



ESCOLA TÈCNICA SUPERIOR
D'ENGINYERIA
Universitat Rovira i Virgili



Práctica 2

Alumno: Alain Martínez Salamanca
Ingeniería Informática
2025

ÍNDICE

GITHUB.....	3
EJERCICIO 1.....	4
EJERCICIO 2.....	8
EJERCICIO 3.....	11

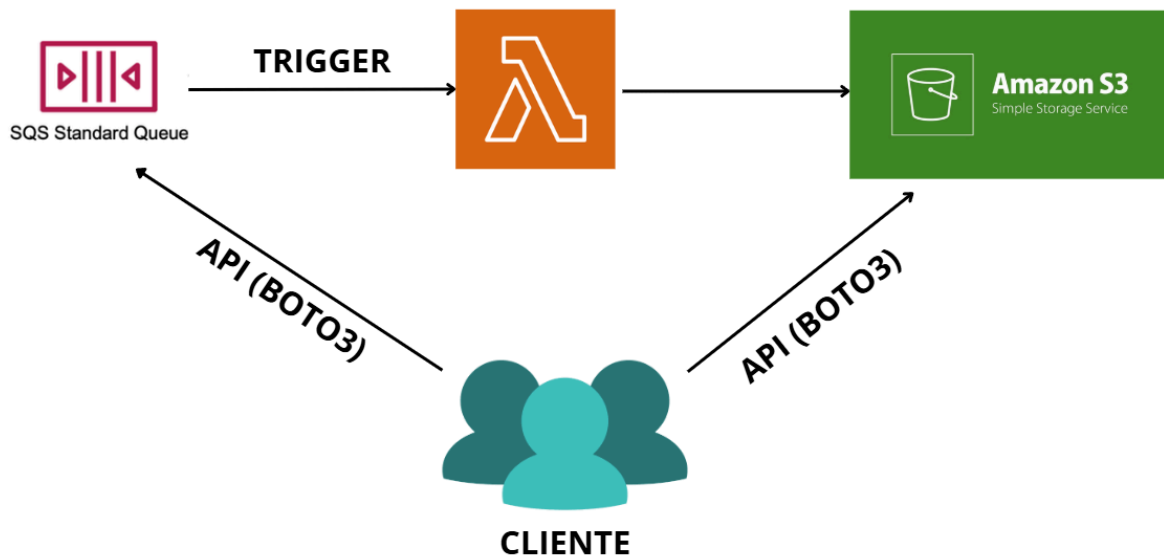
GITHUB

Enlace: https://github.com/alainmartinez23/TASK2_SD.git

EJERCICIO 1

Para el ejercicio 1, he cogido el código del cliente de Redis y le he aplicado una serie de modificaciones. Las funcionalidades son exactamente las mismas.

Antes de entrar en detalle, adjunto una imagen de cómo se ve mi arquitectura:



El cliente utiliza boto3 y S3 para las funcionalidades de ver la lista de insultos, la de textos filtrados y recibir un insulto aleatorio. Simplemente lee los dos ficheros JSON que he añadido para almacenar los insultos. Para estas situaciones, no se utiliza ni SQS ni las Lambdas.

Para las funcionalidades de añadir insultos y de añadir textos filtrados, el cliente añade dichas solicitudes a una cola concreta de SQS. La Lambda tiene un trigger, el cual se activa cada vez que SQS recibe un mensaje. AWS escala horizontalmente las ejecuciones de Lambda de forma eficiente según la carga en la cola. Los mensajes que recibe son de tipo JSON, los cuales tienen una acción ("add_insult", "filter_text"...) y datos, es decir, el insulto o texto a procesar. Cuando se activa la Lambda, esta procesa la solicitud y almacena la respuesta en los ficheros JSON de S3.

Adjunto algunas capturas del script del cliente ejecutado por terminal:

1. Añadir un insulto.
2. Ver toda la lista de insultos disponibles.
3. Obtener un insulto aleatorio.
4. Filtrar un texto.
5. Ver textos filtrados.
6. Salir.

Seleccione una opción: 5

Textos filtrados:


1. Añadir un insulto.
2. Ver toda la lista de insultos disponibles.
3. Obtener un insulto aleatorio.
4. Filtrar un texto.
5. Ver textos filtrados.
6. Salir.


Seleccione una opción: 2



Lista de insultos: ['payaso', 'subnormal']

1. Añadir un insulto.
2. Ver toda la lista de insultos disponibles.
3. Obtener un insulto aleatorio.
4. Filtrar un texto.
5. Ver textos filtrados.

Desde CloudWatch, se puede ver la Lambda lanzada:

Flujos de registros (1)  [Eliminar](#) [Crear flujo de registros](#) [Buscar en todas las secuencias de registro](#)

☐ Coincidencia exacta ☐ Mostrar caducado [Información](#) < 1 > 

☐ Secuencia de registro  Hora del último evento 

<input type="checkbox"/>	2025/05/18/[LATEST]6704c2fcfe374d44a93d99f11086a232	2025-05-18 09:34:14 (UTC+02:00)
--------------------------	---	---------------------------------

Y si entramos dentro de la Lambda, se pueden ver los dos insultos de prueba que he añadido:

▶	2025-05-18T09:34:09.992+02:00	INIT_START Runtime Version: python:3.13.v40 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:67df0a...
▶	2025-05-18T09:34:10.407+02:00	START RequestId: 449fb4b2-628f-53fa-bd49-d411ed218e45 Version: \$LATEST
▶	2025-05-18T09:34:10.760+02:00	Insulto 'payaso' añadido a la lista.
▶	2025-05-18T09:34:10.780+02:00	END RequestId: 449fb4b2-628f-53fa-bd49-d411ed218e45
▶	2025-05-18T09:34:10.780+02:00	REPORT RequestId: 449fb4b2-628f-53fa-bd49-d411ed218e45 Duration: 372.55 ms Billed Duration: 373 ms Memory...
▶	2025-05-18T09:34:14.239+02:00	START RequestId: e0e34d28-13c1-555b-9a3e-2d8b9c6d8df3 Version: \$LATEST
▶	2025-05-18T09:34:14.299+02:00	Insulto 'subnormal' añadido a la lista.

En este ejercicio, es Amazon quien se encarga de hacer el escalado horizontal de forma autónoma. Eso se debe a que tenemos la Lambda configurada con el trigger, lo que significa que cada vez que SQS reciba un mensaje, la Lambda se ejecutará. En el ejercicio siguiente se hace un escalado manual, lanzando Lambdas sin trigger, mediante código.

He realizado una especie de stress test a la Lambda. He hecho un script que solo envía peticiones de filtrar textos, para que siempre se llame a la Lambda.

El objetivo es ver cuánto tarda y cuántas Lambdas se lanzan para ver si realmente está haciendo un escalado horizontal o no según la carga de mensajes.

- 10 mensajes:

```
C:\Users\alain\OneDrive\Escritorio\SD labs\TASK2\EJ1>stress_test.py
Enviando 10 textos a la cola...
[10/10] Enviado

Stress test completado en 1.01 segundos.
Promedio: 0.101 segundos por mensaje.
```

Se lanzan 2 Lambdas:

- | | | |
|--------------------------|---|---------------------------------|
| <input type="checkbox"/> | 2025/05/18/[\$LATEST]fbe0d954c5584d63b78223f6720457f4 | 2025-05-18 09:46:24 (UTC+02:00) |
| <input type="checkbox"/> | 2025/05/18/[\$LATEST]8b278bd8dd6f48d48cc020b4de1ed31a | 2025-05-18 09:46:24 (UTC+02:00) |

- 50 mensajes:

```
C:\Users\alain\OneDrive\Escritorio\SD labs\TASK2\EJ1>stress_test.py
Enviando 50 textos a la cola...
[50/50] Enviado

Stress test completado en 5.31 segundos.
Promedio: 0.1062 segundos por mensaje.
```

- | | | |
|--------------------------|---|---------------------------------|
| <input type="checkbox"/> | 2025/05/18/[\$LATEST]fbe0d954c5584d63b78223f6720457f4 | 2025-05-18 09:49:02 (UTC+02:00) |
| <input type="checkbox"/> | 2025/05/18/[\$LATEST]8b278bd8dd6f48d48cc020b4de1ed31a | 2025-05-18 09:49:02 (UTC+02:00) |

- 300 mensajes:

```
C:\Users\alain\OneDrive\Escritorio\SD labs\TASK2\EJ1>stress_test.py
Enviando 300 textos a la cola...
[100/300] Enviado
[200/300] Enviado
[300/300] Enviado

Stress test completado en 34.33 segundos.
Promedio: 0.1144 segundos por mensaje.
```

Aquí sube la carga de mensajes, por ende sube el número de Lambdas.

