Ejercicio Práctico - Ingeniero Cloud

Postulante: Maya Mafla Bryan Vicente

CI: 1003298799

PREGUNTAS TEÓRICAS:

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

   Nube pública: Infraestructura ofrecida por un proveedor externo como AWS, Azure o GCP y compartida entre múltiples clientes. Ofrece escalabilidad, disponibilidad global y modelo de pago por uso.

   Nube privada: Infraestructura dedicada exclusivamente a una organización, ya sea en sus propios centros de datos o en servicios dedicados. Ofrece mayor control, seguridad y cumplimiento.

   Nube híbrida: Combina nube pública y privada, permitiendo mover cargas de trabajo entre ambas según necesidades de negocio, costos o cumplimiento. Ideal para escenarios de migración gradual o burst computing.

2. **Describa tres prácticas de seguridad en la nube.**

   Gestión de identidades y accesos (IAM): Aplicar el principio de menor privilegio y usar autenticación multifactor para proteger el acceso a los recursos.

   Cifrado de datos: Usar cifrado en tránsito (TLS) y en reposo (AES-256) para proteger la confidencialidad e integridad de la información.

   Monitoreo y auditoría continua: Utilizar herramientas como AWS CloudTrail o Azure Monitor para registrar actividades, detectar accesos no autorizados y responder a incidentes.

3. **¿Qué es IaC y cuáles son sus principales beneficios? Mencione 2 herramientas de IaC y sus principales características.**
   Infrastructure as Code (IaC) es un enfoque que permite definir, suministrar y gestionar infraestructura mediante archivos de configuración escritos en código, en lugar de realizar configuraciones manuales. Esto permite automatizar y estandarizar los entornos de TI.

Herramientas	Caracteristicas
Terraform	- Compatible con múltiples nubes (AWS, Azure, GCP, entre otras). - Utiliza un lenguaje declarativo (HCL). - Permite previsualizar los cambios antes de aplicarlos.
AWS CloudFormation	- Integración nativa con los servicios de AWS. - Define la infraestructura con archivos en YAML o JSON. - Permite automatizar despliegues con soporte para rollback.

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

    - Uptime del servicio: Asegura cumplimiento de SLA.
    - Latencia y tiempos de respuesta: Identifican posibles cuellos de botella en red o backend.
    - Consumo de recursos (CPU, RAM, disco, red): Permite anticiparse a sobrecargas o mal dimensionamiento.
    - Errores 4xx/5xx: Ayudan a detectar fallos de aplicación o configuraciones incorrectas.
    - RPS (requests por segundo): Útil para evaluar comportamiento bajo carga.
    - Eventos de seguridad: Intentos de acceso no autorizado, fallos de login, cambios críticos.
    - Logs y auditoría: Rastrean acciones administrativas y son clave para análisis post incidente y cumplimiento.

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

    Docker es una plataforma de contenedores que permite desarrollar, empaquetar y ejecutar aplicaciones en entornos aislados y portables.

    Considero los siguientes como componentes principales:

    - Docker Engine: Servicio que permite crear y gestionar contenedores.
    - Dockerfile: Archivo que contiene instrucciones para construir una imagen.

- Imagen: Plantilla inmutable que contiene el sistema de archivos y dependencias.
- Contenedor: Instancia en ejecución de una imagen.
- Docker Hub: Registro público para almacenar y compartir imágenes.

CASO PRÁCTICO

Para este caso práctico se propone la migración de un sistema de directorio institucional a la nube, utilizando Amazon Web Services (AWS) como proveedor. Se trata de una aplicación web en la que los usuarios pueden consultar información organizada de personas, unidades, roles, extensiones o documentos institucionales. El objetivo es modernizar la arquitectura actual, facilitar el acceso, mejorar la disponibilidad y reducir la carga operativa sobre el equipo de TI.

La solución se basa en una arquitectura contenedorizada y modular, utilizando servicios gestionados que aseguran escalabilidad, seguridad y continuidad operativa. AWS fue seleccionado por su madurez, ecosistema de servicios robusto, soporte a microservicios, y su modelo flexible de consumo.

