Prueba Técnica - Ingeniero Cloud/DevOps Semi-Senior

Introducción

El presente documento responde a la prueba técnica solicitada para la postulación al puesto de Ingeniero Cloud / DevOps Semi-Senior, abordando conceptos clave relacionados con computación en la nube, seguridad, automatización, monitoreo y contenedores, a través de una serie de preguntas teóricas.

Adicionalmente, se desarrolla un caso práctico enfocado en el diseño de una arquitectura cloud para una aplicación web, planteado como un Proof of Concept (PoC). La solución prioriza buenas prácticas, claridad en las decisiones técnicas y un enfoque realista, complementandose con el uso de Infraestructura como Código (IaC) para modelar los componentes principales de la propuesta.

1. Preguntas teóricas

1.1. Diferencia entre nube pública, privada e híbrida

La principal diferencia entre la nube pública, privada e híbrida radica en el modelo de acceso, propiedad de la infraestructura y forma de operación. En la nube pública, los recursos son provistos por un proveedor externo y están disponibles para múltiples clientes, permitiendo un acceso amplio a través de internet bajo esquemas de autenticación y pago por uso.

Por otro lado, la nube privada está diseñada para una sola organización, ya sea en infraestructura propia o dedicada, lo que ofrece mayor control sobre la seguridad, la configuración y el cumplimiento de políticas internas. La nube híbrida combina ambos modelos, integrando recursos de nube pública y privada, lo que permite aprovechar la escalabilidad de la nube pública mientras se mantienen ciertos sistemas o datos sensibles en entornos privados, según las necesidades del negocio.

1.2 Prácticas de seguridad en la nube

En mi experiencia, algunas de las prácticas más importantes en seguridad cloud son las siguientes:

- **Segmentación de red y control de tráfico:** es fundamental manejar subredes para restringir el acceso a los distintos servicios, complementándolo con reglas de firewall que permitan exponer únicamente los servicios estrictamente necesarios. Esto reduce la superficie de ataque y evita accesos no autorizados dentro de la red.

- **Control de acceso e identidades:** se debe gestionar el acceso a los recursos mediante la asignación de permisos limitados, aplicando el principio de mínimo

privilegio, de modo que cada usuario o servicio cuente únicamente con los permisos necesarios para cumplir su rol.

- **Gestión segura de credenciales y secretos:** es importante evitar el uso de credenciales embebidas en código y gestionar secretos de forma centralizada, asegurando que solo los servicios que lo requieran tengan acceso a ellos. Adicionalmente, se debe verificar que únicamente los servicios necesarios estén expuestos al exterior.

1.3 Infrastructure as Code (IaC)

La Infraestructura como Código (IaC) es un enfoque fundamental en entornos cloud, ya que permite definir y gestionar la infraestructura de manera declarativa y reproducible. Gracias a IaC, es posible crear entornos idénticos en distintas cuentas o ambientes del mismo proveedor, reduciendo errores manuales y facilitando la estandarización de la infraestructura.

Adicionalmente, IaC permite un mayor control sobre los recursos creados, ya que la infraestructura se puede versionar y auditar de la misma forma que el código de una aplicación. Esto facilita su integración con pipelines de CI/CD para automatizar procesos de despliegue e integración. Entre las herramientas más utilizadas se encuentra Terraform, que destaca por su enfoque declarativo y su capacidad de trabajar en entornos multinube, además de las herramientas nativas que cada proveedor cloud ofrece para la gestión de su infraestructura.

1.4 Métricas esenciales para el monitoreo en la nube

Desde mi experiencia, algunas de las métricas más importantes para el monitoreo en la nube son las siguientes:

- **Tiempo de respuesta:** permite evaluar el rendimiento de los servicios y tomar acciones de mejora según las necesidades del proyecto. Por ejemplo, si un servicio presenta tiempos de respuesta elevados, puede ser necesario ajustar su configuración o escalar recursos para mejorar el desempeño.

