

Examen Final

CE191 Software Architecture / Cloud Computing

1. Describe los pros y cons de una arquitectura de software de microservicios.

a. Definición

Teniendo en cuenta lo que significa una arquitectura de microservicios, se puede definir de la siguiente manera son un conjunto de pequeñas, las cuales implementan aplicaciones más grandes y complejas de forma continua, esto quiere decir que puede se pueden implementar de forma independiente, por motivo que están descentralizados y por ende se pueden actualizar de forma autónoma e independiente. Estos microservicios constituyen la parte importante de la entrega continua de mejoras.

b. Pros

- i. *Agilidad en las mejoras:* Como se mencionó anteriormente debido que ahora están separados los servicios, el progreso de mejoras es mucho mejor y sobretodo más rápido, esto debido a varias razones como están divididos también se podrán entregar ciclos de entrega más cortos
- ii. *Fácil escalabilidad:* Debido a que todo está separado de una manera, el hecho de las mejoras que se le pueden hacer son mayores, esto a que se le pueden hacer mejoras tanto horizontales como verticales, más allá de los límites de los servicios, significando esto que cuando un microservicio necesite aumentar su carga, esté procesos será más rápido.
- iii. *Calidad de entrega:* Esto es consecuente de que cada quien tenga sus roles empresariales y con esto, el equipo se pueda enfocar de una mejor manera a sus metas.

c. Cons

- i. *Expansión descontrolada:* A medida que se van dividiendo los servicios, el hecho de poder gestionar y sobre todo tratar de

coordinar todo los procesos, puede ser un proceso un poco tedioso de mantener.

- ii. Costes de la infraestructura: Por motivo que la expansión de necesidades puede aumentar exponencialmente, igualmente los costos esto va ligado a ejemplos como pruebas, estrategias de implementación, alojamiento, herramientas de monitorización, etc.
- iii. Dificultades en la depuración y monitor: Debido a que a que los microservicios se ejecutan de forma independiente, puede ser mas facil dificil realizar la depuración y sobretodo el monitoreo de los servicios que están interactuando entre ellos.

2. Explica la arquitectura de publish and subscribe.

La arquitectura de "publish and subscribe o pub/sub" se puede llamar como un patrón de mensajería que nos permite comunicarnos entre diferentes componentes o sistemas donde no se necesitan conocer o estar relacionados, esas una de sus innovaciones. En esta arquitectura hay dos roles principales: el publisher y el suscribir.

La forma que se podría entender mejor es que el publicador es el responsable de producir los datos o eventos que se desean publicar, por lo general a un "topic" o "channel". Por otra parte el Suscriptor es el responsable de consumir esos datos o información que se tiene por esos "topics" o "channels" que se comentaron.

Una de las ventajas que nos proporciona o por lo cual es muy utilizado es la opción que nos ofrece acoplamiento suelto (loose coupling) esto entre los dos componentes o sistemas. Como se mencionó los publicadores y suscriptores no se necesitan conocerse entre ellos, lo cual esto lo hace muchísimo más fácil, en cuestión de escalabilidad porque puedes tener múltiples suscriptores que pueda, además para añadir o remover componentes se pueden hacer modificaciones, puedes hacerlos sin afectar a los otros porque permite tener múltiples suscriptores saque puedan consumir de los mismo datos y eventos.

Entonces recapitulando las partes importantes de esta arquitectura, están conformadas por tres componentes principales:

1. Publishers (Publicadores): Son los componentes que generan mensajes y los envían a un canal de mensajes. Los publicadores no conocen a los suscriptores ni a los consumidores, simplemente publican mensajes en un canal de mensajes.
2. Channels (Canales): Son los canales de mensajes a los que se suscriben los suscriptores y que se utilizan para transmitir los mensajes. Los canales pueden ser temáticos (por ejemplo, todos los mensajes relacionados con un tema específico se publican en un solo canal) o distribuidos (por ejemplo, se distribuyen los mensajes a varios canales).
3. Suscriptores (Suscriptores): Son los componentes que se suscriben a un canal de mensajes para recibir los mensajes publicados en ese canal. Los suscriptores pueden recibir mensajes en tiempo real o bien recibir una copia de los mensajes almacenados en el canal.