1. **Diseño de la arquitectura**

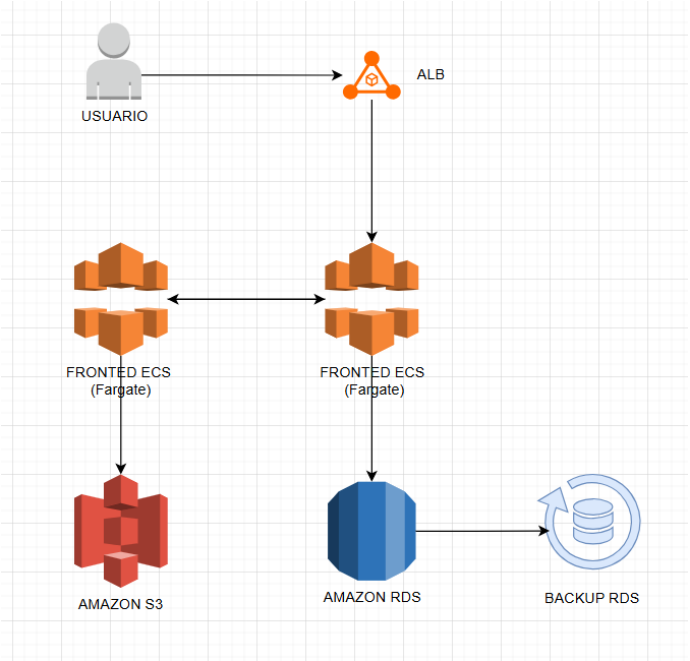
   El usuario accede a la aplicación desde el navegador. Todo el trafico entra por el Application Load Balancer (ALB), que se encarga de redirigir las peticiones hacia los servicios que esten activos. Esto ayuda a distribuir bien la carga y mantener la app siempre disponible.

   El Frontend está desplegado en contenedores usando ECS con Fargate, lo que permite no preocuparse por servidores ni infraestructura. Aquí se encuentra la parte visual de la app, la que usa el cliente final. Este servicio también accede a Amazon S3, donde se guardan los archivos estaticos como imagenes o documentos

   El Backend, tambien en Fargate, contiene la lógica del sistema. Se conecta con el frontend para recibir solicitudes y con la base de datos para guardar o consultar informacion. Para eso se usa Amazon RDS, que es una base de datos gestionada que facilita respaldos y alta disponibilidad sin tener que administrarla directamente.

   Por último, la base de datos tiene configurado un sistema de respaldo automatico (Backup RDS) para poder recuperar informacion en caso de errores o perdida de datos.

Toda la infraestructura es monitoreada con CloudWatch, y los accesos estan controlados con IAM Ademas, se usan certificados SSL gestionados para asegurar las comunicaciones por HTTPS todo esto con enfoque a las buenas practicas.



1.1. Frontend

El frontend de la aplicación representa la interfaz con la que interactúa el usuario. En este caso, se trata de un directorio web institucional, desarrollado como una aplicación de página única (SPA) utilizando React. Esta estructura permite una navegación fluida, sin recarga completa de la página, y una experiencia más ágil para el usuario final.

La estructura del proyecto se mantiene sencilla y funcional, enfocada en contener solo lo necesario para la construcción y despliegue en producción:

```
directorio-frontend/
├── public/
│   └── index.html
├── src/
│   ├── App.js
│   └── index.js
├── package.json
├── Dockerfile
├── .dockerignore
```

Descripción de los archivos y su interacción:

- **public/index.html** es la plantilla HTML base del sitio. Contiene el contenedor donde React renderiza toda la interfaz visual.
- **src/index.js** es el punto de entrada en JavaScript. Se encarga de montar el componente App en el HTML.
- **src/App.js** contiene la estructura principal de la aplicación: layout general, navegación, rutas internas y lógica de presentación.
- **package.json** define las dependencias, scripts y configuración necesaria para construir y ejecutar la aplicación.
- **Dockerfile** permite empaquetar la aplicación como una imagen Docker, compilando el código con Node.js y sirviéndolo con Nginx.
- **.dockerignore** excluye archivos innecesarios del proceso de construcción de la imagen, optimizando espacio y tiempo.

Una vez construida la imagen Docker, esta puede desplegarse en AWS ECS Fargate. El servicio se expone públicamente mediante un Application Load Balancer que enruta el tráfico hacia el contenedor y aplica certificados SSL (ACM) para mantener las comunicaciones seguras. El frontend también interactúa con Amazon S3 para cargar archivos estáticos (como imágenes, documentos o recursos públicos) y con el backend para consultar información en tiempo real.

1.2. Backend

El backend de la aplicación contiene la lógica de negocio y es responsable de procesar las solicitudes provenientes del frontend. En este caso, se implementa como un servicio REST que expone endpoints para consultar y administrar la información del directorio (por ejemplo, datos de personas, departamentos o servicios).
El backend está desarrollado con Node.js y Express, una combinación ligera y adecuada para APIs. La aplicación está contenida en un proyecto que puede ejecutarse localmente o empaquetarse como contenedor Docker para su despliegue en producción, específicamente en AWS ECS Fargate.

