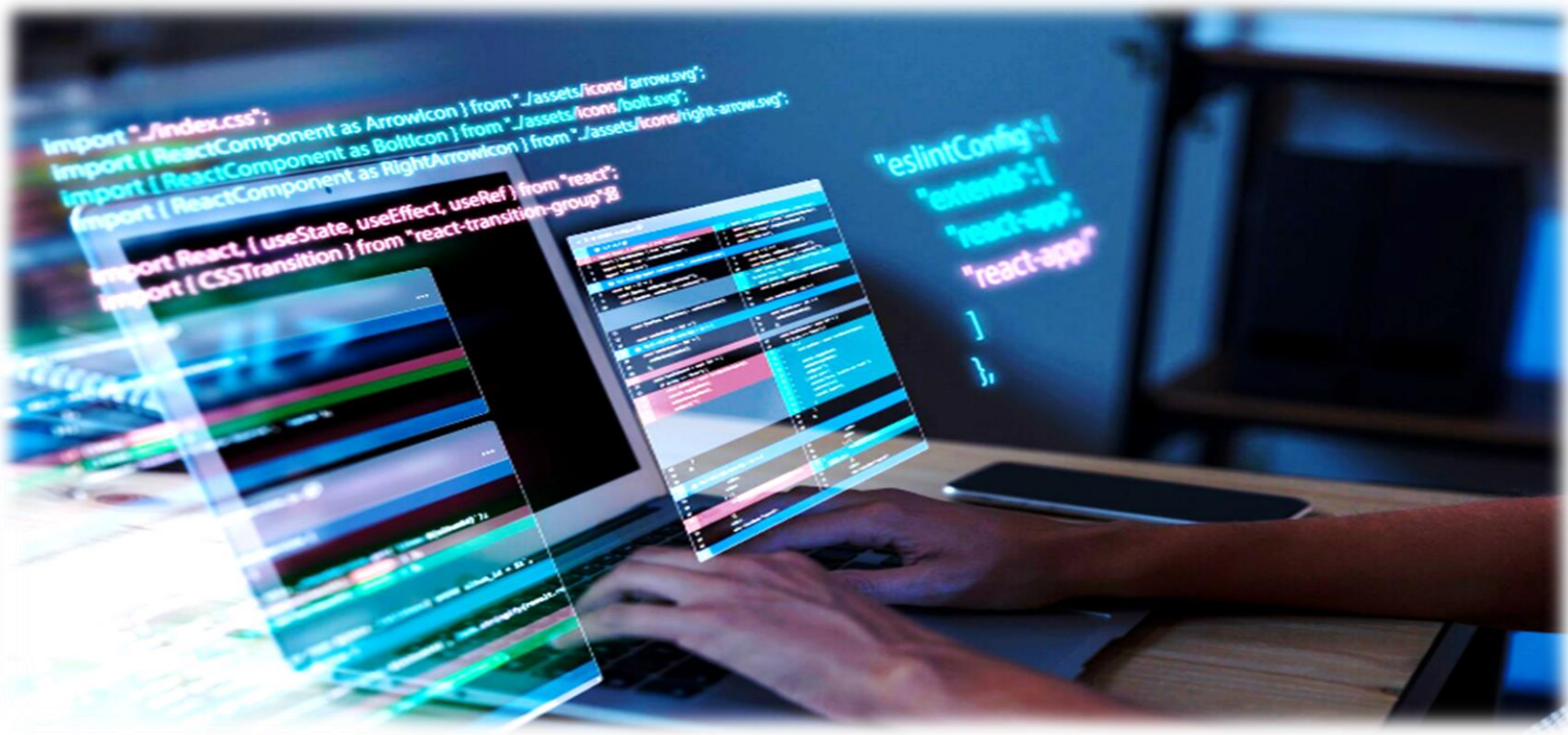

DOCUMENTO TÉCNICO

DOCUMENTO DE RETO TÉCNICO DE INGENIERO CLOUD

DEVSU

Consultor: Jordy Táez

Fecha: 05/08/2025

Versión: 1.0

INTRODUCCIÓN 4
OBJETIVOS 4
AUDIENCIA 4
GLOSARIO DE TÉRMINOS 5
PREGUNTAS TEÓRICAS 6
1. ¿Cuál es la diferencia entre nube pública, privada e híbrida? 6
2. Describa tres prácticas de seguridad en la nube 6
3. ¿Qué es IaC y cuáles son sus principales beneficios? 6
4. ¿Qué métricas considera esenciales para el monitoreo de soluciones en la nube? 7
5. ¿Qué es Docker y cuáles son sus componentes principales? 7
CASO PRÁCTICO 8
PLANTEAMIENTO 8
DISEÑO DE LA SOLUCION 9
DESARROLLO 10
FRONTEND 10
BACKEND 12
BASE DE DATOS 13
ALMACENAMIENTO DE OBJETOS 14
CONCLUSIONES 15
BIBLIOGRAFIA 16

Revisión y aprobación del documento

Historial de cambios:

Versión	Autor	Fecha	Descripción
1.0	Jordy Táez	05/08/2025	Elaboración del documento

Aprobado por:

Nombre	Cargo	Fecha
DEVSU	Líder Técnico	05/08/2025

INTRODUCCIÓN

El presente documento tiene como objetivo abordar conceptos relacionados con la computación en la nube y presentar un diseño de arquitectura para una aplicación nativa en la nube. Todo esto se lo realizará en dos partes, primero a través de una serie de preguntas teóricas y segundo con un caso práctico que debemos documentar.

OBJETIVOS

1. Responder un cuestionario con sustentación teórica las preguntas definidas en la prueba de reclutamiento.
2. Crear un diseño de arquitectura para una aplicación de nube considerando los componentes específicos en el documento.

AUDIENCIA

- Personal de Reclutamiento: pueden ser líder técnico, personal de RRHH o cargos similares al personal de desarrollo.
- Stakeholders: Personal clave relacionados al proyecto que soporte y/o se beneficie de la solución.

CONFIDENCIALIDAD