- **Costos:** el monitoreo del consumo y los costos asociados a los servicios cloud es fundamental, especialmente en proyectos con presupuestos limitados. Analizar estas métricas permite optimizar configuraciones y seleccionar los recursos que mejor se ajusten a las necesidades reales del sistema.

- **Logs y alertas:** los registros de los sistemas permiten detectar de forma temprana fallos en la ejecución del código o problemas a nivel de infraestructura. Complementar los logs con alertas configuradas según distintas condiciones facilita una respuesta rápida ante incidentes y mejora la estabilidad del entorno.

1.5 Docker y sus componentes principales

Docker es una herramienta que permite crear entornos de ejecución portables y reproducibles, facilitando la estandarización de aplicaciones y evitando problemas de configuración entre distintos entornos. Gracias a Docker, es posible definir cómo debe ejecutarse una aplicación junto con sus dependencias, además de permitir ejecuciones parametrizadas mediante variables de entorno.

Entre los componentes principales de Docker se encuentran:

- **Imágenes**: son plantillas predefinidas que contienen el sistema base y las dependencias necesarias para ejecutar una aplicación. A partir de ellas se pueden crear y modificar imágenes según los requerimientos del proyecto.

- **Contenedores**: son instancias en ejecución de una imagen. Permiten ejecutar aplicaciones de forma aislada y pueden configurarse mediante parámetros o variables de entorno. Su sistema de archivos es escribible pero efímero, por lo que cualquier información que deba persistir debe almacenarse en volúmenes.

- **Volúmenes**: permiten persistir datos fuera del ciclo de vida del contenedor, evitando la pérdida de información. Son especialmente útiles en entornos de desarrollo y en aplicaciones que requieren almacenamiento persistente.

- **Docker Daemon**: es el servicio encargado de gestionar los contenedores, imágenes, redes y volúmenes. La línea de comandos de Docker se comunica con el daemon para ejecutar las distintas operaciones.

- **Redes**: permiten definir cómo se comunican los contenedores entre sí, creando redes aisladas donde solo ciertos servicios pueden interactuar.

- **Docker Compose**: facilita la definición y ejecución de entornos compuestos por múltiples contenedores, permitiendo configurar servicios, redes y volúmenes de forma declarativa.

- **Dockerfile**: es el archivo que define cómo se construye una imagen. Se recomienda utilizar un enfoque de multi-stage build para optimizar el tamaño final de las imágenes y mejorar su eficiencia.

2. Caso Práctico: Diseño de Arquitectura Cloud

2.1 Objetivo del diseño

El objetivo de este diseño es proponer una arquitectura en la nube para una aplicación web compuesta por distintos componentes, como frontend, backend, base de datos y almacenamiento de archivos. La propuesta se plantea como un Proof of Concept (PoC), con el fin de definir una base arquitectónica clara y coherente.

En este diseño se priorizan aspectos como la correcta comunicación entre los componentes, la seguridad del sistema y la incorporación de mecanismos de monitoreo que permitan observar el comportamiento de la infraestructura y de la aplicación.

2.2 Selección del proveedor de nube

Para este ejercicio se opta por Microsoft Azure como proveedor de nube, debido a que ofrece un modelo claro de gobernanza y organización de recursos. Azure permite agrupar de forma lógica todos los componentes de una aplicación o proyecto dentro de un Resource Group, facilitando su administración, control de accesos y visibilidad.

Este enfoque simplifica la gestión del ciclo de vida de los recursos, así como el control de permisos y costos asociados al proyecto, lo que resulta especialmente útil en entornos cloud donde se busca mantener orden y claridad desde las primeras etapas del diseño.

2.3 Arquitectura propuesta

2.3.1 Frontend

Para el componente de frontend se opta por utilizar contenedores desplegados en Azure Container Apps, ya que se trata de un servicio gestionado que simplifica la ejecución de aplicaciones basadas en contenedores. Este servicio permite configurar autoescalado de forma nativa, ajustándose a la carga y al tipo de aplicación que se ejecute.