En cuestión a los pros y contras de de los publicadores y suscriptores:

Pros:

1. Acoplamiento suelto: Esta arquitectura de pub/sub, como se mencionó anteriormente, los componentes no necesita conocer la identidad o ubicación de los demás componentes, lo que permite mejor escalabilidad y flexibilidad en la implementación de nuevos componentes.
2. Escalabilidad: A como está configurada la arquitectura, es muy escalable y puede manejar grandes cantidades de volúmenes de datos y múltiples componentes que se suscriben a los mismos canales.

Cons:

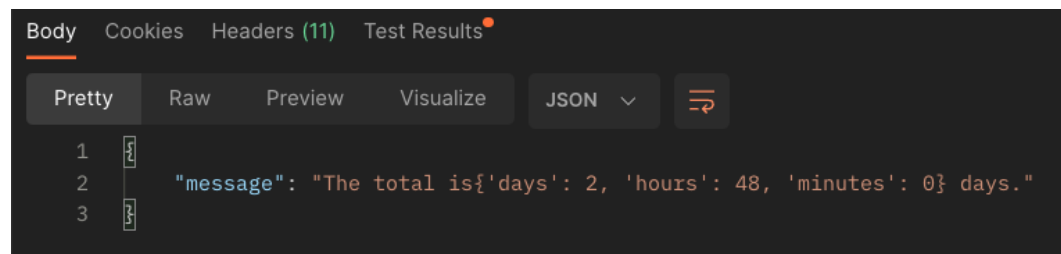
1. Pérdida de mensajes: En algunas ocasiones, los mensajes pueden perderse en el trayecto, ya que puede pasar que los suscriptores no estén disponibles en el momento que se publica el mensaje, puede ser algún motivo por el cual no llego el mensaje.
2. A priori podríamos pensar que tanto tráfico de datos por la red se podría saturar, pues sí podría darse el caso, cuando se publican demasiados mensajes o si el contenido de los mensajes son demasiado grandes.

3. Crea una Lambda + API Gateway que reciba como parámetro un rango de fechas con formato YYYYMMDD y regrese la cantidad de días, horas y minutos entre dichas fechas.

- La forma que calcule básicamente fue más que todo calcular respecto a los días las horas, los minutos todavía me faltaron depurar la fórmula
- Para entender un poco como funciona mostraré un ejemplo de la API

```
GET https://qjq1b5i1zi.execute-api.us-east-1.amazonaws.com/develop/rango/20230301-20230303
```

- Respecto a estos valores introducidos, el input es el siguiente



```

Body  Cookies  Headers (11)  Test Results
Pretty  Raw  Preview  Visualize  JSON
1  {
2    "message": "The total is{'days': 2, 'hours': 48, 'minutes': 0} days."
3  }

```

- Básicamente toma los días como 24 horas así que por cada día se sumarán 24 horas, así consecutivamente
- Lambda
 - Arn:aws:lambda:us-east-1:292274580527:function:date_roldan_beta
- API Gateway
 - roldanApiV2

- <https://jqj1b5i1zi.execute-api.us-east-1.amazonaws.com/develop/rango/20230301-20230303>

4. Replica la respuesta anterior, pero en este nuevo end point regresa HTML, creando un dynamic web site.

5. Describe la diferencia entre los S3 object storage class standard, infrequent access y Glacier.

- S3 object storage class standard:
 - Es un almacenamiento de objetos que ofrece:
 - Escalabilidad
 - Disponibilidad de tus datos
 - Seguridad
 - Rendimiento
 - El uso de esta herramienta puede ser usado para diferentes cosas e industrias como por ejemplo filtración de datos, aplicaciones nativas de la nube o aplicaciones móviles.
 - Diseñada para datos de acceso frecuente
 - Almacena datos en al menos tres zonas de disponibilidad.
 - Datos que se requiera una alta velocidad de recuperación de ellos.
 - Se maneja un costo rentable el cual es fácil de manejar
- S3 IA (Infrequent access).
 - En cuestión es un poco similar al S3, pero se diferencia por acceder con menos frecuencia a tus datos, aunque tiene la herramienta que cuando necesites tus datos sea rápido el proceso. Otras ventajas que nos ofrece esta alternativa son:
 - Alta durabilidad
 - Alto rendimiento
 - Baja latencia del S3 standard
 - Pequeño cargo mensual
 - Esta gran combinación de bajo costo y alto rendimiento hace al S3 Standar IA perfecto para almacenamiento a largo plazo, copias de seguridad, por mencionar algunas

- S3 Glacier.
 - Proporciona almacenamiento a bajo costo diseñado para archivado de datos.
 - Te cobran muy poco por cada GB pero te cobran bastante por retirar datos (tiempos de demora de minutos a hora), es la forma que se calcula el costo por la transacción.
 - Optimizada para los datos de archivo.

En conclusión la diferencia entre estos 3 tipos de almacenamientos proporcionados por AWS, creo que los podrías diferenciar por velocidad y accesibilidad, dado que el S3 estándar nos ofrece la posibilidad de acceder a nuestro datos con alta disponibilidad y baja latencia para recuperarlos. Por otra parte el S3 IA nos ofrece la posibilidad de alta durabilidad de nuestros datos, ya que se utiliza más a datos que no se accedan con menos frecuencia pero aun así cuando sea el momento se puedan sean de rápido acceso a ellos. Mientras que por último el S3 Glacier es la opción más económica de las 3, pero es la que la recuperación es más lenta además que se calcula por el tiempo que tarde, así que principalmente es escogida para datos que se acceden raramente.

Como vimos todo es en cuestión que tanto accedas a tus datos y a que velocidad requieras obtenerlos, es la opción a escoger.

6. Explica cómo hacer root de un subdominio de Route 53 a un static web site en S3.

- Primeramente tenemos que tener un bucket S3, así que para este ejemplo este ejemplo vamos a suponer que ya se cuenta con el bucket.
- Así que lo siguiente será saber si nuestro dominio estático se encuentra en la lista de los sitios hosteados para esto se utiliza el siguiente comando:
 - `aws route53 list-hosted-zones`
- Para poder unir nuestro website con el subdominio tenemos que hacer un CNAME record, lo que se busca es unir nuestro bucket con el subdominio hacia el dominio principal, de esta forma lo redireccionaremos, la siguiente información es un ejemplo de cómo iría acomodada la información del JSON, básicamente lo que hacemos es convertir nuestro url enorme que usábamos en un url mas simple de entender.

- `"Comment": "Updating record to add a new CNAME record in Route 53", "Changes": [{ "Action": "CREATE", "ResourceRecordSet": { "Name": "david.rolan.cetystijuana.com", "Type": "CNAME", "TTL": 600, "ResourceRecords": [{ "Value": "david.rolan.cetystijuana.com.s3-website-us-east-1.amazonaws.com" }] } }] }`
- Ahora ya teniendo nuestro JSON ahora toca enviarlo por medio de Route53, el comando a utilizar es el siguiente:
 - `aws route53 change-resource-record-sets --hosted-zone-id Z03346142C3RKH191036Y --change-batch "Ruta exacta de tu archivo"`
 - Nota importante, el hosted-zone-id: Se obtuvo desde el primer comando de la lista de zonas hosteadas
- En el output nos enfocaremos en la opción que dice Pending, esta tarda un poco pero puede tomar su tiempo la forma de comprobar cuando ya está sincronizada o aprobada es mediante el mismo comando o viendo si en la lista de páginas ya te encuentras visible, esto se realiza mediante este comando.
 - `aws route53 list-resource-record-sets --hosted-zone-id Z03346142C3RKH191036Y`

7. Escribe el código en python para leer y borrar un mensaje de SQS

import boto3

```
import boto3
```

```
def read_messages(queue_name, max_messages=10,
visibility_timeout=30, wait_time=20):
```

```
    """
```

```
    Lee los mensajes de una cola de SQS y devuelve una
    lista de diccionarios con el cuerpo del mensaje y el
    receipt_handle.
```

```
    queue_name (str): El nombre de la cola de SQS.
```

`max_messages (int)`: El número máximo de mensajes a leer en nuestro caso maximo 10.

`visibility_timeout (int)`: La duración en segundos durante la cual los mensajes se mantienen ocultos para otros consumidores.

`wait_time (int)`: El tiempo de espera en segundos para recibir mensajes de la cola, en nuestro caso 20.

Nos regresa una lista una lista de diccionarios con el cuerpo del mensaje y el receipt_handle.

```
"""
sqs = boto3.client('sqs')
queue_url =
sqs.get_queue_url(QueueName=queue_name) ['QueueUrl']
response = sqs.receive_message(QueueUrl=queue_url,
MaxNumberOfMessages=max_messages,
VisibilityTimeout=visibility_timeout,
WaitTimeSeconds=wait_time)

messages = []
if 'Messages' in response:
    for message in response['Messages']:
        messages.append({
            'body': message['Body'],
            'receipt_handle': message['ReceiptHandle']
        })
    return messages

def delete_message(queue_name, receipt_handle):
```



```

#La funcion nso elimina un mensaje especifico de una
cola

#Nos crea un cliente de SQS
sqs_client = boto3.client('sqs')

#Utiliza el metodo get_queue para obtener la URL de la
cola de SQS
queue_url =
sqs.get_queue_url(QueueName=queue_name) ['QueueUrl']

#Elimina el mensaje de la cola que especificamos del
queue_name asi tambien utilizando el identificador del
mensaje 'receipt_handle',
response = sqs_client.delete_message(
    queue_url = queue_url,
    ReceiptHandle=receipt_handle,
)
print(response)

```

8. Explica las diferencias entre SNS y SQS

SQS es un servicio de colas (queues) y topics altamente escalables, sencillos de usar y que no necesitan la configuración de agentes de mensajes. En este método los mensajes se colocan en una cola y se entregan a los consumidores en un orden en específico, este es útil cuando se necesita procesamiento en segundo plano para cuando estos estén listos se realice la operación.

SNS también son un sistema de servicios de mensajes administrado, pero este proporciona la entrega de mensaje mediante los publicadores a los suscriptores, donde los publicadores se comunican de forma asíncrona con los suscriptores mediante envío de mensajes por topics, que es un punto de accesos lógico y un canal de comunicación.

En resumen, la mayor diferencia entre SNS y SQS es su capacidad de envío de mensajes. SQS puede enviar muchos mensajes a un solo destinatario, mientras que

SNS puede enviar un mensaje a múltiples destinatarios. Además, con SNS, el proceso de envío de mensajes es instantáneo, mientras que con SQS, los mensajes son recolectados y enviados después de cierto tiempo. Por lo tanto, SNS puede tener múltiples suscriptores, mientras que SQS solo puede tener un receptor o consumidor para cada mensaje.

9. Explica el ciclo de TDD.

Por sus siglas en inglés (Test Driven Development) es el proceso en el cual los "test cases" son escritos antes de que se validen los casos. Esto depende de la repetición de pequeños ciclos para poder lograrlo. Es una técnica en la cual se automatizados "unit test" son usados para dirigir el diseño y el desacoplamiento de las dependencias y la automatización ayuda a asegurar que el software cumpla con los requisitos. Básicamente por cada test que se falla, se escribe el código para pasar el test y se refactoriza el código para mejorarlo.