El material contenido en este documento es propiedad del Emisor y Remitente. Este material incluye información que no debe ser discutida fuera de este proceso de reclutamiento en DEVSU. El documento no puede ser duplicado parcial o completamente para ningún efecto, lo cual significa que es de uso exclusivo del personal designado.

El sello de confidencialidad hace referencia a que esta información no puede ser reproducida ni revelada a terceros.

GLOSARIO DE TÉRMINOS

TÉRMINO	DESCRIPCIÓN
Azure Blob Storage	Servicio de almacenamiento de objetos de Azure, ideal para guardar archivos e imágenes.
Azure Functions	Plataforma serverless de Azure que permite ejecutar código en respuesta a eventos o APIs.
Azure SQL Database	Base de datos relacional PaaS administrada, escalable y segura en la nube de Azure.
Azure AD B2C (Entra)	Servicio de identidad para autenticación de usuarios, permite iniciar sesión con múltiples proveedores.
Serverless	Modelo donde el proveedor gestiona automáticamente la infraestructura y el escalado.
Token JWT	Token estándar para autenticación, portador de identidad y permisos de un usuario.
SAS Token	Token de acceso temporal y limitado para operaciones sobre blobs en Azure Storage.
Key Vault	Servicio de Azure para gestionar secretos y credenciales de manera segura.
PaaS	Platform as a Service: servicios gestionados donde el usuario no administra servidores.
Managed Identity	Identidad gestionada por Azure para acceder de forma segura a otros servicios sin contraseñas.
CDN	Content Delivery Network: red de servidores para distribuir contenido de manera rápida y global.
Redundancia zonal	Replicación de datos en varias zonas de disponibilidad para mayor tolerancia a fallos.
Private Endpoint	Punto de acceso privado para conectar servicios dentro de una red virtual de Azure.

