



# PRÁCTICA 5: DESACOPLAMIENTO

RAÚL MATEUS SÁNCHEZ

COMPUTACIÓN EN LA NUBE  
Escuela de Ingeniería Informática

# Índice

Introducción.....	3
Objetivos.....	3
Desarrollo de la práctica .....	4
1.    Implemente su aplicación de ejemplo en el lenguaje deseado y despléguela en Amazon (contenedor o instancia).....	4
2.    Divida su aplicación en tres aplicaciones distintas (fA, fB, fC) y modifíquelas para que cada aplicación le pase su resultado a la siguiente usando colas (e.g. SQS o redis) y despléguelas en AWS. En el ejemplo anterior 'x' e 'y' se mandarían por colas.....	5
3.    Configure un 'topic' en AWS SNS para que fA se suscriba y tome su dato de entrada de ahí. En el ejemplo anterior, fA recibiría 'w' de un topic. ....	9
Presupuesto y estimación de gastos .....	13
Fuentes de información .....	14

# Introducción

Hoy en día, la computación en la nube tiene uno de sus pilares en la programación en paralelo y el procesamiento distribuido. En este contexto, el desacoplamiento en colas permite mejorar la eficiencia y optimizar el uso de recursos separando aplicaciones para que se comuniquen por estas.

## Objetivos

El objetivo de esta práctica es, principalmente, experimentar con los servicios de colas y eventos de AWS para desacoplar una aplicación monolítica en un conjunto de microservicios escalable.

# Desarrollo de la práctica

1. Implemente su aplicación de ejemplo en el lenguaje deseado y despléguela en Amazon (contenedor o instancia)

Para implementar la aplicación, se parte de la base de la aplicación mostrada como ejemplo en el guion de la práctica, donde se hará lo mismo, pero en Python:

```
import time

def fA(DataA):
    print(DataA)
    time.sleep(5)
    return DataA+'A'

def fB(DataB):
    print(DataB)
    time.sleep(4)
    return DataB+'B'

def fC(DataC):
    print(DataC)
    time.sleep(3)
    return DataC+'C'

w = 'Inicio:'
x = fA(w)
y = fB(x)
z = fC(y)
print(z)
```

Si se quisiera desplegar en un contenedor o instancia de Amazon, se puede seguir los pasos de prácticas anteriores, ya sea mediante Docker o mediante una instancia de EC2. En este caso, el despliegue solo servirá para probar la aplicación y posteriormente gestionar los mensajes de las colas pertinentes. Es por ello por lo que, en este caso, se desplegará en una instancia únicamente.

```
rmateus@RAUL-LAPTOP1:~/p5-cn$ python3 p5-cn-1.py
Inicio:
Inicio:A
Inicio:AB
Inicio:ABC
```

Figura 1: Prueba de aplicación en local

2. Divida su aplicación en tres aplicaciones distintas (fA, fB, fC) y modifíquelas para que cada aplicación le pase su resultado a la siguiente usando colas (e.g. SQS o redis) y despléguelas en AWS. En el ejemplo anterior 'x' e 'y' se mandarían por colas