- | | | |
|--------------------------|---|---------------------------------|
| <input type="checkbox"/> | 2025/05/18/[\$LATEST]411e52899d2445758a5953778b17e480 | 2025-05-18 09:50:25 (UTC+02:00) |
| <input type="checkbox"/> | 2025/05/18/[\$LATEST]fbe0d954c5584d63b78223f6720457f4 | 2025-05-18 09:50:25 (UTC+02:00) |
| <input type="checkbox"/> | 2025/05/18/[\$LATEST]21fa7c8887884da1b017831ce467225f | 2025-05-18 09:50:17 (UTC+02:00) |
| <input type="checkbox"/> | 2025/05/18/[\$LATEST]07cb2c28810044faabfe8e63c8da0ba1 | 2025-05-18 09:50:17 (UTC+02:00) |

- 700 mensajes:

```
C:\Users\alain\OneDrive\Escritorio\SD labs\TASK2\EJ1>stress_test.py
Enviando 700 textos a la cola...
[100/700] Enviado
[200/700] Enviado
[300/700] Enviado
[400/700] Enviado
[500/700] Enviado
[600/700] Enviado
[700/700] Enviado

Stress test completado en 79.8 segundos.
Promedio: 0.114 segundos por mensaje.
```

<input type="checkbox"/>	2025/05/18/[\$LATEST]4bbeb24f3a448a6a78209b4d1a631c1	2025-05-18 09:53:03 (UTC+02:00)
<input type="checkbox"/>	2025/05/18/[\$LATEST]07cb2c28810044faabfe8e63c8da0ba1	2025-05-18 09:53:03 (UTC+02:00)
<input type="checkbox"/>	2025/05/18/[\$LATEST]411e52899d2445758a5953778b17e480	2025-05-18 09:53:07 (UTC+02:00)
<input type="checkbox"/>	2025/05/18/[\$LATEST]fbe0d954c5584d63b78223f6720457f4	2025-05-18 09:53:09 (UTC+02:00)
<input type="checkbox"/>	2025/05/18/[\$LATEST]21fa7c8887884da1b017831ce467225f	2025-05-18 09:53:09 (UTC+02:00)

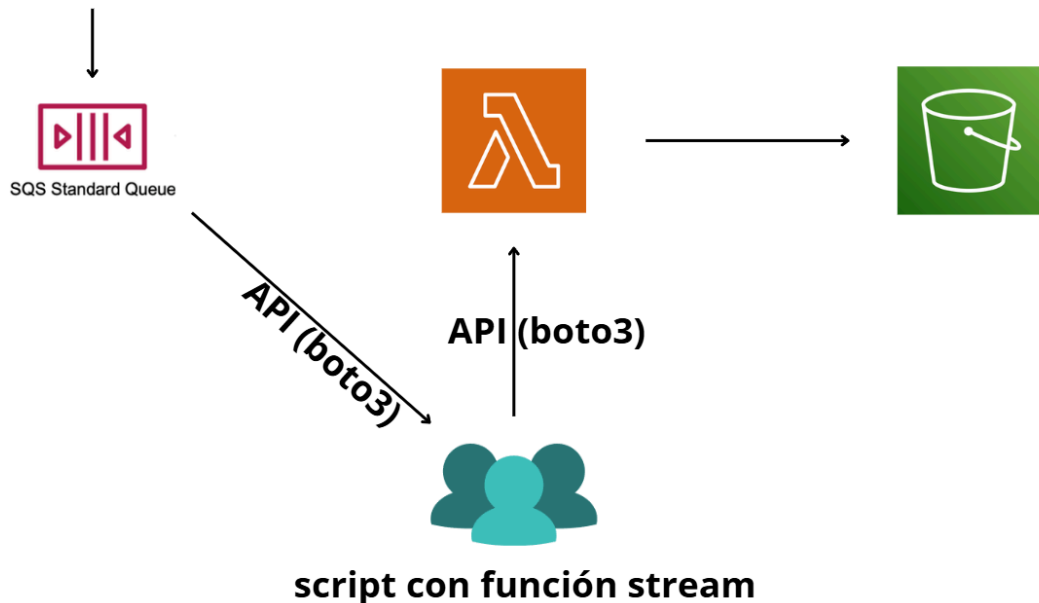
Como se puede ver, el servicio escala de forma dinámica según la carga que tenga.

EJERCICIO 2

Para este ejercicio había que hacer el escalado horizontal de forma manual lanzando lambdas a través de Python.

A modo de resumen, voy a explicar la arquitectura mediante una imagen y después con palabras para que se entienda bien.

STRESS TEST



Tengo un script que contiene la función `stream()` y se encarga de lanzar las Lambdas. Esto lo hace mediante la librería `boto3`, que sirve de API. El código está en un bucle `'while True'` escuchando de SQS y mirando la cantidad de mensajes. Siempre que haya mensajes disponibles, intentará lanzar tantas Lambdas como mensajes haya o, el máximo de Lambdas, que es 10, según cuántos mensajes haya encolados. Pero cabe recalcar que no se lanzan todas las Lambdas que invocamos, ya sea porque una Lambda ejecuta varias peticiones, porque Amazon considera que no se necesitan más, etc.

Se podría reutilizar el código del cliente anterior, quitando el trigger y cambiando los nombres de las colas, y teniendo en ejecución el código con `stream`. Sería hacer exactamente eso para que funcione.

Pero para demostrar el escalado de forma simple, yo he creado un stress test que carga peticiones en SQS y mi script va lanzando Lambdas según la carga. Este stress test envía insultos a SQS y la Lambda los añade a un fichero JSON del bucket de S3.

- 10 insultos:

```
No hay mensajes nuevos.
No hay mensajes nuevos.
No hay mensajes nuevos.
No hay mensajes nuevos.
Cola con 10 mensajes. 10 Lambdas...
Cola con 10 mensajes. 10 Lambdas...
Cola con 9 mensajes. 9 Lambdas...
No hay mensajes nuevos.
No hay mensajes nuevos.
Cola con 2 mensajes. 2 Lambdas...
Cola con 10 mensajes. 10 Lambdas...
No hay mensajes nuevos.
No hay mensajes nuevos.
No hay mensajes nuevos.
```

En 1 se lanza el stress test con 10 insultos y el script con la función stream() detecta que están llegando mensajes a SQS.

```
C:\Users\alain\OneDrive\Escritorio\SD labs\TASK2\EJ2>stress_test.py
Enviando 10 insultos a SQS...
Stress test finalizado.
```

<input type="checkbox"/>	2025/05/18/[\$LATEST]83b9d1b642a64ae1960c6d931dd133da	2025-05-18 23:18:20 (UTC+02:00)
<input type="checkbox"/>	2025/05/18/[\$LATEST]2b908a1755cf4631be6717dd7e5fdbb0	2025-05-18 23:18:18 (UTC+02:00)
<input type="checkbox"/>	2025/05/18/[\$LATEST]bbeaffff315d40d5abd97f89be6ebb12	2025-05-18 23:18:17 (UTC+02:00)
<input type="checkbox"/>	2025/05/18/[\$LATEST]6367f2666824467d8bf2281f13069669	2025-05-18 23:18:17 (UTC+02:00)

Se han lanzado 4 lambdas.

- 50 insultos:

```
C:\Users\alain\OneDrive\Escritorio\SD labs\TASK2\EJ2>stress_test.py
Enviando 50 insultos a SQS...
Stress test finalizado.
```

<input type="checkbox"/>	2025/05/18/[\$LATEST]54a751d6406e4f40995c02f85fd67009	2025-05-18 23:25:47 (UTC+02:00)
<input type="checkbox"/>	2025/05/18/[\$LATEST]4c3fc8ba144a4fef859420e983232844	2025-05-18 23:25:45 (UTC+02:00)
<input type="checkbox"/>	2025/05/18/[\$LATEST]f3517369fe2e4da0afb90931df0b234f	2025-05-18 23:24:38 (UTC+02:00)
<input type="checkbox"/>	2025/05/18/[\$LATEST]6367f2666824467d8bf2281f13069669	2025-05-18 23:24:36 (UTC+02:00)
<input type="checkbox"/>	2025/05/18/[\$LATEST]bbeaffff315d40d5abd97f89be6ebb12	2025-05-18 23:24:35 (UTC+02:00)
<input type="checkbox"/>	2025/05/18/[\$LATEST]2b908a1755cf4631be6717dd7e5fdbb0	2025-05-18 23:24:35 (UTC+02:00)
<input type="checkbox"/>	2025/05/18/[\$LATEST]83b9d1b642a64ae1960c6d931dd133da	2025-05-18 23:24:33 (UTC+02:00)

Se han lanzado 7 lambdas.