PREGUNTAS TEÓRICAS

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

Nube pública	Administrada por terceros (como Azure, AWS o Google Cloud). Los recursos son compartidos entre múltiples clientes, y se accede a ellos a través de Internet.
Nube privada	Es administrada solo por la organización. Puede estar ubicada en las instalaciones del cliente o alojada por un proveedor externo, pero siempre con infraestructura dedicada.
Nube híbrida	Combina la nube pública y privada permitiendo la comunicación de datos entre ambas según se necesite. Ejemplo: - Una empresa puede mantener sus datos sensibles en una nube privada y usar la nube pública para procesamiento o escalabilidad.

2. Describa tres prácticas de seguridad en la nube

Gestión de identidad y acceso (IAM): Para acceder a los servicios usar autenticación multifactor (MFA) y roles correctamente definidos.

Principio de menor privilegio (PoLP): Otorgar solo los permisos estrictamente necesarios a usuarios y acceso por políticas a los servicios.

Cifrado de datos: Usar protocolos como TLS para datos en tránsito y cifrado AES para datos almacenados.

3. ¿Qué es IaC y cuáles son sus principales beneficios?

IaC (Infrastructure as Code) es una metodología que permite definir y administrar la infraestructura de TI mediante archivos de configuración. Estos archivos están escritos en lenguajes declarativos o de scripting (como YAML, JSON, HCL, o scripts de PowerShell/Bash).

Beneficios:

- Automatización: Se eliminan tareas manuales y repetitivas.
- Replicación: La infraestructura puede ser replicada en múltiples entornos, debido a los archivos de configuración.
- Control de versiones: El código de infraestructura puede versionarse como cualquier software.
- Estandarización: Estandariza configuraciones para minimiza errores.

4. ¿Qué métricas considera esenciales para el monitoreo de soluciones en la nube?

- Disponibilidad: Tiempo en que el servicio está activo y funcionando correctamente.
- Latencia: Tiempo de respuesta entre la petición y la respuesta del sistema.
- Uso de recursos: CPU, memoria, disco y red.
- Errores: Tasas de error HTTP (como 4xx, 5xx) o errores internos.
- Escalabilidad: Monitorear si los recursos aumentan/disminuyen adecuadamente ante cambios en la carga.

Herramientas:

- Azure Monitor: Para ver, analizar y actuar sobre métricas y logs.
- Prometheus con Grafana: Para monitoreo personalizado con visualización atractiva.

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

Docker es una plataforma de software de código abierto que permite desarrollar, empaquetar y ejecutar aplicaciones en contenedores.

- Un contenedor es un espacio ligero, portátil y aislado que incluye todo lo necesario para ejecutar la aplicación como: código, librerías, dependencias y archivos de configuración.

A diferencia de las máquinas virtuales, los contenedores no necesitan un sistema operativo completo, lo que los hace mucho más eficientes y rápidos.

Componentes principales

- Docker Engine: Es el motor que ejecuta y gestionar contenedores en el sistema operativo.
- Dockerfile: Es un archivo de texto donde se describen las instrucciones para crear una imagen. Define, por ejemplo, el sistema operativo base, los archivos a copiar, los comandos a ejecutar, etc.
- Imágenes (Images): Son plantillas que contienen todo lo necesario para ejecutar una aplicación (código, librerías, configuración). Se crean a partir de archivos llamados Dockerfile.
- Contenedores (Containers): Son instancias en ejecución de las imágenes. Cada contenedor es un entorno aislado y ligero.
- Docker Hub:
  Es una plataforma (en la nube) donde se almacenan y comparten imágenes Docker, públicas o privadas. Puedes descargar imágenes para usarlas o subir tus propias imágenes.
- Docker Compose: Herramienta que permite ejecutar aplicaciones multicontenedor usando un archivo docker-compose.yml, facilitando la orquestación de varios servicios (por ejemplo: una base de datos y una aplicación web juntas).

CASO PRÁCTICO

PLANTEAMIENTO

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.

