Definiciones teóricas prueba técnica, ejemplos y lógica dada desde un ambiente de desarrollador.
Nombre: Juan David Meneses Galeano.

Amazon RDS vs Amazon DynamoDB

Comparación General

Característica	Amazon RDS	Amazon DynamoDB
Tipo de base de datos	Relacional (SQL)	NoSQL (clave-valor, documento)
Escalabilidad	Vertical y parcialmente horizontal	Horizontal automático y sin servidor
Latencia	Milisegundos	Microsegundos / milisegundos
Administración	Semi-administrada (EC2 detrás)	Completamente serverless
Transacciones	Soporte ACID completo	Soporte limitado a nivel de ítem
Consultas complejas	JOIN, subconsultas, funciones SQL	Clave-partición, rangos, filtros simples
Seguridad	IAM, KMS, cifrado, VPC	IAM, KMS, VPC

Casos de Uso

Amazon RDS

Usar cuando:

- Se requieren relaciones entre tablas y transacciones ACID fuertes
- Se ejecutan consultas SQL complejas
- Se trabaja con frameworks ORM (Django, .NET, Laravel, etc.)

Ejemplos:

- Sistemas financieros o bancarios
- ERP o CRM con estructuras complejas
- Backend de aplicaciones *legacy* con SQL tradicional

Amazon DynamoDB

Usar cuando:

- Se necesita baja latencia en lecturas/escrituras
- Se espera carga variable o muy alta
- Se puede diseñar la base orientada a consultas específicas

Ejemplos:

- Aplicaciones móviles o gaming en tiempo real
- Catálogos de productos o dashboards IoT
- Sistemas de autenticación o sesiones

Desde la Perspectiva de un Desarrollador

Factor	DynamoDB	RDS
Familiaridad	Requiere nuevo enfoque mental	Más tradicional (SQL)
Costos	Escalable por demanda	Costos fijos según instancia
Modelado de datos	Basado en patrones de acceso	Normalizado o relacional
Dev/Prod igualdad	Fácil de simular con LocalDynamo	Requiere motor SQL o contenedor

AWS Lambda y el enfoque Serverless

¿Qué es AWS Lambda?

AWS Lambda es un servicio de cómputo serverless que permite ejecutar código sin necesidad de administrar servidores.
Tú escribes la función, defines el evento que la dispara (por ejemplo, una carga en S3), y Lambda se encarga automáticamente de:

- Ejecutar el código
- Escalar bajo demanda
- Administrar disponibilidad y tolerancia a fallos

Ventajas del enfoque Serverless

- **Sin infraestructura que gestionar**
- **Escalabilidad automática según demanda**
- **Pago por ejecución** (no por tiempo encendido)

- **Integración nativa** con servicios como S3, DynamoDB, API Gateway, etc.

Casos de uso comunes

- Procesamiento de archivos (por ejemplo, generación de miniaturas)
- Automatización de flujos con eventos de base de datos o almacenamiento
- Backends para aplicaciones mediante API Gateway
- Tareas programadas (cron jobs) con Amazon EventBridge (CloudWatch Events)

Ejemplo práctico: Miniaturas con Lambda + S3

Esta función Lambda se activa automáticamente cada vez que se sube una imagen a un bucket S3.
Redimensiona la imagen y guarda una miniatura en la carpeta miniaturas/ dentro del mismo bucket.

```
import boto3

from PIL import Image

import io


s3 = boto3.client('s3')


def lambda_handler(event, context):

  # Obtener datos del evento

  bucket = event['Records'][0]['s3']['bucket']['name']

  nombre_archivo = event['Records'][0]['s3']['object']['key']


  # Descargar la imagen original desde S3
```

```
response = s3.get_object(Bucket=bucket, Key=nombre_archivo)
imagen_original = Image.open(response['Body'])

# Redimensionar imagen
imagen_original.thumbnail((200, 200))  # miniatura de 200x200 px

# Guardar en buffer y volver a S3
buffer = io.BytesIO()
imagen_original.save(buffer, format='JPEG')
buffer.seek(0)

s3.put_object(
   Bucket=bucket,
   Key=f"miniaturas/{nombre_archivo}",
   Body=buffer,
   ContentType='image/jpeg'
)
return {
   'statusCode': 200,
   'body': f"Miniatura de {nombre_archivo} generada con éxito."
}
```

DevOps y su importancia en el desarrollo moderno

¿Qué es DevOps?