- 200 insultos:

```
C:\Users\alain\OneDrive\Escritorio\SD labs\TASK2\EJ2>stress_test.py
Enviando 200 insultos a SQS...
Stress test finalizado.
```

Flujos de registros (10)

[Eliminar](#)[Crear flujo de registros](#)[Buscar en](#)☐ Coincidencia exacta ☐ Mostrar caducado

<input type="checkbox"/> Secuencia de registro	▼ Hora del último evento
<input type="checkbox"/> 2025/05/18/[\$LATEST]d1f72a1bbcc141c8afadaa6af40e9e3a	2025-05-18 23:32:10 (UTC+02:00)
<input type="checkbox"/> 2025/05/18/[\$LATEST]a67d8b3943b54fbf829ea3290deced25	2025-05-18 23:32:10 (UTC+02:00)
<input type="checkbox"/> 2025/05/18/[\$LATEST]5eae3d11ddb4295a02f93ad976b3c52	2025-05-18 23:32:10 (UTC+02:00)
<input type="checkbox"/> 2025/05/18/[\$LATEST]f3517369fe2e4da0afb90931df0b234f	2025-05-18 23:32:08 (UTC+02:00)
<input type="checkbox"/> 2025/05/18/[\$LATEST]4c3fc8ba144a4fef859420e983232844	2025-05-18 23:32:07 (UTC+02:00)
<input type="checkbox"/> 2025/05/18/[\$LATEST]83b9d1b642a64ae1960c6d931dd133da	2025-05-18 23:31:53 (UTC+02:00)
<input type="checkbox"/> 2025/05/18/[\$LATEST]2b908a1755cf4631be6717dd7e5fdbb0	2025-05-18 23:30:57 (UTC+02:00)
<input type="checkbox"/> 2025/05/18/[\$LATEST]6367f2666824467d8bf2281f13069669	2025-05-18 23:30:56 (UTC+02:00)
<input type="checkbox"/> 2025/05/18/[\$LATEST]bbeaffff315d40d5abd97f89be6ebb12	2025-05-18 23:30:55 (UTC+02:00)
<input type="checkbox"/> 2025/05/18/[\$LATEST]54a751d6406e4f40995c02f85fd67009	2025-05-18 23:28:59 (UTC+02:00)

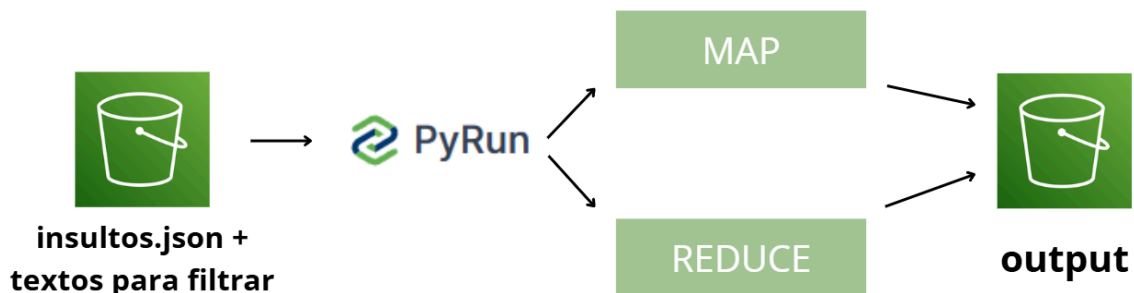
El detalle que voy a comentar a continuación es importante tenerlo en cuenta. Para permitir que el sistema de escalado detecte correctamente la carga en la cola, se introduce un `time.sleep(3)` al final de la función Lambda. Esto simula una carga de trabajo mínima y evita que las Lambdas finalicen antes de que el script que contiene `stream()` vuelva a consultar el tamaño de la cola. Sin esta pausa, las Lambdas terminan demasiado rápido, vaciando la cola, lo cual impide que se activen nuevas instancias. Con `time.sleep(3)`, el sistema tiene tiempo suficiente para observar que hay trabajo pendiente y escalar dinámicamente según `maxfunc`.