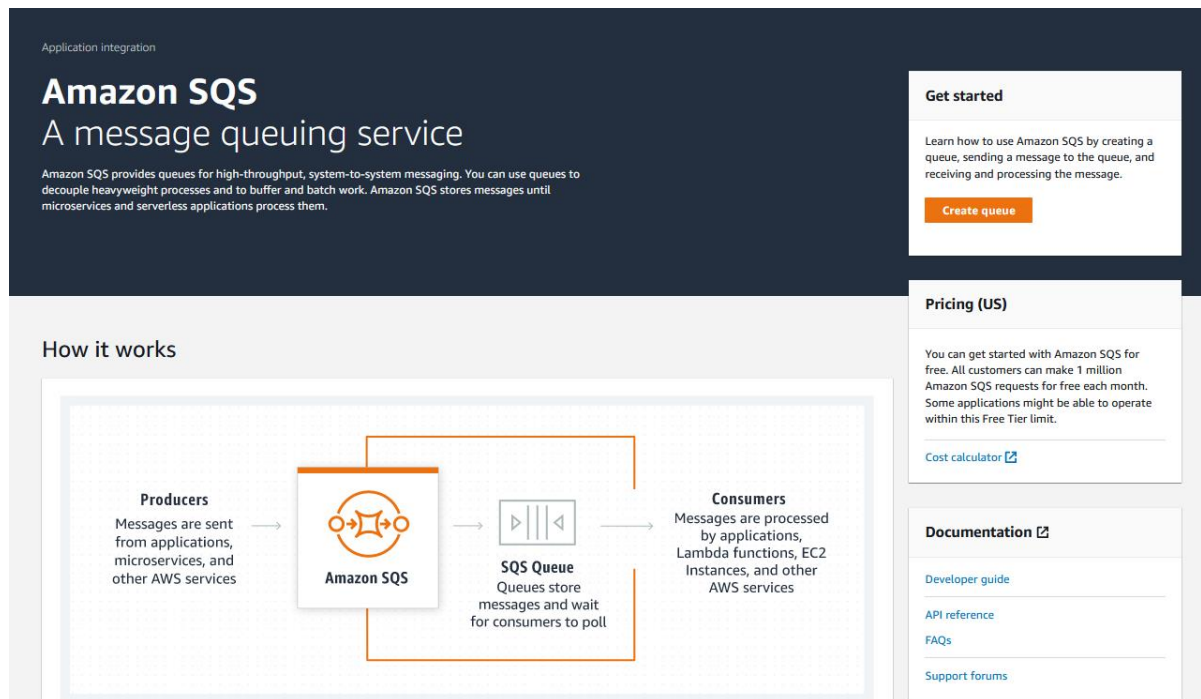


Figura 2: Interfaz de Amazon SQS

Para dividir nuestra aplicación en tres distintas, cada función se “desplegará” en una cola, cuya función será tratar el dato de entrada que reciba para tratarlo, y enviar su resultado a la siguiente cola. Para ello, se empleará Amazon SQS.

Details

Type

Choose the queue type for your application or cloud infrastructure.

You can't change the queue type after you create a queue.

Standard Info

At-least-once delivery, message ordering isn't preserved

- At-least once delivery
- Best-effort ordering

FIFO Info

First-in-first-out delivery, message ordering is preserved

- First-in-first-out delivery
- Exactly-once processing

Name

fA\_p5\_cn

A queue name is case-sensitive and can have up to 80 characters. You can use alphanumeric characters, hyphens (-), and underscores (\_).

Configuration

Set the maximum message size, visibility to other consumers, and message retention. Info

Visibility timeout Info

30

Seconds

Should be between 0 seconds and 12 hours.

Delivery delay Info

0

Seconds

Should be between 0 seconds and 15 minutes.

Receive message wait time Info

10

Seconds

Should be between 0 and 20 seconds.

Message retention period Info

4

Days

Should be between 1 minute and 14 days.

Maximum message size Info

256

KB

Should be between 1 KB and 256 KB.

Figura 3: Configuración de creación de cola en SQS

Desde la interfaz de SQS se crearán tres colas. En este punto, dentro de la configuración, seleccionaremos que la cola sea de tipo “Estándar”. Esto se debe a que si bien podría ser más correcto que la cola fuese de tipo FIFO, en el enunciado no se requiere que el orden de los mensajes se preserve, además de que, si se implementa bien, a cada cola le llegará siempre el mismo resultado de la cola anterior, por lo que no habrá diferencias entre mensajes recibidos por una misma cola si no modificamos el input (“Inicio:”).

Por otra parte, se le asigna un nombre (deberá coincidir con el nombre de la aplicación pertinente) y aumentaremos el tiempo de espera de recepción de mensaje a 10 segundos.

Este proceso se repetirá tres veces, hasta crear las tres colas que se ha comentado, cada una con un nombre característico.

