

# Tarea 2

Integrantes: Maximiliano Bustos  
                  Jonas Oviedo  
Profesor:      Nicolas Hidalgo

Fecha de entrega: 24 de Mayo de 2023  
                  Santiago de Chile

# Introducción

El presente informe trata del desarrollo de dos sistemas de intermediación de mensajes, siendo estos Kafka y RabbitMQ. Para ambos sistemas se crean dispositivos que son capaces de recibir mensajes llamados Consumers y dispositivos capaces de enviar mensajes llamados Producers. Para luego analizar ambos sistemas para compararlos en categorías como por ejemplo escalabilidad, también responder algunas preguntas tales como ¿Qué opciones ofrecen ambos sistemas para el almacenamiento y recuperación de mensajes?

# Desarrollo

Antes de empezar se debe tener en cuenta que se define una variable  $\Delta T$  la cual hace que los Producers envíen mensajes cada cierto tiempo y esta debe ser capaz de modificarse para cada dispositivo. Además, cada mensaje enviado debe ser un JSON y debe contener Timestamp que indica cuando fue enviado el mensaje y Values el cual es un texto que contiene información aleatoria y un largo de este debe ser variable.

Luego de haber definido las variables y tener en cuenta el formato de los mensajes, se procede a implementar los sistemas de Kafka y RabbitMQ y con los sistemas listos se desarrollan los siguientes casos:

## 1. Primer caso

El primer caso que se nos presenta trata de tener  $n$  Producers, donde  $n$  es un parámetro que puede variar y solo un Consumers, y después imprimir por terminal los mensajes recibidos.

Como se puede observar en la figura 1 y 2, la implementación de los sistemas se realizó correctamente y por tanto, se puede ver como un Consumer está recibiendo varios mensajes de distintos Producers para cada sistema.

```
shsnow@shsnow-Modern-14-B5M:~/Documents/CodeWatonesGPT/T2SD/Kafka$ node consumerKafka.cjs
[{"device_ID":4,"timestamp":1684969411019,"value":{"data":"t3UL"},"numeroMensaje":5}
 {"device_ID":0,"timestamp":1684969411019,"value":{"data":"EIUWvLQSUFB9B6o07soHVTGYx0gwouEV"},"numeroMensaje":3}
 {"device_ID":3,"timestamp":1684969411019,"value":{"data":"Alwqyo5jLfu12WirsqvHjfqK5FgZFVCDwDIuR2jHc0u2"},"numeroMensaje":2}
 {"device_ID":2,"timestamp":1684969411019,"value":{"data":"0MLJFnB8o0IKrgcAIDY3ol3jJdJ56hbVpMSHMa2ij0Zg0QkRTM6Md"},"numeroMensaje":4}
 {"device_ID":1,"timestamp":1684969411018,"value":{"data":"fk0xKLhvVrecJfdXlhEshKctiIFTXus9TgTidI7IqTWK25hiArbh4jIai63PH9e15D4SjxHxWL7mCQswWBeTUUKM05K"},"numeroMensaje":1}
```

Figura 1: Caso 1 Kafka.

```
shsnow@shsnow-Modern-14-B5M:~/Documents/CodeWatonesGPT/T2SD/rabbitMQ$ node consum
erRabbit.js
[*] Waiting for messages in hello. To exit press CTRL+C
[x] Received {"device": 5, "message_count": 5, "timestamp": 1684969306.8788424,
"value": "R9V02yM2rsCbh9ZD2GSoCffWlutwlTFPdiRJ0apdk27QcfjjDwd4UoCvj09rCjkY2HiqLJ
AthU40iUpM"}
[x] Received {"device": 4, "message_count": 4, "timestamp": 1684969306.8792894,
"value": "FGxJ5AD022PzCjzYtRV0dMxdR0NjWxdKMMVd7AQ8qIOx5Qokjyw1NJTQgQBhkt7v"}
[x] Received {"device": 3, "message_count": 3, "timestamp": 1684969306.8783534,
"value": "Yk3oWuepKZtp1n8IdEFnvbHGu7a74N2H9tKWG0HgkHxD9sc92CJfqYkvjhe7ka2qpUSm8B
8xZRF2jn6L0p2M7ufDsjbXE7"}
[x] Received {"device": 1, "message_count": 1, "timestamp": 1684969306.8812597,
"value": "7ctTf2TKEUvQkdzDbdwmxkjwDyLGHM2YB58cvIjqXVCsH8HfW44bJV6gu2"}
```

Figura 2: Caso 1 RabbitMQ.

## 2. Segundo caso

El segundo caso sería el inverso del primer caso, es decir, tiene un solo Producer y tienen que existir  $m$  Consumers, donde  $m$  se puede ir variando, para luego imprimir por terminal los mensajes recibidos.