Adicionalmente, Azure Container Apps facilita la integración con flujos de CI/CD, permitiendo automatizar el despliegue de nuevas versiones de la aplicación de manera sencilla y controlada, lo que resulta adecuado para un escenario de Proof of Concept.

2.3.2 Backend

Para el backend se opta igualmente por el uso de contenedores, desplegados en **Azure Container Apps**, con el fin de mantener coherencia con el frontend y simplificar la gestión de los servicios. Este enfoque permite un mejor aislamiento de la aplicación, así como la posibilidad de escalar el backend de forma independiente según la carga.

El uso de contenedores facilita además la comunicación controlada con los demás componentes del sistema, como la base de datos y el almacenamiento de archivos, permitiendo definir reglas de red y acceso claras dentro de la arquitectura.

2.3.3 Base de datos

Para la base de datos se opta por utilizar un servicio SQL gestionado de Microsoft Azure, como **Azure SQL Database**, ya que facilita la integración con el backend y reduce la complejidad operativa al encargarse de tareas como mantenimiento, parches y disponibilidad.

No obstante, la elección del tipo de base de datos dependerá de las necesidades específicas de la aplicación, ya que en algunos casos podría ser más adecuado utilizar una

base de datos NoSQL, dependiendo del modelo de datos, la escalabilidad requerida y los patrones de acceso.

2.3.4 Almacenamiento de objetos

Para el almacenamiento de archivos se utiliza **Azure Blob Storage**, ya que es un servicio gestionado que permite almacenar y administrar objetos de forma segura y escalable. Este servicio ofrece diferentes niveles y tipos de acceso, lo que facilita controlar quién puede acceder a los archivos y bajo qué condiciones.

Adicionalmente, Azure Blob Storage permite manejar desde archivos pequeños hasta archivos de gran tamaño, adaptándose a distintos escenarios de uso dentro de la aplicación, sin requerir una gestión compleja de la infraestructura subyacente.

2.3.5 Networking y seguridad

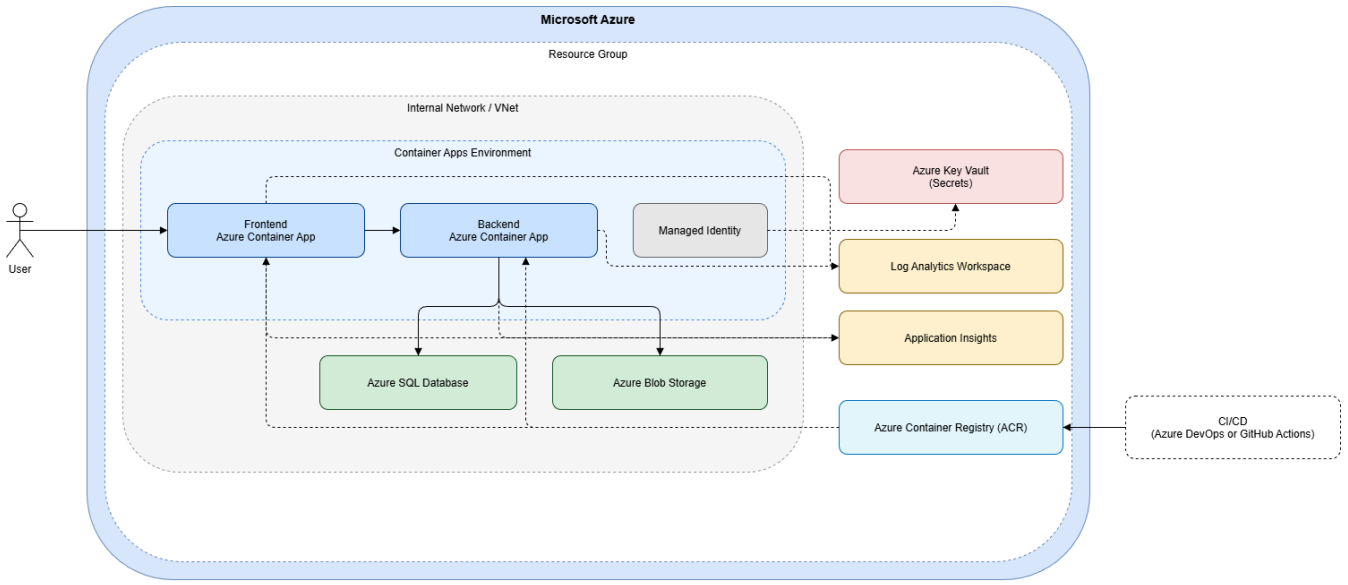
En cuanto a networking y seguridad, se adopta el **principio de mínimo privilegio**, asignando únicamente los permisos necesarios entre los distintos servicios y componentes del sistema. Esto permite reducir riesgos y limitar el impacto ante posibles fallos o accesos no autorizados.