Diseño:

- Seleccione un proveedor de servicios de nube (AWS, Azure o GCP) y sustente su selección.
- Diseñe una arquitectura de nube. Incluya diagramas que representen la arquitectura y justifique sus decisiones de diseño (Utilice https://app.diagrams.net/).

DISEÑO DE LA SOLUCION

Para esta solución todos los componentes son servicios PaaS (Platform as a Service) o serverless de Azure, lo que permite alta escalabilidad y reduce costos de administración. A continuación, se detallan cada uno de los componentes principales, la justificación de los servicios de Azure seleccionados y las prácticas recomendadas en cuanto a seguridad, disponibilidad y mantenimiento.

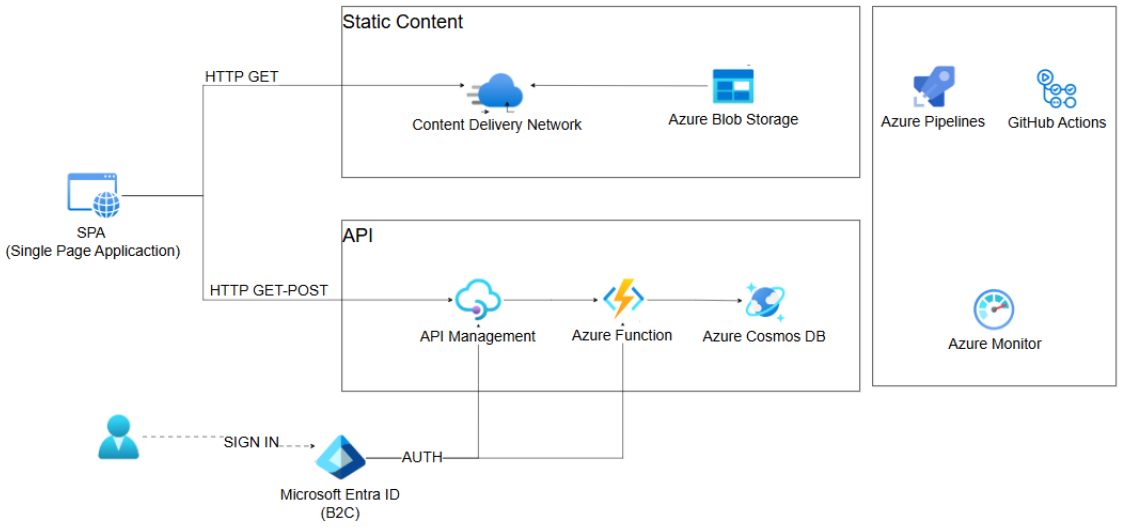

Diagrama general de la solución

Se muestra un frontend web estático hospedado en Azure Blob Storage (mediante la funcionalidad de sitio web estático) y distribuido con Azure Content Delivery Network (CDN). Los usuarios acceden al frontend desde sus navegadores (cliente), y este realiza llamadas a un backend implementado con Azure Functions (funciones serverless desencadenadas por solicitudes HTTP).

Opcionalmente, Azure API Management puede actuar como puerta de enlace (API Gateway) para unificar y securizar las APIs expuestas. El backend interactúa con una base de datos administrada en Azure (en el diagrama de referencia se ilustra Azure Cosmos DB como ejemplo, aunque podría usarse Azure SQL Database según las necesidades).

Las imágenes y archivos estáticos de la aplicación se almacenan en Azure Blob Storage (almacenamiento de objetos). Adicionalmente, servicios como Azure Monitor brindan monitoreo centralizado, mientras que herramientas de integración continua/entrega continua (CI/CD) como Azure Pipelines o GitHub Actions facilitan el despliegue y mantenimiento.

DESARROLLO

FRONTEND

El frontend de la aplicación es una Single Page Application (SPA) desarrollada, por ejemplo, en Angular, React o cualquier framework moderno que genere archivos estáticos HTML, CSS y JavaScript.

