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

Nube Pública: La infraestructura es alojada y mantenida por un proveedor externo como AWS, Azure, Google Cloud o Huawei, etc.

Nube Privada: Puede estar en las propias instalaciones de la empresa o ser administrada por un tercero, por ejemplo un centro de datos propio de un banco.

Nube híbrida: Combina nubes públicas y privadas, permitiendo tener flujos y cargas de trabajo entre ambas según las necesidades de la empresa.

2. Describa tres prácticas de seguridad en la nube.

- Control de acceso usando roles de usuario, usando por ejemplo IAM.
- Cifrado de datos, usando por ejemplo KMS.
- Monitoreo de infraestructura, usando por ejemplo CloudTrail, Cloudwatch, WAF.

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

IAC, es el manejo de infraestructura con código en lugar de hacerlo manualmente.

Permite versionamiento, automatización y escalabilidad.

- **Terraform**
  - Multicloud
  - Control de estado
  - Declarativo
- **CloudFormation**
  - Integración nativa con AWS
  - Declaración usando JSON o YAML

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

- CPU/Memoria
- Apdex Score
- Throughput
- Latencia
- Uptime
- Errores por segundo

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

Es una plataforma que permite crear, desplegar y ejecutar aplicaciones dentro de contenedores, aislando el software y sus dependencias del hardware.

Componentes:

- **daemon** (dockerd): motor de administración de objetos de docker: volúmenes, imágenes, redes, etc.
- **docker-cli:** línea de comandos para interactuar con el daemon.

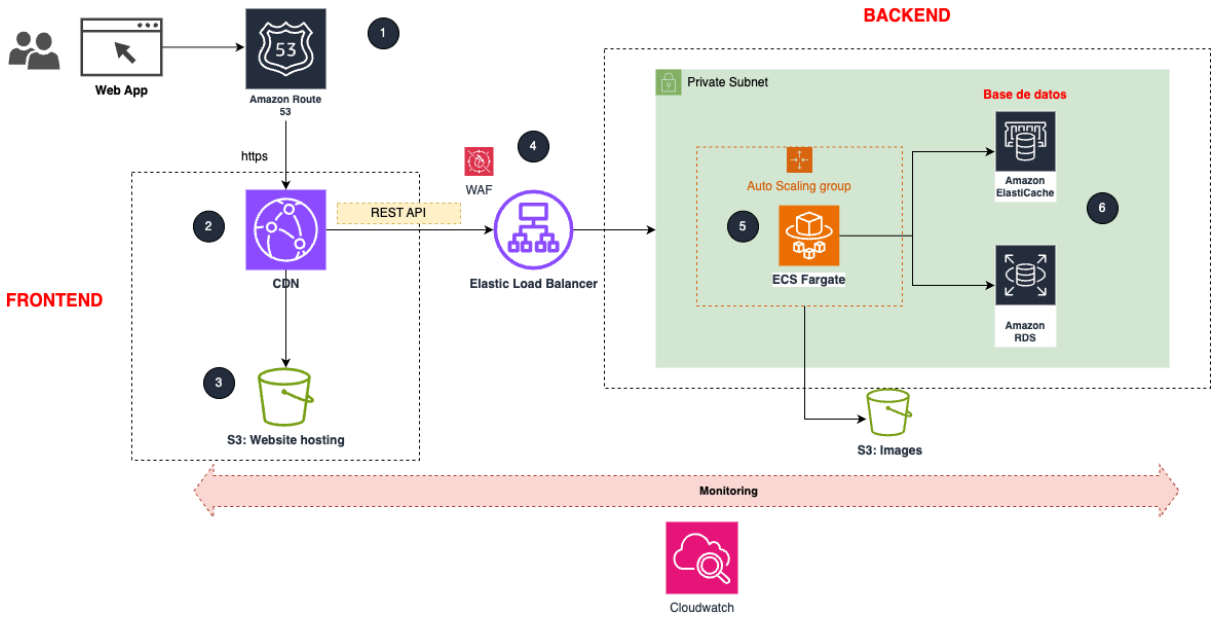
- **docker image:** es un template que contiene el sistema de archivos para ejecutar la aplicación.
- **dockerfile:** es un archivo de instrucciones para construir la imagen.
- **container**: es la unidad principal en docker, es una instancia de ejecución de una imagen. Es efímero.
- **volume**: sistema de almacenamiento para persistencia de datos.
- **network**: sistema de gestión de comunicación entre contenedores con el exterior.
- **docker compose:** herramienta para definir y ejecutar aplicaciones multi contenedor.