Queues (1)										<div> <div></div> <div>Edit</div> <div>Delete</div> <div>Send and receive messages</div> </div>
<div> <div></div> <div>Search queues by prefix</div> </div>										
	Name	Type	Created	Messages available	Messages in flight	Encryption				Co
	fA_p5_cn	Standard	18 dic 2022, 12:14:30 WET	0	0	Amazon SQS key (SSE-SQS)				-
	fB_p5_cn	Standard	18 dic 2022, 12:14:54 WET	0	0	Amazon SQS key (SSE-SQS)				-
	fC_p5_cn	Standard	18 dic 2022, 12:15:08 WET	0	0	Amazon SQS key (SSE-SQS)				-

Figura 4: Colas creadas en SQS

6

A continuación, se actualiza las credenciales de AWS en el fichero ~/.aws/credentials, ya que no ha sido posible asignar los permisos necesarios para evitar la entrada de estas credenciales a mano.

En la instancia, separaremos nuestra aplicación en tres, pero antes, se instala boto3 mediante la orden:

```
python3 -m pip install boto3
```

```
import boto3
if __name__ == "__main__":
    sqs = boto3.resource('sqs', region_name='us-east-1')
    input_queue = sqs.get_queue_by_name(QueueName='fA_p5_cn')
    response = input_queue.send_message (MessageBody="Inicio:')
```

**Figura 5:** Script para introducir mensaje en la cola A

```
import boto3
import json
if __name__ == "__main__":
    sqs = boto3.resource('sqs', region_name='us-east-1')
    input_queue = sqs.get_queue_by_name(QueueName='fA_p5_cn')
    output_queue = sqs.get_queue_by_name(QueueName='fB_p5_cn')
    while(True):
        for message in input_queue.receive_messages():
            message_dict = json.loads(message.body)
            message_body = message_dict["Message"]
            new_message = message_body + "A"
            print("Message received: " + message_body)
            print("Message to send: " + new_message)
            output_response = output_queue.send_message(MessageBody=new_message)
            if message.delete():
                print("Message deleted successfully\n")
        break
```

**Figura 6:** Script de cola A

```
import boto3
if __name__ == "__main__":
    sqs = boto3.resource('sqs', region_name='us-east-1')
    input_queue = sqs.get_queue_by_name(QueueName='fB_p5_cn')
    output_queue = sqs.get_queue_by_name(QueueName='fC_p5_cn')
    while(True):
        for message in input_queue.receive_messages():
            message_body = message.body
            new_message = message_body + "B"
            print("Message received: " + message_body)
            print("Message to send: " + new_message)
            output_response = output_queue.send_message(MessageBody=new_message)
            if message.delete():
                print("Message deleted successfully\n")
~
```

**Figura 7:** Script de cola B

```

import boto3

if __name__ == "__main__":
    sqs = boto3.resource('sqs', region_name='us-east-1')
    input_queue = sqs.get_queue_by_name(QueueName='fC_p5_cn')
    while(True):
        for message in input_queue.receive_messages():
            message_body = message.body
            new_message = message_body + "C"
            print("Message received: " + message_body)
            print("Message to send: " + new_message)
            if message.delete():
                print("Message deleted successfully\n")

        break