DevOps es una metodología cultural y técnica que busca integrar el desarrollo de software (**Dev**) con las operaciones (**Ops**) para automatizar y mejorar el ciclo de vida del software: **desarrollo, pruebas, despliegue y monitoreo**.

Beneficios principales

- Despliegues frecuentes y confiables
- Reducción de errores humanos
- Detección temprana de fallos
- Tiempo de entrega más corto
- Mayor colaboración entre equipos

Herramientas clave en AWS

Servicio AWS	Función
CodeCommit	Repositorio Git privado administrado por AWS
CodeBuild	Compilación, pruebas y empaquetado automatizado
CodeDeploy	Despliegue automático a EC2, Lambda o ECS
CodePipeline	Orquestación completa del flujo CI/CD

graph TD

A[CodeCommit] --> B[CodeBuild]

B --> C[CodeDeploy]

C --> D[Producción]

CI/CD con AWS CodePipeline + CodeBuild

¿Qué es CodePipeline?

AWS CodePipeline es un servicio que permite automatizar flujos de integración y entrega continua (CI/CD).
Orquesta tareas como:

- Obtener código desde una fuente (ej. GitHub, CodeCommit)
- Compilar el proyecto (con CodeBuild)
- Configurar CodePipeline: establecer las etapas del flujo (fuente, construcción, despliegue)
- Desplegar automáticamente a entornos como S3, ECS, Lambda, etc.

Componentes principales de un pipeline

1. **Fuente:** Repositorio de código (ej. GitHub, CodeCommit)
2. **Compilación:** Ejecutada con CodeBuild
3. **Despliegue:** Servicios como S3, Lambda, ECS, Elastic Beanstalk, etc.

Ejemplo de `buildspec.yml` para Python

```yaml
version: 0.2

phases:
  install:
    runtime-versions:
      python: 3.10
    commands:
      - pip install -r requirements.txt

  build:
    commands:
      - echo "Ejecutando pruebas..."
      - pytest  # o cualquier otro comando de pruebas

artifacts:
  files:
```

```
    - '**/*'
```

Ejemplo de `buildspec.yml` para C# (.NET Core)

```yaml
version: 0.2

phases:
  install:
    runtime-versions:
      dotnet: 6.0
    commands:
      - echo "Instalando dependencias..."
      - dotnet restore

  build:
    commands:
      - echo "Compilando proyecto..."
      - dotnet build --configuration Release

  post_build:
    commands:
      - echo "Publicando artefactos..."
      - dotnet publish -c Release -o output

artifacts:
  files:
    - '**/*'
  base-directory: output
```

5. Amazon Simple Storage Service (S3)

Amazon Simple Storage Service (S3) es un servicio de almacenamiento de objetos altamente escalable, duradero y seguro ofrecido por **AWS**.
Permite almacenar y recuperar cualquier cantidad de datos desde cualquier lugar en la web.

Características clave

- **Almacenamiento basado en objetos** (no en bloques o archivos)
- **Alta durabilidad**: 99.999999999%
- **Alta disponibilidad**: 99.99%
- **Escalabilidad ilimitada**
- **Seguridad integrada**: encriptación y control de acceso

Usos comunes en aplicaciones en la nube

- **Almacenamiento de archivos estáticos**: imágenes, CSS, JS para sitios web
- **Backup y recuperación de desastres**
- **Almacenamiento de datos para análisis** (Data Lakes)
- **Hosting de sitios web estáticos**
- **Almacenamiento de logs y archivos de registro**
- **Distribución de contenido** mediante integración con CloudFront
- **Almacenamiento de archivos** para aplicaciones móviles y web

Buenas prácticas con S3

- Usar **nombres de buckets únicos globalmente**
- Implementar **políticas de acceso adecuadas**
- Considerar el uso de **versionado** para protección contra eliminaciones accidentales
- Usar **clases de almacenamiento adecuadas**:
    - Standard
    - Intelligent-Tiering
    - Glacier
    - Entre otras
- Habilitar **encriptación para datos sensibles**
- Usar **transferencia acelerada** para distribuciones globales