Lo que se ve en las figuras 3 y 4 es como un solo Producer le está enviando mensajes a todos los Consumers conectados.

```
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/Kafka$ node producerKa
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/Kafka$ node consumerK
afka.js
{"device_ID":4,"timestamp":1684969776885,"value":{"data":"BhiHXOIo9t9T74wTTDnUb
FjmqX0osK2JrFx889nI3PKABIUeqb9Q9j*","numeroMensaje":48}
{"device_ID":2,"timestamp":1684969776885,"value":{"data":"6","numeroMensaje":49}
{"device_ID":3,"timestamp":1684969776885,"value":{"data":"dvbSlAK2lmVIoD1995q0w
cYRj7HTplpXevHuk6hR2c8P8tjc5RXJn0ribiaed1c9tUXEx6lge0GZp4ix6CdwxYw*","numer
oMensaje":47}
{"device_ID":1,"timestamp":1684969776885,"value":{"data":"7dnEz5vrgyyfPRobj4N1r
9KB7hNo7lvqvCA)bc3HFjmbuRAfecswXjghTnwulK7o144u7A4*","numeroMensaje":46}
{"device_ID":0,"timestamp":1684969776885,"value":{"data":"muFz5hWAggDUvFCkmCP
Mq0vNlw8IV1gTFKdIMgtv8j3Epqelpnzp60zwufqPMiWcXwN0rKB7aQDzrql1010n0Q8MK*",
"numeroMensaje":50}
{"device_ID":0,"timestamp":1684969781889,"value":{"data":"oxq3mDUkUwoksBHTCFrX
vOKNvN9ZTcaXB6t0iphJmtPVq54Gjb*","numeroMensaje":55}
{"device_ID":1,"timestamp":1684969781889,"value":{"data":"kyEE3yzd5M29xkRft0a4
KZLJr1xFZtFzYeWCgyv81H5YEcj00yE1yTAbaJ3*","numeroMensaje":51}
{"device_ID":2,"timestamp":1684969781889,"value":{"data":"jdxVBSpbPca0fCq*","nu
meroMensaje":53}
{"device_ID":2,"timestamp":1684969781889,"value":{"data":"kg6nV5LQ3ztpjjTrBfo6Tg
3YF81aBAEuVmnb7ykP00AJa3e*","numeroMensaje":54}
{"device_ID":3,"timestamp":1684969781889,"value":{"data":"U0zy4RH9KXqhqRVU30rjc
omPrso0xf40*","numeroMensaje":52}

shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/Kafka$ cd Documents/CodeWatonesGPT/T2SD/Kafka
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/Kafka$ node consumerK
afka.js
{"device_ID":2,"timestamp":1684969731854,"value":{"data":"2xEBgMSKCJEcdwVsX093Tz
5qPfeBx0B00W6*","numeroMensaje":4}
{"device_ID":4,"timestamp":1684969731854,"value":{"data":"oaHnAogkVuZYp8zKrgHE8
Yh0K03ustrya9PccXY*","numeroMensaje":3}
{"device_ID":0,"timestamp":1684969731854,"value":{"data":"dtAYL1f32U5NLpB0o8j8t1q
5*device_ID":3,"timestamp":1684969731854,"value":{"data":dtaYl1f32U5NLpB0o8j8t1q
5*device_ID":4,"timestamp":1684969731854,"value":{"data":dtaYl1f32U5NLpB0o8j8t1q
5*device_ID":0,"timestamp":1684969731854,"value":{"data":dtAYL1f32U5NLpB0o8j8t1q
5*device_ID":2,"timestamp":1684969731854,"value":{"data":dtAYL1f32U5NLpB0o8j8t1q
5*device_ID":3,"timestamp":1684969731854,"value":{"data":XAF*,"numeroMensaje":2}
*device_ID":1,"timestamp":1684969731854,"value":{"data":Ccqkig4L3JPVmEoDkbbug
*device_ID":1,"timestamp":1684969731854,"value":{"data":7tov4Zk0Akdb2Rd7a1K8dV1qjbySvryYU0k03qP5f75fw*,"numeroMensaje":1}
*device_ID":1,"timestamp":1684969731854,"value":{"data":Kn0CY64RjZFrqzhuctVT6
*device_ID":1,"timestamp":1684969731854,"value":{"data":N90t5fJv2asYymlr7LNZyflR*,"numeroMensaje":6}
*device_ID":2,"timestamp":1684969731854,"value":{"data":ERTqo*,"numeroMensaje
*:9}
*device_ID":1,"timestamp":1684969731854,"value":{"data":v5ep5mbxE19bqe*,"num
eroMensaje":10}
*device_ID":0,"timestamp":1684969731854,"value":{"data":3AiM9*,"numeroMensaje
*:7}
*device_ID":0,"timestamp":1684969731854,"value":{"data":VWaLrXjg3nGxEMiAgZweh
*device_ID":0,"timestamp":1684969731854,"value":{"data":R5TvbhrhBcLA1h0fli3754vilt
dXPcE2sgKMsf39Ae60h*,"numeroMensaje":8}
```

Figura 3: Caso 2 Kafka.