La estructura mínima y funcional del proyecto es la siguiente:

```
directorio-backend/
├── src/
│   ├── routes/
│   │   └── personas.js
│   ├── controllers/
│   │   └── personasController.js
│   ├── db/
│   │   └── connection.js
│   └── index.js
├── package.json
├── Dockerfile
├── .dockerignore
```

Descripción de los archivos y su interacción:

- **src/index.js** es el punto de arranque del backend. Aquí se configura el servidor Express y se integran todas las rutas disponibles en la aplicación.
- **src/routes/personas.js** define las rutas de acceso a los recursos del directorio, especificando las URL y los métodos HTTP permitidos.
- **src/controllers/personasController.js** contiene la lógica que maneja las solicitudes del cliente: se conecta con la base de datos, procesa los datos y envía la respuesta.
- **src/db/connection.js** gestiona la conexión a la base de datos. Define los parámetros de conexión y se encarga de exponerlos para su uso en los controladores.
- **package.json** centraliza la configuración del proyecto: define qué bibliotecas se instalan, cómo se ejecuta el servidor y qué versiones de dependencias se usan.
- **Dockerfile** contiene las instrucciones para empaquetar el backend como una imagen, desde la instalación de Node.js hasta la ejecución del servidor.
- **.dockerignore** sirve para evitar que archivos que no se necesitan en producción —como logs o dependencias ya instaladas— se copien dentro del contenedor.

Una vez construida la imagen, se despliega en un servicio ECS Fargate, accesible internamente por el frontend a través del Application Load Balancer. La base de datos utilizada es Amazon RDS, a la cual el backend accede desde subredes privadas para mantener la seguridad. Toda la lógica sensible queda encapsulada en este servicio, separado del cliente, lo que permite mayor control y mantenimiento centralizado de la aplicación.

1.3. Base de Datos

La base de datos de la aplicación es un componente clave para almacenar la información del directorio, como datos personales, cargos, departamentos y cualquier otro registro necesario para la operación del sistema.

Para esta solución se utiliza Amazon RDS (Relational Database Service) con el motor PostgreSQL, debido a su estabilidad, escalabilidad y soporte nativo en AWS. Al ser un servicio gestionado, RDS se encarga de tareas operativas como actualizaciones, respaldos automáticos, monitoreo, replicación y recuperación ante fallos, lo cual reduce la carga de administración y mejora la disponibilidad.

Estructura

La estructura lógica del esquema puede incluir tablas como:

- **personas:** contiene nombres, correos, extensiones, unidad a la que pertenecen, etc.
- **departamentos:** estructura organizacional y relaciones jerárquicas.
- **roles o tipos:** clasificaciones específicas de usuarios o entradas.

La conexión con la base de datos se establece desde el backend, utilizando controladores como pg para PostgreSQL. Esta conexión se define en un archivo específico (por ejemplo, src/db/connection.js) donde se especifican credenciales, nombre de la base de datos, puerto y host.

Seguridad y acceso

La base de datos se despliega dentro de una subred privada en una VPC, lo que significa que no tiene exposición pública directa. Solo el backend puede comunicarse con RDS a través de reglas de seguridad definidas en los grupos de seguridad de AWS.

Respaldo y recuperación

Amazon RDS permite configurar backups automáticos diarios, que se almacenan durante un período definido por ejemplo, 7 o 14 días. También es posible generar snapshots manuales antes de cada despliegue importante. En caso de falla o corrupción, RDS permite restaurar la base de datos en un nuevo punto en el tiempo, minimizando la pérdida de datos.

1.4. Almacenamiento

El sistema requiere almacenar y servir archivos estáticos asociados al directorio, como fotografías de perfil, documentos institucionales o recursos vinculados a las entradas del sistema. Para esto se utiliza Amazon S3 permite subir, almacenar y acceder a archivos desde cualquier parte de la aplicación. En este caso, el frontend accede a S3 directamente para mostrar imágenes públicas (como fotos de personas) o para permitir descargas de documentos. Por otro lado, el backend puede gestionar la carga o eliminación de archivos desde rutas seguras, utilizando la SDK de AWS para Node.js o desde API REST autenticadas.

Estructura sugerida de almacenamiento
Para mantener el orden y facilitar el acceso, se recomienda estructurar el bucket de S3 en carpetas lógicas:

```
directorio-bucket/
├── personas/
│   └── 123_foto.jpg
├── documentos/
│   └── reglamento.pdf
└── otros/
```

Descripción de los archivos y su interacción:

- **personas/:** contiene las imágenes de perfil, nombradas con algún identificador único.
- **documentos/:** archivos generales o asociados a unidades o procesos.
- **otros/:** cualquier contenido adicional que se requiera más adelante.

Seguridad y control de acceso

Por defecto, el bucket puede configurarse como privado, y el acceso a los objetos se puede gestionar de dos formas:
Acceso público limitado: Solo para ciertos objetos como fotos de perfil visibles para todos los usuarios.
URLs firmadas: El backend genera una URL temporal que permite acceder al archivo por un tiempo limitado, sin exponer el bucket directamente.
Además, es posible aplicar políticas de bucket, usar roles IAM específicos para los servicios ECS que acceden a S3, y registrar todas las operaciones en CloudTrail para auditoría.

ENLACES: