Big Queue

//Async Comm Service

IT BOARDING BOOTCAMP

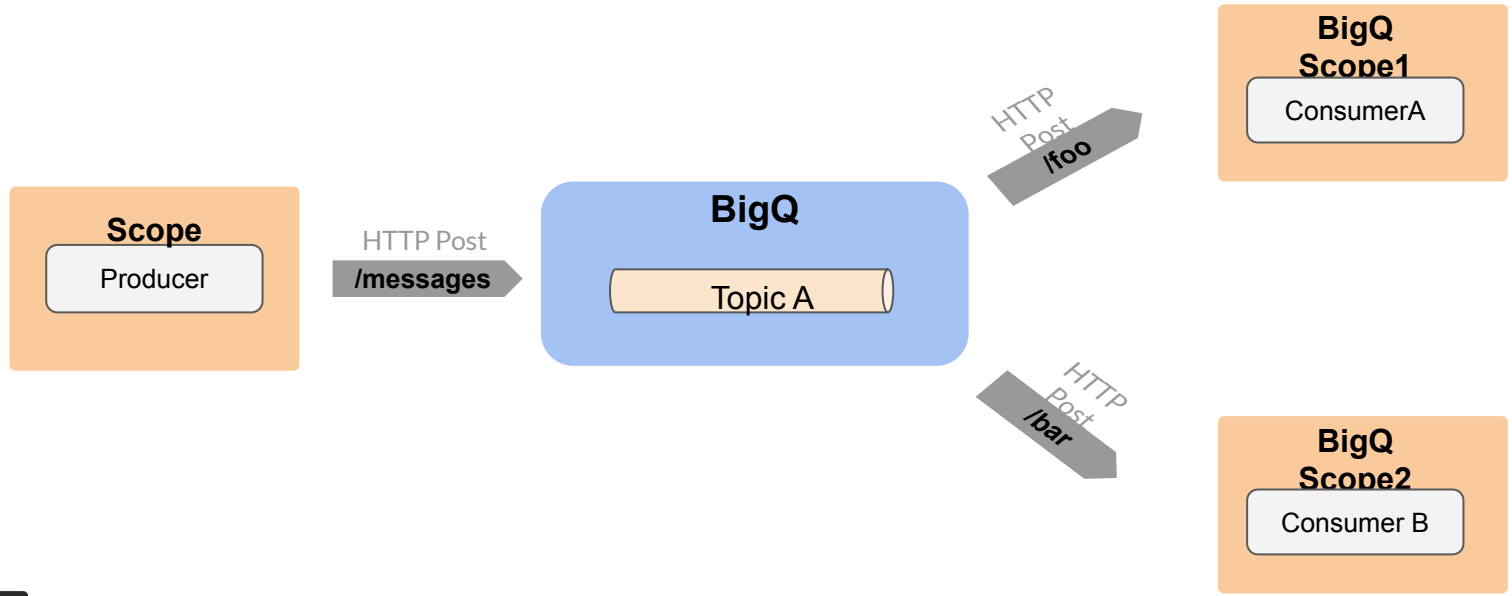
// Responsabilidad

Su responsabilidad es la de interconectar a diferentes aplicaciones en forma asincrónica, implementando el patrón publish-subscribe