```

**Figura 8:** Script de cola C

Se destaca el uso de la librería JSON para recoger el mensaje de entrada que se enviará desde el *topic*. En este ejemplo, se usa una versión modificada de esa cola sin la librería JSON, pues no es necesaria. Además, con el fin de probar estos scripts, se crea uno, visible en la **Figura 5**: Script para introducir mensaje en la cola A, para que introduzca el mensaje a la cola A, siendo el resultado de la ejecución el siguiente:

```

rmateus@RAUL-LAPTOP1:~/p5-cn$ python3 p5-cn-2-cola.py
rmateus@RAUL-LAPTOP1:~/p5-cn$ vim p5-cn-2-fA.py
rmateus@RAUL-LAPTOP1:~/p5-cn$ python3 p5-cn-2-fA.py
Message received: Inicio:
Message to send: Inicio:A
Message deleted successfully

rmateus@RAUL-LAPTOP1:~/p5-cn$ python3 p5-cn-2-fB.py
Message received: Inicio:A
Message to send: Inicio:AB
Message deleted successfully

rmateus@RAUL-LAPTOP1:~/p5-cn$ python3 p5-cn-2-fC.py
Message received: Inicio:AB
Message to send: Inicio:ABC
Message deleted successfully

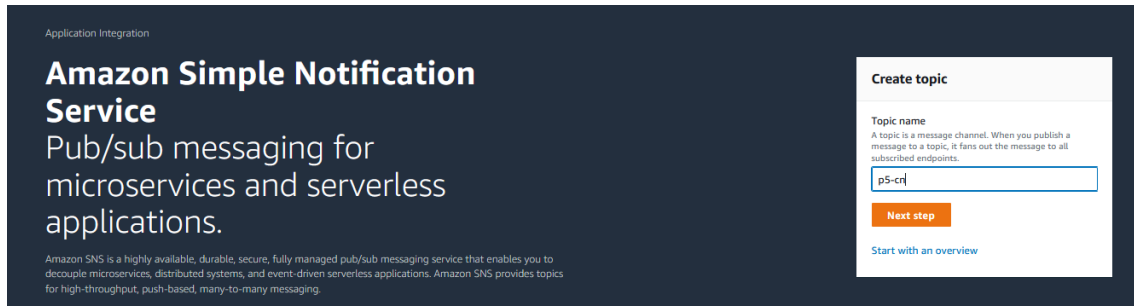
rmateus@RAUL-LAPTOP1:~/p5-cn$

```

**Figura 9:** Prueba de funcionamiento



3. Configure un 'topic' en AWS SNS para que fA se subscriba y tome su dato de entrada de ahí. En el ejemplo anterior, fA recibiría 'w' de un topic.



**Figura 10:** Interfaz de Amazon SNS

Desde la interfaz de SNS, se creará un *topic*. Para ello, elegiremos la opción estándar de nuevo y se introduce un nombre. Con esto será suficiente.

The image shows the 'Create topic' form in the AWS SNS console. The form has a title 'Create topic' at the top. Below it is a 'Details' section. Under 'Details', there's a 'Type' section with an 'Info' link. A note says 'Topic type cannot be modified after topic is created'. There are two radio button options: 'FIFO (first-in, first-out)' and 'Standard'. The 'Standard' option is selected. Below the radio buttons are two boxes listing features for each type. The 'Standard' box lists: 'Best-effort message ordering', 'At-least once message delivery', 'Highest throughput in publishes/second', and 'Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints'. Below the type selection is a 'Name' section with a text input field containing 'p5-cn'. A note below the field says 'Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (\_).' Below the name section is a 'Display name - optional' section with an 'Info' link. A note says 'To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.' There is a text input field containing 'My Topic'. A note below the field says 'Maximum 100 characters.'

**Figura 11:** Creación del "topic"

### Create subscription

**Details**

**Topic ARN**

**Protocol**  
The type of endpoint to subscribe

**Endpoint**  
Only Amazon SQS standard queues will be listed and can receive notifications from an Amazon SNS standard topic.

☐ Enable raw message delivery

After your subscription is created, you must confirm it. [Info](#)

► **Subscription filter policy - optional** [Info](#)  
This policy filters the messages that a subscriber receives.

► **Redrive policy (dead-letter queue) - optional** [Info](#)  
Send undeliverable messages to a dead-letter queue.

Cancel

Create subscription

**Figura 12:** Creación de la suscripción

### Subscribe to Amazon SNS topic [Info](#)

**Amazon SNS topic**  
To allow your queue to receive messages from an Amazon SNS topic, subscribe it to an Amazon SNS topic.

Specify an Amazon SNS topic available for this queue.

Cancel

Save

**Figura 13:** Suscripción al topic

Una vez el “topic” haya sido creado, se realiza la suscripción al mismo desde la cola A, con la finalidad de que este obtenga su dato de entrada desde ahí. Ya suscrita la cola, se procede a enviar un mensaje a esta. Para ello, desde la opción “Publish message”, sin nada a destacar en la configuración del mismo, se introduce el cuerpo del mensaje, que, en nuestro caso, será la *string* “Inicio”.