Para alojar este frontend en Azure de forma escalable y económica, se recomienda utilizar Azure Static Web Apps (ideal para proyectos Angular, React o Vue) o, alternativamente, Azure Blob Storage con static website hosting. Ambos servicios permiten desplegar y servir aplicaciones Angular directamente, sin necesidad de servidores web dedicados, ya que solo requieren archivos estáticos generados tras el proceso de build del framework.

Todo el contenido dinámico se obtendrá desde el navegador mediante llamadas a las APIs del backend, optimizando así el rendimiento y evitando sobrecargar el servidor.

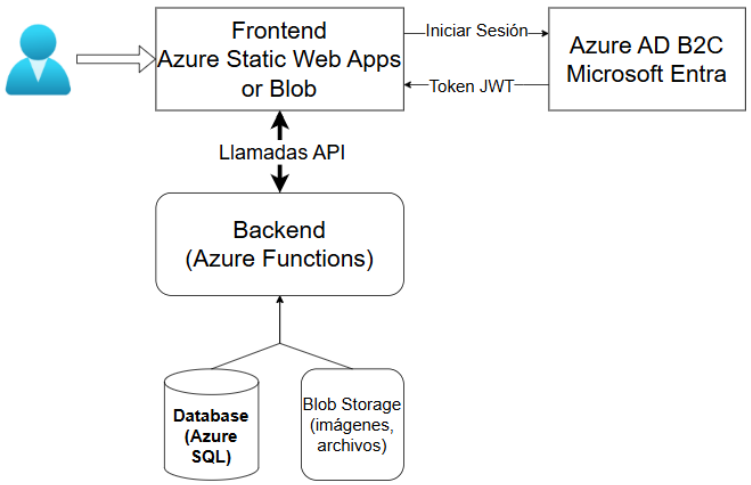

Diagrama Frontend - Flujo de Autenticación y Comunicación Frontend

El siguiente diagrama muestra cómo interactúan los diferentes componentes de la arquitectura cloud para autenticar usuarios y manejar la comunicación entre el frontend y los servicios backend en Azure.

1. Acceso a la Aplicación:
   El usuario ingresa a la aplicación web a través de su navegador. El frontend está alojado en Azure Static Web Apps o como sitio web estático en Azure Blob Storage, lo que garantiza alta disponibilidad y bajo costo.
2. Inicio de Sesión:
   Desde la página principal, si el usuario no está autenticado, la aplicación ofrece la opción de iniciar sesión. Al seleccionarla, el usuario es redirigido al portal de autenticación de Azure AD B2C (Microsoft Entra ID), donde puede ingresar sus credenciales de manera segura.
3. Autenticación y Token:
   Una vez autenticado, Azure AD B2C devuelve al frontend un token JWT (JSON Web Token), que representa la identidad y permisos del usuario. Cuando el usuario intenta acceder a cualquier parte protegida de la aplicación, el frontend (por ejemplo, en el enrutador o en un middleware/interceptor de la SPA) verifica si existe un token JWT válido en el almacenamiento local/sessionStorage/cookies.
   - Si NO existe: Redirige automáticamente al flujo de inicio de sesión de Azure AD B2C.
   - Si existe pero está expirado: Puede intentar refrescarlo (si se usa refresh token), o volver a redirigir al login.

   Esto mantiene la experiencia de usuario segura y fluida.

4. Llamadas API Seguras:

Con el token JWT, el frontend realiza llamadas a las APIs del backend (implementadas con Azure Functions), incluyendo el token en cada solicitud para que el backend pueda validar la identidad y autorización del usuario.

5. Acceso a Datos y Archivos:
   El backend puede consultar la base de datos (Azure SQL Database) para obtener o modificar información, y acceder a Azure Blob Storage para gestionar imágenes u otros archivos que pertenezcan al usuario o a la aplicación.
6. Respuesta al Usuario:
   Los resultados y datos solicitados se devuelven al frontend, que los muestra al usuario en la interfaz web.

BACKEND

El backend de la solución es responsable de procesar la lógica de negocio y exponer las APIs que consume el frontend. En esta arquitectura, el backend está implementado con Azure Functions, una plataforma serverless de Azure que permite ejecutar código en respuesta a solicitudes HTTP, eventos o programaciones, sin necesidad de administrar servidores.