Funcionamiento General

IT BOARDING
BOOTCAMP

// Producción y consumo de mensajes

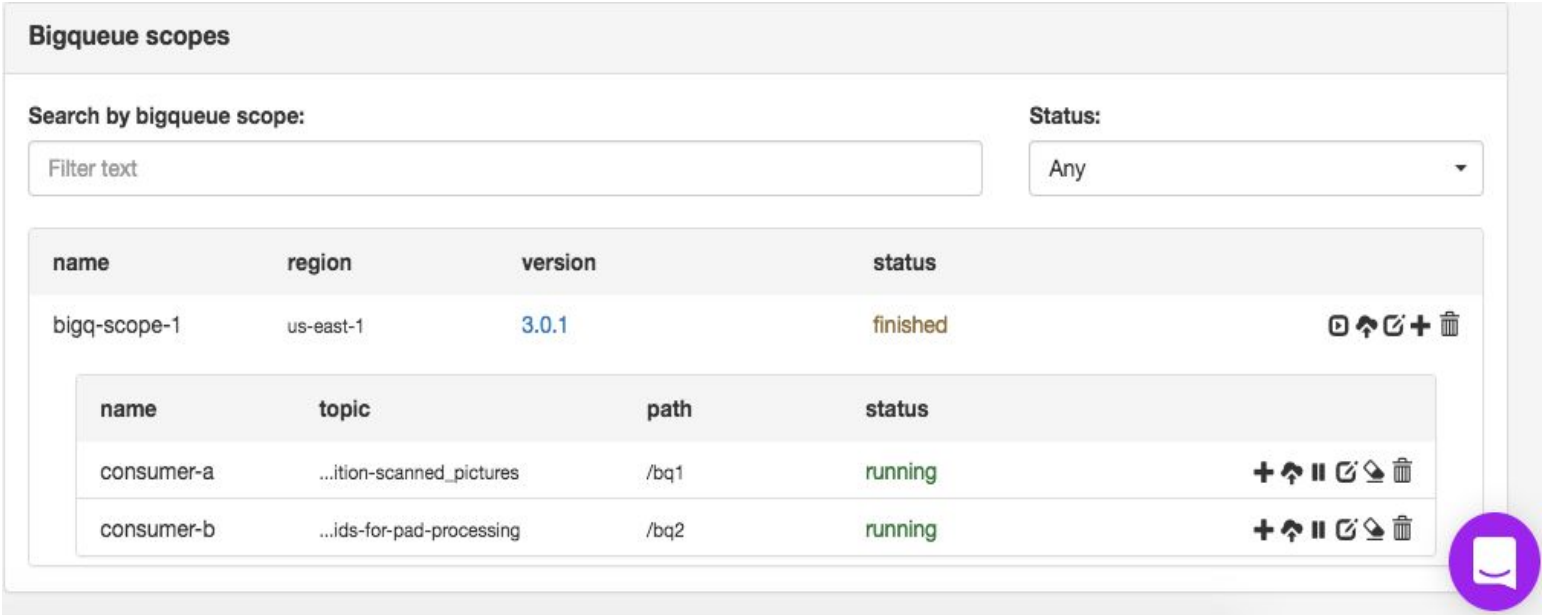


// Async Comm

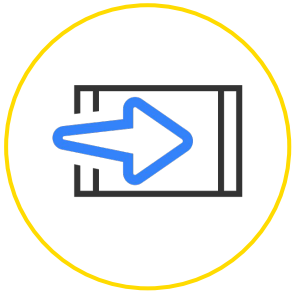
CUANDO ES POSITIVA?

- Fuente de novedades hacia múltiples interesados (conocidos o no)
- Desacople de request con capacidad de procesarlo
- Poder controlar picos de tráfico (buffer de olas de mensajes)
- Features extra independientes del producer (i.e. filtros)

// BQ Scope - Múltiples consumers



// Principales conceptos por grupo

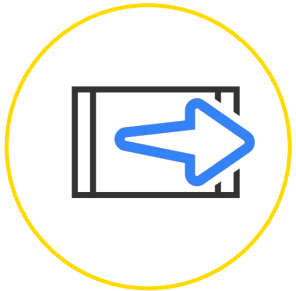


TOPICOS

Rate Limit (P)
Source
Tags
TTL
Filters (P)