### Message details

Topic ARN  
arn:aws:sns:us-east-1:340954026236:p5-cn

Subject - *optional*

Maximum 100 printable ASCII characters

Time to Live (TTL) - *optional* [Info](#)  
This setting applies only to mobile application endpoints. The number of seconds that the push notification service has to deliver the message to the endpoint.

### Message body

Message structure

☒ Identical payload for all delivery protocols.  
The same payload is sent to endpoints subscribed to the topic, regardless of their delivery protocol.

☐ Custom payload for each delivery protocol.  
Different payloads are sent to endpoints subscribed to the topic, based on their delivery protocol.

Message body to send to the endpoint

1 Inicio:

**Figura 14:** Publicación de mensaje

```

rmateus@RAUL-LAPTOP1:~/p5-cn$ python3 p5-cn-2-fA.py
Message received: Inicio:
Message to send: Inicio:A
Message deleted successfully

rmateus@RAUL-LAPTOP1:~/p5-cn$ python3 p5-cn-2-fB.py
Message received: Inicio:A
Message to send: Inicio:AB
Message deleted successfully

rmateus@RAUL-LAPTOP1:~/p5-cn$ python3 p5-cn-2-fC.py
Message received: Inicio:AB
Message to send: Inicio:ABC
Message deleted successfully

```

**Figura 15:** Resultado de las tres colas

Como se ve, ya con el mensaje enviado, se ejecutan las tres colas, recordando que ahora si usaremos la librería JSON para no obtener un mensaje enorme, quedándonos solo con el dato de entrada.

# Presupuesto y estimación de gastos

Para esta práctica, el presupuesto total restante de los 100\$ iniciales es de 91,81\$, quitando así el gasto de las prácticas anteriores. Se puede considerar este valor como presupuesto total, dado que no existen más prácticas y las cuentas cerrarán el 31 de diciembre.

El coste de la práctica vendrá dado, principalmente, por la instancia que se usará para desplegar la aplicación. En este punto, empleando una AMI de Amazon Linux y un tipo de instancia t2.micro, el coste de cada instancia por hora es de 0.0116\$. Además, el uso de volúmenes SSD de uso general (gp2) cuesta 0,10\$ por GB-mes.

Amazon calcula el coste de Amazon EBS como el número de instancias, por su capacidad y por su precio GB-mes. Es por ello por lo que, mensualmente, el coste del almacenamiento de EBS será de:

$$8 \text{ GB} * 1 \text{ instancia} * 0.10\$ = 0.80\$$$

Añadiendo el coste de la propia instancia, el precio mensual será:

$$0.80\$ + (0.0116\$) * (30 \text{ días} * 24 \text{ horas}) = 9,15\$$$

Por otra parte, la aplicación, para ser correctos, debería haberse desplegado en tres instancias (cada función en una), por lo que el precio mensual pasaría a ser:

$$2,40\$ + (0.0116\$ * 3) * (30 \text{ días} * 24 \text{ horas}) = 27,46\$$$

En cuanto a Amazon SQS, el primer millón de solicitudes/mes incurriría en 0\$, ya que Amazon lo establece gratis. Obviamente nos quedaremos con este precio, ya que, en la realidad, es prácticamente imposible siquiera acercarnos a un millón de solicitudes.

# Fuentes de información

- <https://aws.amazon.com/blogs/developer/using-python-and-amazon-sqs-fifo-queues-to-preserve-message-sequencing/>
- <https://boto3.amazonaws.com/v1/documentation/api/latest/guide/sqs-example-sending-receiving-msgs.html>
- <https://boto3.amazonaws.com/v1/documentation/api/latest/guide/sqs.html>
- <https://stackoverflow.com/questions/42666631/how-to-get-the-object-details-from-sqs-body>
- [https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/python/example\\_code/sqs](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/python/example_code/sqs)
- [https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/python/example\\_code/polly](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/python/example_code/polly)
- <https://aws.amazon.com/es/sqs/pricing/>