de la plataforma: La arquitectura serverless implica una mayor dependencia de la plataforma Azure, lo que puede dificultar la migración a otros proveedores en el futuro.

Fuente: https://learn.microsoft.com/es-es/azure/architecture/web-apps/serverless/architectures/web-app

Finalmente, Para este caso, y sin información adicional sobre los requerimientos específicos de la aplicación, me inclinaría por la **arquitectura Serverless en Azure**. Azure Functions, en combinación con servicios como Azure Cosmos DB para la base de datos y Azure Blob Storage para el almacenamiento de objetos, ofrece una solución escalable, eficiente y de fácil gestión, ideal para aplicaciones nativas de nube.