MENSAJES

Filters
Size



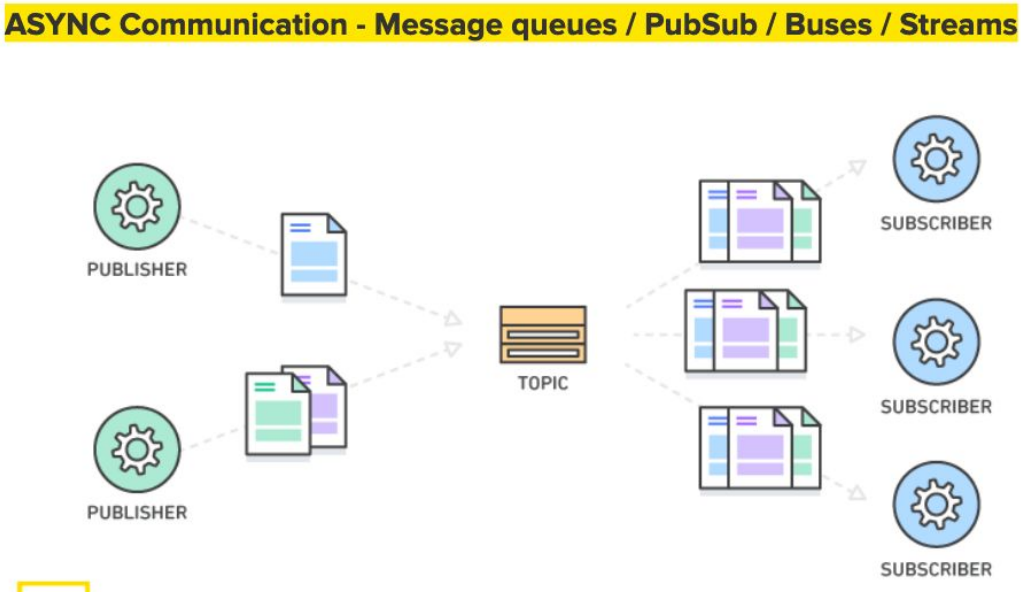
CONSUMERS

Rate Limit (C)
Reset / Rewind
Lag
Tags
Peek
Filters

// Otros conceptos

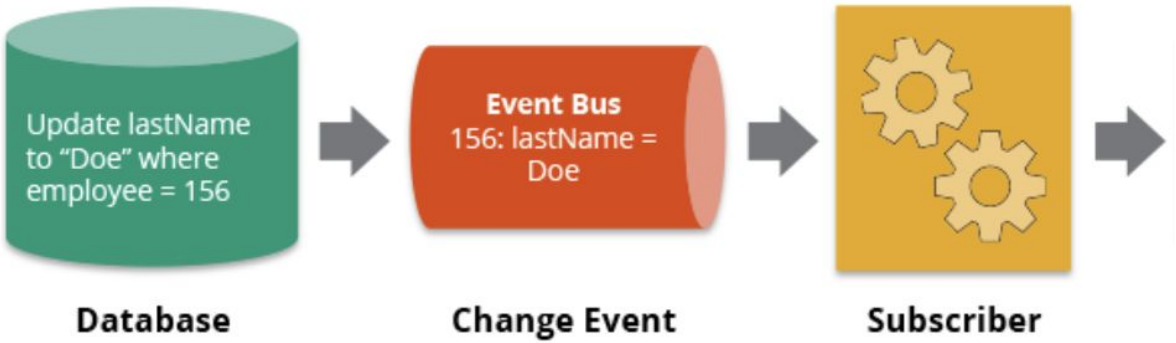
- Tópicos públicos o privados
- Puedo producir en tópicos de otra app (attach) con autorización del dueño
- Puedo crear consumers de cualquier tópico (público) o de los de mi app

// Casos de uso típicos



<Deja vu link>

// Casos de uso típicos



<[Deja vu link]>

// Garantías

- ~~Orden~~
- ~~Unicidad~~
- At least once delivery
- Retry until delivered
- Time to delivery (*) - SLO WIP