Adicionalmente, se configura una **red interna** para la comunicación entre los componentes de la arquitectura, asegurando que los servicios internos no estén expuestos directamente al exterior y que el acceso se encuentre controlado y segmentado.

2.4 Diagrama de arquitectura

El diagrama presenta la arquitectura propuesta dentro de un Resource Group en Microsoft Azure, donde los componentes principales se encuentran aislados dentro de una red interna. El acceso externo se limita únicamente al frontend, mientras que el backend, la base de datos y el almacenamiento de objetos permanecen protegidos y accesibles solo desde la red interna.

La solución utiliza Azure Container Apps para el despliegue de frontend y backend, apoyándose en Azure Container Registry como repositorio de imágenes y en un flujo de CI/CD para la automatización de despliegues. La gestión de secretos se realiza mediante Azure Key Vault utilizando Managed Identity, y la observabilidad se cubre a través de Log Analytics Workspace y Application Insights.



2.5 Consideraciones operativas

La arquitectura propuesta permite escalar de forma independiente los componentes de frontend y backend mediante Azure Container Apps, adaptándose a la demanda sin requerir gestión directa de infraestructura. Esto facilita la operación del sistema y simplifica los despliegues.

En cuanto a seguridad y operación, el uso de Managed Identity, Azure Key Vault y una red interna permite controlar el acceso entre servicios y reducir la exposición al exterior. Adicionalmente, la integración con Log Analytics Workspace y Application Insights proporciona visibilidad sobre el estado de la aplicación y facilita la detección temprana de incidentes.

3. Infraestructura como Código (IaC)

Para la definición y gestión de la infraestructura se adopta un enfoque de Infraestructura como Código (IaC), ya que permite describir los recursos de forma declarativa, reproducible y versionable. Este enfoque facilita la creación de entornos consistentes, reduce errores manuales y permite auditar cambios en la infraestructura de la misma forma que el código de una aplicación.

Se elige Terraform como herramienta principal debido a su madurez, claridad y capacidad para gestionar infraestructura de manera consistente en distintos entornos. Terraform permite definir los recursos de Azure de forma estructurada, integrarse fácilmente con pipelines de CI/CD y mantener un control claro sobre los cambios aplicados. Adicionalmente, su enfoque independiente del proveedor lo convierte en una herramienta flexible y adecuada para escenarios donde se busca estandarizar la gestión de infraestructura.

4. Conclusiones

A lo largo de este ejercicio se abordaron conceptos fundamentales relacionados con computación en la nube, seguridad, automatización y monitoreo, complementados con el diseño de una arquitectura cloud planteada como un **Proof of Concept (PoC)**. La solución propuesta prioriza simplicidad, buenas prácticas y el uso de servicios gestionados, manteniendo un equilibrio entre funcionalidad y complejidad operativa.

El diseño presentado demuestra cómo una arquitectura bien estructurada, apoyada en principios de seguridad, observabilidad e **Infraestructura como Código**, puede facilitar la operación y evolución de una aplicación en la nube. Este enfoque permite sentar una base sólida que puede ser extendida y adaptada a escenarios productivos según las necesidades del proyecto.