6. Caso práctico

Diseño de Arquitectura - Aplicación WEB en la nube

Contexto

El siguiente diagrama detalla la arquitectura de una aplicación Web nativa en AWS. Se toman en consideración las capas principales solicitadas en el ejercicio y puede ser extendible dependiendo de la necesidad.



Decisión
Se eligió AWS como proveedor de nube ya que es la plataforma más utilizada a nivel mundial según datos de Gartner superando a GCP y AZure. Además es intuitivo, fácil de administrar y cuenta con un SLA de 99.99% lo que garantiza un nivel de calidad y confianza robusta para sus clientes.

A continuación se detalla cada componente elegido por capa y se sustenta la decisión técnica de cada uno. Cada componente es nativo de AWS por lo que no es

Frontend

- **Route53:** El dominio y certificados se manejan en Route53 y ACM. El registro en Route53 tipo será tipo CNAME y estará atachado al CDN.
- **CDN:** Se usa CDN para asegurarse que el contenido estático sea entregado a los usuarios con la menor latencia posible. El CDN debe estar atachado al bucket S3 que servirá el contenido estático.
  Esto es opcional y lo consideré en caso de ser una aplicación web de alto tráfico.
- **S3 Web Hosting:** Este bucket almacenará el contenido estático como el index.html y el bundle. El bucket debe estar configurado para Static Web Hosting

Security

- **WAF:** Este es un firewall que protegerá el load balancer del backend de intrusiones y posibles ataques.

Backend

- **ELB:** Se usa un Application Load balancer para balancear la carga del backend. También se puede usar un api gateway si se necesita una arquitectura serverless.
- **ECS Fargate:** Este servicio permite levantar instancias de contenedor con 0 administración manual. Además cuenta con servicios nativos de escalamiento, manejo de variables y despliegue sin downtime usando múltiples técnicas como “Rolling update”. Podemos empezar con una tarea de 0.25CPU y 256 de Memoria lo cual es cost-efficient y puede mantener nuestra aplicación encendida 24/7.
- **RDS:** La base de datos será relacional usando una instancia micro que puede ser escalada en base a las necesidades de la empresa. Se podría usar una opción Serverless pero se debe hacer un análisis costo beneficio previo.
- **Elasticache:** Este componente es opcional pero me parece fundamental en aplicaciones de alto tráfico para evitar sobrecargar la base de datos y cachear respuestas de endpoints con alto throughput.
- S3 Object Storage: Este servicio permitirá almacenar blob object como imágenes, metadata, etc. También nos sirve para persistir los logs del load balancer y analizarlos en caso de que existan errores a nivel del balanceador de tráfico.

Monitoring

- Cloudwatch: Este servicio permitirá monitorear cada componente de la arquitectura de manera separada como el Load Balancer, ECS, RDS, Caché, etc. Además podemos monitorizar los logs a nivel de aplicación. Si se necesita un monitoreo más preciso de la aplicación se pueden usar otras soluciones en la nube como Newrelic o Prometheus.

Networking

- **Backend**: ECS en subnet privada, y en al menos 2 zonas de disponibilidad. Si se requiere acceso a internet se puede poner una subnet pública o un NAT Gateway con subnet privada
- **Load Balancer:** Debe estar en al menos 2 subnets públicas en diferentes zonas de disponibilidad..
- **Base de datos**: En la misma subnet privada que el backend. RDS debe estar en la misma VPC del backend.
- **CDN:** SIN VPC ya que es un servicio global.

Deployment

- **Terraform:** La definición de todos los componentes de la arquitectura se hará usando terraform
- **Github Actions:** Se configurarán actions para el frontend y el backend con la finalidad de cumplir con los estándares del CI/CD.