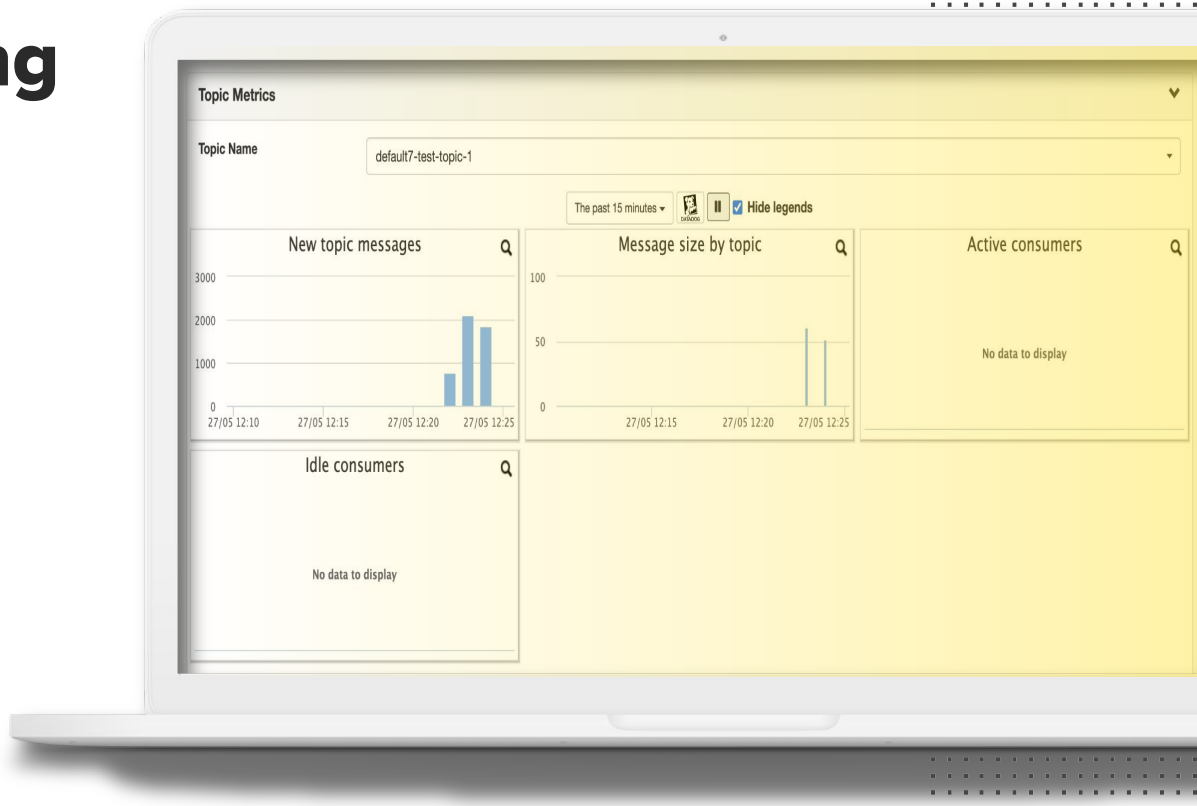
(*) Se mide para consumers “sanos” -sin errores-

// Troubleshooting

PRODUCCION

- Ante un problema ver tablero completo
- Ajustar timeouts acorde a “Produce time”