Configuración y Validación de JWT en el Backend:

Habilitar la autenticación integrada en Azure Functions (EasyAuth) para Azure AD B2C. De este modo, Azure verifica automáticamente los tokens antes de ejecutar el código y proporciona la información del usuario autenticado.

- Alternativa:
  Realizar la validación manual del JWT en el código usando una librería estándar según el lenguaje (por ejemplo, Microsoft.IdentityModel.Tokens en C#). Esto implica configurar el emisor, el público (audience) y las claves públicas de B2C, y responder con error 401 si el token no es válido.

Acceso a la Base de Datos y Manejo de la Cadena de Conexión:

Para mantener la seguridad y flexibilidad, la cadena de conexión a la base de datos no se almacena en el código fuente. La cadena de conexión se pondrá como una variable de entorno (Application Setting) en Azure Functions.

- Alternativa:
  Utilizar Azure Key Vault para almacenar la cadena de conexión como un secreto seguro y referenciarla desde la configuración de Azure Functions.

Estructura de Archivos en Azure Functions

Para mantener un backend organizado, escalable y fácil de mantener, es importante definir una estructura de archivos clara dentro del proyecto de Azure Functions. A continuación, se describe una estructura sugerida para proyectos medianos o grandes:

```
/MiProyectoAzureFunctions/
│
├── host.json                 # Configuración global del Function App
├── local.settings.json       # Configuración solo para desarrollo local
│
├── MiProyectoAzureFunctions.csproj   # Archivo de proyecto (C#)
│
├── /FunctionFolder1/
│   └── MiPrimeraFunction.cs   # Una clase por función, puede incluir varios métodos
│
├── /FunctionFolder2/
│   └── OtraFunction.cs
│
├── /Models/
│   └── User.cs                # Modelos de datos (clases POCO)
│
├── /Services/
│   └── UserService.cs         # Lógica de negocio reutilizable, servicios, helpers
│
├── /Utils/                    # (Opcional) utilidades y clases compartidas
│
└── README.md                  # Documentación del backend
```

Estructura general de los archivos en el Backend

BASE DE DATOS

En la arquitectura propuesta, la persistencia de la información se gestiona utilizando Azure SQL Database, una base de datos relacional en la nube, administrada y altamente disponible, ideal para aplicaciones modernas que requieren transacciones confiables y consultas SQL estándar.

Selección del Servicio:

Se utiliza Azure SQL Database en modo serverless para contar con una base de datos relacional escalable y administrada, que ajusta recursos automáticamente y reduce costos al cobrar solo por uso real.

Azure SQL Database escala recursos según demanda y garantiza alta disponibilidad con opciones de redundancia local, zonal y geográfica. Ofrece copias de seguridad automáticas y recuperación ante desastres, protegiendo la información incluso frente a fallos regionales.

Estructura y Organización:

La información principal (usuarios, roles, transacciones, etc.) se almacena en tablas relacionales. Los archivos y las imágenes se guardan externamente en Blob Storage y solo se registra su enlace o referencia en la base de datos.

Nomenclatura Recomendada para Objetos de Base de Datos:

- Tablas:
  Prefijo tbl_ seguido del nombre descriptivo y en singular.
  Ejemplo: tbl_Usuario, tbl_Transaccion, tbl_Producto
- Vistas:
  Prefijo vta_ seguido de un nombre descriptivo que indique claramente el propósito de la vista.
  Ejemplo: vta_UsuariosActivos, vta_VentasMensuales
- Índices:
  Prefijo idx_ seguido del nombre de la tabla y columna(s) indexada(s).
  Ejemplo: idx_Usuario_Email, idx_Transaccion_Fecha
- Triggers:
  Prefijo trg_ seguido del nombre de la tabla y el evento que dispara el trigger (Insert, Update, Delete).
  Ejemplo: trg_Usuario_Insert, trg_Transaccion_Update

ALMACENAMIENTO DE OBJETOS

Para almacenar y servir archivos (imágenes, documentos, etc.), se utiliza Azure Blob Storage, el servicio de almacenamiento de objetos de Azure, ideal por su escalabilidad, bajo costo y alta durabilidad.