Estos son los pasos que se suelen seguir:

1. Agrega un test -> Escribe un caso de prueba que describa la función completamente. Para hacer los casos de prueba, el desarrollador debe entender las características y requisitos utilizando historias de usuario y casos de uso.
2. Ejecuta todos los casos de prueba y asegúrate de que el nuevo caso de prueba falle.
3. Escribe el código que pasa el caso de prueba.
4. Ejecuta los casos de prueba.
5. Refactoriza el código -> Esto se hace para eliminar la duplicación de código.
6. Repite los pasos mencionados anteriormente una y otra vez

Para cumplir con las fases se tiene que hacer lo siguiente, ya que el ciclo TDD consta de estas tres fases:

1. Red: En esta fase, se escribe una prueba automatizada para la funcionalidad que se va a desarrollar. Esta prueba fallará porque todavía no tenemos la funcionalidad.

2. *Green*: En esta fase, se escribe el código de producción necesario para que la prueba automatizada se pase.
3. *Refactorizar*: En esta fase, se mejora el código de producción recién escrito para hacerlo más eficiente, legible y mantenible.

Estas fases agregadas van implícitas en los 6 pasos mencionados anteriormente. Por lo que el ciclo se repite.

Pros:

- Las prueba unitarias te dan mucha retroalimentación sobre tus cuestiones
- La calidad del diseño se incrementa muy rápido, lo cual ayuda a mejorar.
- El método TDD, nos ayuda como una red segura en contra de los fallos o bugs.
- El método nos ayuda a saber que nuestra aplicación cumpla con los requerimientos declarados antes.

Cons:

- El hecho de escribir muchas pruebas sobre tu código, puede llevar consigo mucho consumo del tiempo más que todo al principio del desarrollo.
- El método TDD se enfoca principalmente en pruebas unitarias sobre tu código, lo cual puede que no atrapes todos los errores.
- Algunos desarrolladores pueden dejarse llevar por la escritura de las pruebas y acabar sobrepasando la solución, algo que ocurre muy seguido y esto conlleva a crear código más complejo del necesario.

10. Explica si el amor lo puede todo y por qué

Depende mucho del contexto pero bueno podemos dividir esto en tres vertientes, la primera que se me viene a la mente es la motivación que nos da realizar o en este caso no tenemos, como a que sabemos que el tener motivación nos va a llevar a mejorar poquito o simplemente superar cualquier obstáculo que tengamos en ese momento, como podría ser flojera o cansancio, es en esos momentos cuando la motivación sale a la luz. Pasando al siguiente punto donde el amor lo puede ser podría ser resolver conflictos amorosos

podría ser un ejemplo cuando por más grande que sea el problema cuando en verdad se quieren las personas de forma mutua y buscan avanzar entre ellas siempre habra una solucion de prosperar, pero ojo aquí es cuando ambas partes están comprometidas porque puede que una de ellas, ya todo se va dejando fuera o el motivo que se estaba resolviendo se pasa a deshacer, son varias hipótesis pero entre lazos emocionante el amor podría resolver gran parte de los conflictos que se han vivido incluso hablando de historia como podría ser las guerras. Básicamente con todo esto podemos entender que el amor nos hace ver las cosas de forma diferente, porque no implemente las vertiente de las emociones porque son ellas las que nos hacen solucionar problemas y mejorar bastantes situaciones.

Una parte donde el amor no lo puede todo, es mas relacionado a las enfermedades terminales, es donde ya no se puede hacer nada por el paciente básicamente va a morir, en estas situaciones donde la ciencia no tiene solución hacia el problema, por mas que se quiera a la persona o se haga todo el esfuerzo, no se va a llegar a alguna solución o mejoría. Justo aquí entra la otra parte de la psicológica donde una persona al haber pasado una situación traumática, las cuales podrían ser muchísimas, en estas situaciones hay unas las cuales las cuales si se pueden sanar o resolver pero lamentablemente hay otras las cuales nunca se van a entender.

En conclusión, creo que mas que lamentablemente en esta vida la mayorías de las cosas o si bien son correctas o incorrectas, blancas o negras, este es uno de estos temas no hay una respuesta acertada, ya que como ya se mencion hay situaciones donde se pueden resolver situaciones con el amor, mientras va a ver otras en las que no se podrá y está bien es parte de la vida, no todo es color de rosa, se tiene que ver o vivir situaciones así para entender lo bonito de vivir.