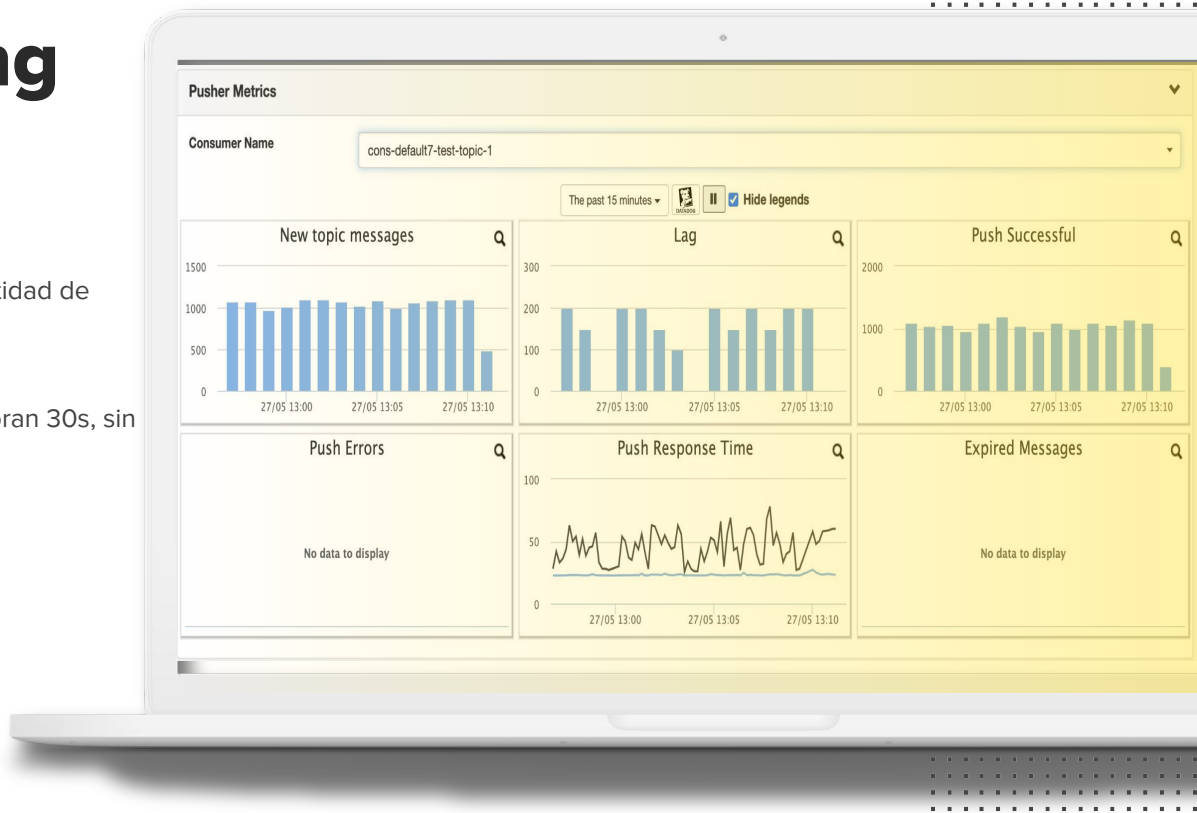
Fury - BigQ Topics - Topic Metrics



// Troubleshooting

CONSUMO

- Tiempos altos y errores pueden afectar la cantidad de mensajes que recibo
- Los reintentos no son exactos (al menos demoran 30s, sin garantías)



// Malas Prácticas

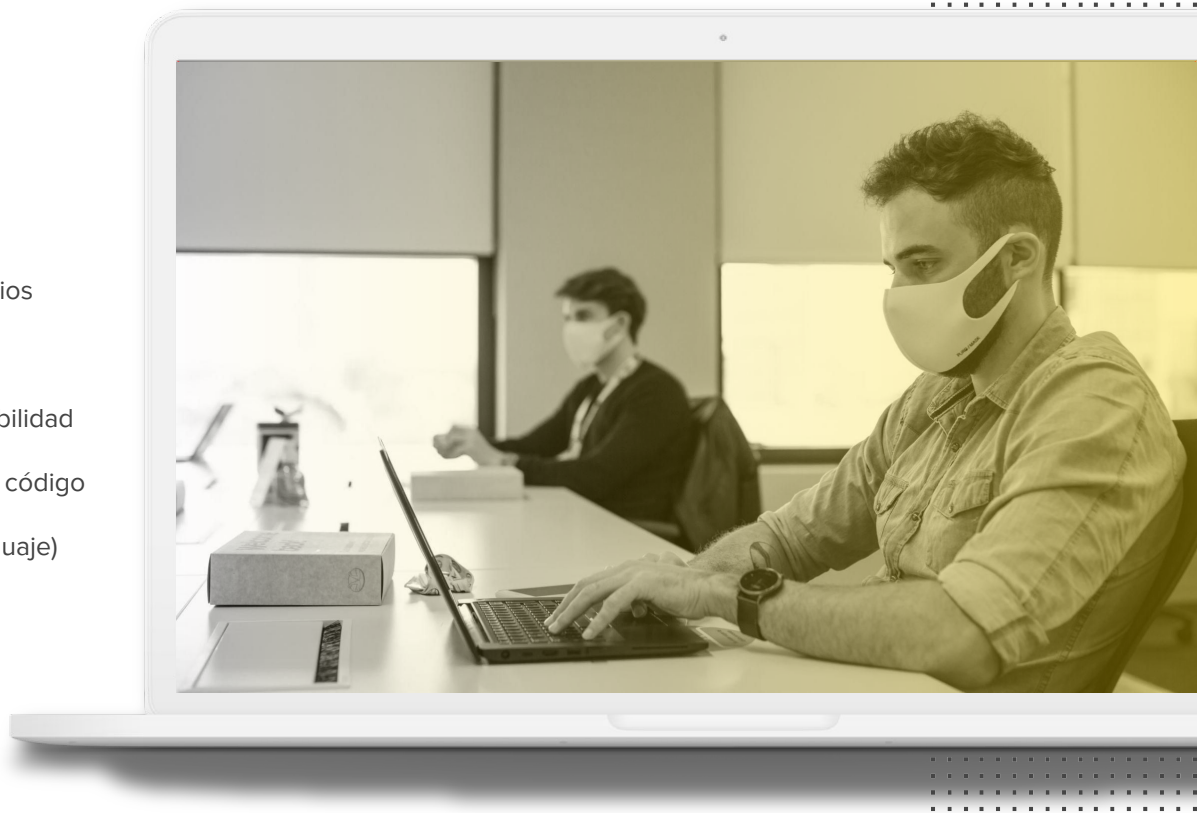
- Tópicos multi-propósito
- Mensajes muy grandes
- Push errors constantes
- Falta de monitoreo
- Sleeps / Retries en consumers
- Rate limits incorrectos
- Tópicos sin consumers
- No-Tagging
- Calendar queues para agendar reintentos
- Acoplarse a metadata del pusher