Almacenamiento y estructura de Paths:

En Azure Blob Storage, los archivos (blobs) se organizan en contenedores y cada archivo tiene un path único que puede simular una estructura de carpetas usando / en el nombre del blob, aunque técnicamente es un espacio plano.

Ejemplo:
https://<account>.blob.core.windows.net/<contenedor>/usuarios/123/foto.jpg

- <account>: Nombre de la cuenta de almacenamiento.
- <contenedor>: Nombre lógico del contenedor (ej. usuarios, imagenes, documentos).

- usuarios/123/foto.jpg: Ruta lógica del archivo; aquí se usa el ID de usuario para organizar archivos individuales, facilitando búsquedas y mantenimientos.

Uso de la API REST de Azure Blob Storage:

Azure Blob Storage ofrece una API REST completa para manipular archivos desde cualquier lenguaje o herramienta que soporte HTTP. Las operaciones principales incluyen:

- Subir archivos (PUT Blob):
- Leer/Descargar archivos (GET Blob):
- Listar archivos (GET List Blobs)
- Eliminar archivos (DELETE Blob)

Autenticación:

El acceso a la API REST puede realizarse mediante:

- SAS Token: Token temporal que otorga permisos específicos sobre uno o varios blobs.
- Shared Key: Clave maestra de la cuenta (no recomendado para exponer en clientes).
- Azure AD: Acceso autenticado vía identidades administradas (preferible desde backend).

CONCLUSIONES

Dado que no se especifica el alcance funcional exacto de la aplicación, se ha definido una arquitectura base en Azure orientada a optimizar costos y garantizar escalabilidad, utilizando componentes serverless y PaaS para reducir la administración operativa.

Se seleccionó Azure SQL Database en modo serverless para lograr flexibilidad y ahorro en entornos de carga variable, y Azure Blob Storage para el almacenamiento eficiente y seguro de archivos y documentos.

La autenticación y seguridad se gestionan con Azure AD B2C, permitiendo integraciones con múltiples proveedores y facilitando la gestión de usuarios, recordemos que los primeros 50000 usuario no tienen costo lo cual es un gran beneficio.

Cada decisión técnica prioriza la facilidad de despliegue, la administración mínima y el cumplimiento de buenas prácticas de seguridad y mantenimiento, asegurando que la solución pueda evolucionar según los requerimientos futuros sin reingeniería significativa.

BIBLIOGRAFIA

Las fuentes de información a continuación complementan la documentación e investigación realizada.

Azure Blob Storage – Almacenamiento de objetos y archivos:
https://learn.microsoft.com/en-us/azure/storage/blobs/

Azure SQL Database – Base de datos relacional administrada:
https://learn.microsoft.com/en-us/azure/azure-sql/database/

Azure Functions – Ejecución de código serverless:
https://learn.microsoft.com/en-us/azure/azure-functions/

Azure AD B2C – Autenticación y control de acceso:
https://learn.microsoft.com/en-us/azure/active-directory-b2c/

Azure AD B2C – MFA (Autenticación multifactor):
https://learn.microsoft.com/en-us/azure/active-directory-b2c/multi-factor-authentication

Azure Key Vault – Gestión segura de secretos:
https://learn.microsoft.com/en-us/azure/key-vault/general/

Azure Managed Identity – Acceso seguro a recursos:
https://learn.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/overview

Azure CDN – Distribución global de contenido:
https://learn.microsoft.com/en-us/azure/cdn/

Azure Private Endpoint – Conexión privada y segura:
https://learn.microsoft.com/en-us/azure/private-link/private-endpoint-overview

Azure Architecture Center – Ejemplo de arquitectura serverless:
https://learn.microsoft.com/en-us/azure/architecture/example-scenario/serverless/web-app