Contexto rápido

Cuando usás **Spring Cloud Stream** con funciones (`Function`, `Consumer`, `Supplier`), para conectarlas a **mensajes** que entran y salen, Spring las "envuelve" con **canales** o **bindings**. Esto permite que no se deba escribir programáticamente configuraciones relacionadas a un broker en particular, de forma que si el broker se cambia (por ejemplo de Kafka a RabbitMQ), el sistema sea independiente de ello.

Estos bindings tienen nombres como:

- `<funcion>-in-0` → el canal/stream/tema por donde **entra** el mensaje a la función
- `<funcion>-out-0` → el canal/stream/tema por donde **sale** el mensaje de la función

---

Por qué hay IN y OUT aunque sea una sola función

Porque la función puede:

- Recibir un mensaje (input) desde un topic/cola (por eso está el binding **in**)
- Enviar un mensaje procesado (output) a otro topic/cola (por eso está el binding **out**)

Por eso, aunque tengas **una sola función**, hay **dos canales** asociados:

- Entrada (`-in-0`) para recibir mensajes
- Salida (`-out-0`) para enviar mensajes procesados

Ejemplo sencillo con una función `sendRiderLocation`

```
@Bean public Function<RiderLocation, RiderLocation>
sendRiderLocation() {     return riderLocation -> {
// Procesar ubicación         return riderLocation;  //
devuelve el mismo objeto, o modificado     }; }
```

Esta función:

- **Recibe** un objeto `RiderLocation` desde Kafka (topic configurado en `sendRiderLocation-in-0`)
- **Devuelve** un objeto `RiderLocation` que Spring envía a Kafka (topic configurado en `sendRiderLocation-out-0`)

¿Qué pasa si tengo solo Consumer o Supplier?

- **Consumer**: solo tiene salida `-in-0` (porque solo consume mensajes)
- **Supplier**: solo tiene salida `-out-0` (porque solo produce mensajes)

Resumen:

Tipo de función	**Canales que tiene**
Function<T, R>	Entrada: `funcion-in-0`, Salida: `funcion-out-0`
Consumer<T>	Solo entrada: `funcion-in-0`

Tipo de función	Canales que tiene
Supplier<R>	Solo salida: `funcion-out-0`

En definitiva

- Que tengas **`sendRiderLocation-in-0`** y **`sendRiderLocation-out-0`** es porque tu función es un `Function` que recibe y devuelve algo.
- Spring necesita saber de dónde sacar el mensaje (input) y a dónde mandarlo (output).
- Por eso se configuran ambos bindings, aunque la función sea una sola.

Ejemplo con Spring Cloud Function

Código Java

```
@Bean public Function<String, String> uppercase() {
     return value -> {
       System.out.println("uppercase recibió: " + value);
     return value.toUpperCase();
     };
}

@Bean public Function<String, String> reverse() {
      return value -> {
            System.out.println("reverse recibió: " +
      value);
            return new
      StringBuilder(value).reverse().toString();
      };
}
```

---

1) Funciones independientes con ;

YAML:

```
spring:
 cloud:
  function:
   definition: uppercase;reverse
```

Cómo se usan:

- Con HTTP:
  - POST `/uppercase` con body `"hola"` → responde `"HOLA"`
  - POST `/reverse` con body `"hola"` → responde `"aloh"`
- Con Kafka:
  - Configurás bindings para cada función (ejemplo más abajo).
  - Mensajes que llegan a `input-topic-uppercase` van a `uppercase`.

- Mensajes que llegan a `input-topic-reverse` van a `reverse`.

YAML bindings Kafka ejemplo:

```
spring:
 cloud:
  stream:
   bindings:
    uppercase-in-0:
     destination: input-topic-uppercase
    uppercase-out-0:
     destination: output-topic-uppercase
    reverse-in-0:
     destination: input-topic-reverse
    reverse-out-0:
     destination: output-topic-reverse
```

---

2) Funciones encadenadas con |

YAML:

```
spring:
 cloud:
  function:
   definition: uppercase|reverse
```

Qué pasa:

- El mensaje entra a la función `uppercase`.
- El output de `uppercase` es input de `reverse`.
- El output final es lo que devuelve `reverse`.

Uso con HTTP:

- POST `/uppercase|reverse` con `"hola"` responde `"ALOH"` (primero uppercase → `HOLA`, luego reverse → `ALOH`).

Uso con Kafka:

- Solo necesitás configurar bindings para la función compuesta:

```
spring:
 cloud:
  stream:
   bindings:
    uppercase|reverse-in-0:
     destination: input-topic-composed
    uppercase|reverse-out-0:
     destination: output-topic-composed
```

- Los mensajes en `input-topic-composed` pasan por `uppercase` y luego por `reverse`.

- El resultado final va a `output-topic-composed`.

¿Qué es `StreamBridge`?

`StreamBridge` es una clase de Spring Cloud Stream que te permite **enviar mensajes de forma programática (manual)** a cualquier destino (Kafka, RabbitMQ, etc.) **sin necesidad de usar un `Supplier` o `Function` como bean**.

¿Para qué sirve?

- Te permite **enviar mensajes dinámicamente** a uno o varios **bindings** (temas, colas) desde cualquier parte de tu código.
- No necesitás definir funciones como `@Bean`.
- Es ideal cuando querés tener **control completo sobre el momento y contenido del mensaje**, como:
    - En una petición HTTP
    - Desde un listener de base de datos
    - Desde un cron job
    - Desde lógica de negocio