// Buenas Prácticas

- Bibliotecas actualizadas
- Artefactos de test marcados como test
- Mantenete al tanto / actualizado
- Bulk delivery
- Comunicate, reportá issues, preguntá
- Utiliza los canales correctos de soporte
- Service Modeling
- Documentación

// SDKs

- Oficiales para Java, Go y Python, con sus propios core-mantainers
- Garantizan trazabilidad, compliance y mantenibilidad
- Snippets disponibles con muy pocas líneas de código
- Versiones sync y async (dependiendo del lenguaje)



// Links Utiles

[Fury Docs]

[Services Site]

Detalles de implementación

<Repo de Referencia>

IT BOARDING
BOOTCAMP

// Producción de Mensajes (Java)

Sync

```
Producer producer = Producer
    .builder()
    .withTopic(topicName)
    .build();
...
...

producer.send(messageData);
```

Non-Blocking

```
Producer producer = Producer
    .builder()
    .withTopic(topicName)
    .build();
...
...

CompletableFuture<Object> result =
    producer.sendAsync(messageData);

... do something
```

// Consumo de Mensajes (Single - Java)

Controller

```java
@PostMapping("/consume-items-price-change/single")
public ResponseEntity<Boolean> processSingleUpdate(@RequestBody @Valid ConsumerMessageDTO dto) throws ParseException, RestException, KvsException, JsonExcepti
    Map<String,Object> feedUpdateData = dto.getMessage();
    String itemId = (String) feedUpdateData.get("id");
    itemsService.processFeedUpdate(itemId);
    return ResponseEntity.ok(true);
}
```

// Consumo de Mensajes (Single - Java)

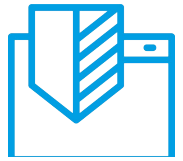
DTO

```java
public class ConsumerMessageDTO {
    @JsonProperty("id")
    private String id;

    @NotNull
    @JsonProperty("msg")
    private Map<String,Object> message;

    @NotNull
    @JsonProperty("publish_time")
    private Long publishTime;
}
```

// Consumo de Mensajes (Bulk - Java)



RECOMMENDED

Controller

```java
@PostMapping("/consume-items-price-change/bulk")
public ResponseEntity<BulkDeliveryResponseDTO> processBulkUpdate(@RequestBody @Valid BulkDeliveryMessagesDTO allMessages) throws ParseException, RestException,
    Collection<ConsumerMessageDTO> messages = allMessages.getMessages();
    Collection<BulkDeliveryResponseElementDTO> responses = messages.stream().map(dto -> {
        try {
            Map<String,Object> feedUpdateData = dto.getMessage();
            String itemId = (String) feedUpdateData.get("id");
            itemsService.processFeedUpdate(itemId);
            return new BulkDeliveryResponseElementDTO(dto.getId(),  code: 200);
        } catch (Exception e) {
            return new BulkDeliveryResponseElementDTO(dto.getId(),  code: 500);
        }
    }).collect(Collectors.toList());
    BulkDeliveryResponseDTO resp = new BulkDeliveryResponseDTO(responses);
    return ResponseEntity.ok(resp);
}
```

// Consumo de Mensajes (Bulk - Java)

DTOs

```java
public class BulkDeliveryMessagesDTO {
    @JsonProperty("messages")
    Collection<ConsumerMessageDTO> messages;
}
```

```java
public class BulkDeliveryResponseDTO {
    private Collection<BulkDeliveryResponseElementDTO> responses;
}
```

```java
public class BulkDeliveryResponseElementDTO {
    private String id;

    private Integer code;
}
```

Gracias.

IT BOARDING

BOOTCAMP