He hecho pruebas sin `time.sleep()` y en el mejor de los casos, para la misma cantidad de insultos que antes, se me han lanzado 2 lambdas.

EJERCICIO 3

En el ejercicio 3 había que filtrar insultos contenidos en una serie de textos almacenados en S3 y devolver el output con los textos filtrados, obteniendo también el número exacto de insultos procesados. Esto había que hacerlo utilizando Map - Reduce de Lithops.

Para que quede más claro, adjunto una imagen de cómo es la arquitectura:



Sistema de ficheros dentro de mi bucket de S3:

insultos.json → JSON con insultos.

input/ → carpeta que contiene "texto1.txt", "texto2.txt", etc... Cada fichero .txt tiene una frase que puede contener insultos o no.

output/ → carpeta en la que se guardarán los textos filtrados.

A continuación voy a mostrar cómo sería una ejecución:

Parto de 5 ficheros.txt en la carpeta input/:

<input type="checkbox"/>	texto1.txt	txt	19 May 2025 9:36:42 AM CEST	43.0 B	Estándar
<input type="checkbox"/>	texto2.txt	txt	19 May 2025 9:36:42 AM CEST	28.0 B	Estándar
<input type="checkbox"/>	texto3.txt	txt	19 May 2025 9:36:42 AM CEST	53.0 B	Estándar
<input type="checkbox"/>	texto4.txt	txt	19 May 2025 9:36:43 AM CEST	40.0 B	Estándar
<input type="checkbox"/>	texto5.txt	txt	19 May 2025 9:36:43 AM CEST	46.0 B	Estándar

Y de un fichero con insultos:

<input type="checkbox"/>	input/	Carpeta	-	-	-
<input type="checkbox"/>	insultos.json	json	19 May 2025 9:54:34 AM CEST	84.0 B	Estándar
<input type="checkbox"/>	output/	Carpeta	-	-	-

Después de tener S3 configurado, vamos a Pyrun.


En Pyrun añadimos el código a un nuevo workspace y ejecutamos. Resultado:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
2025-05-19 08:47:45,249 [DEBUG] monitor.py:282 -- ExecutorID 2a4df0-0 | RabbitMQ job monitor finished
2025-05-19 08:47:45,310 [DEBUG] future.py:229 -- ExecutorID 2a4df0-0 | JobID R000 - Got status from call 00000
ID: b56e88cf-b089-49cc-a7cd-5935dfd8202c - Time: 12.13 seconds
2025-05-19 08:47:45,310 [DEBUG] future.py:286 -- ExecutorID 2a4df0-0 | JobID R000 - Got output from call 00000
ID: b56e88cf-b089-49cc-a7cd-5935dfd8202c
2025-05-19 08:47:45,311 [INFO] executors.py:631 -- ExecutorID 2a4df0-0 - Cleaning temporary data
2025-05-19 08:47:45,314 [DEBUG] executors.py:532 -- ExecutorID 2a4df0-0 - Finished getting results

Total de insultos censurados: 11
```

En total ha censurado 11 insultos, lo cual está perfecto, ya que es el número de insultos que había puesto. En la carpeta output/, anteriormente vacía, ahora aparece lo siguiente:

<input type="checkbox"/>	 texto1.txt	txt	19 May 2025 10:47:32 AM CEST	44.0 B	Estándar
<input type="checkbox"/>	 texto2.txt	txt	19 May 2025 10:47:31 AM CEST	28.0 B	Estándar
<input type="checkbox"/>	 texto3.txt	txt	19 May 2025 10:47:31 AM CEST	51.0 B	Estándar
<input type="checkbox"/>	 texto4.txt	txt	19 May 2025 10:47:33 AM CEST	42.0 B	Estándar
<input type="checkbox"/>	 texto5.txt	txt	19 May 2025 10:47:31 AM CEST	46.0 B	Estándar

Esto es un ejemplo del “texto1.txt” en la carpeta output después de haber sido procesado.