The figure displays three terminal windows from a Linux system (shsnow@shsnow-Modern-14-B5M) illustrating a RabbitMQ setup. The left window shows a producer (producerRabbit.py) sending messages to a queue. The middle and right windows show consumers (node consumerRabbit.js) receiving these messages. The logs in the windows show timestamped messages being sent and received, demonstrating the interaction between the different components.

```

shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/rabbitMQ$ python3 producerRabbit.py
Device 1 sent Message 1: {"device": 1, "message_count": 1, "timestamp": 1684970114.0213923, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}
Device 4 sent Message 4: {"device": 4, "message_count": 4, "timestamp": 1684970114.0467825, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}
Device 5 sent Message 5: {"device": 5, "message_count": 5, "timestamp": 1684970114.0464804, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}
Device 2 sent Message 2: {"device": 2, "message_count": 2, "timestamp": 1684970114.0228138, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}
Device 3 sent Message 3: {"device": 3, "message_count": 3, "timestamp": 1684970114.0269458, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}
Device 1 sent Message 6: {"device": 1, "message_count": 6, "timestamp": 1684970114.0440109, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}
Device 4 sent Message 9: {"device": 4, "message_count": 9, "timestamp": 1684970114.0483118, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}
Device 3 sent Message 8: {"device": 3, "message_count": 8, "timestamp": 1684970114.0464804, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}
Device 5 sent Message 10: {"device": 5, "message_count": 10, "timestamp": 1684970114.0464804, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/rabbitMQ$ node consumerRabbit.js
[*] Waiting for messages in hello. To exit press CTRL+C
[x] Received {"device": 1, "message_count": 1, "timestamp": 1684970114.0213923, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}
[x] Received {"device": 2, "message_count": 2, "timestamp": 1684970114.0467825, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}
[x] Received {"device": 3, "message_count": 3, "timestamp": 1684970114.0269458, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}
[x] Received {"device": 4, "message_count": 4, "timestamp": 1684970114.0464804, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}
[x] Received {"device": 5, "message_count": 5, "timestamp": 1684970114.0464804, "value": "z16NLGvnj77czy30trAcylPrnsbBQcb56j03CddL6Dlp0"}]]
```

Figura 4: Caso 2 RabbitMQ.

### 3. Tercer caso

El tercer caso se trata de tener varios Producers y Consumers, para luego analizar la escalabilidad, tolerancia a fallos, latencia y rendimiento, persistencia de mensajes y facilidad de uso y administración.

Ahora analizaremos las categorías anteriormente mencionadas

#### Escalabilidad

¿Cómo manejan Kafka y RabbitMQ el crecimiento en el número de productores (dispositivos) y consumidores de datos?

Para responder la pregunta primero nos ponemos en distintos casos con distintas cantidades de Producers y dos Consumers en ambos sistemas. El primer caso se realiza con 5 Producers y el segundo caso con 10.

Como se ve en la figura 5 se están realizando los mensajes y luego se toma el primer mensaje y el último de un solo Consumer, los cuales se muestran en las figuras 6 y 7.

The screenshot shows two terminal windows side-by-side. Both windows have a blue header bar with the text 'shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/Kafka\$'. The left window displays the command 'node producerKafka.js' followed by several JSON message logs. The right window displays the command 'node consumerKafka.js' followed by similar JSON message logs. The messages are identical in both windows, indicating successful communication between producers and consumers.

```

shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/Kafka$ node producerKafka.js
{"device_ID":2,"timestamp":1684979851483,"value":{"data":"BW0l5NsF0v3CuJAw7DbUseQNT0hAyIPNISF3FODNQuv2niLk6XG8b6QCUzhu9Juinbf7eU50jYgjgvi"}, "numeroMensaje":1046}el timestamp del receptor es: 1684979851992
{"device_ID":1,"timestamp":1684979851483,"value":{"data":"35"}, "numeroMensaje":1049}el timestamp del receptor es: 1684979851992
{"device_ID":0,"timestamp":1684979851483,"value":{"data":"WKhIIsn7rj24e5rs3XecRcf09Mqr6lrlqf0h0hGKrLgaba9MlT0U6W6KI"}, "numeroMensaje":1048}el timestamp del receptor es: 1684979851992
{"device_ID":4,"timestamp":1684979851483,"value":{"data":"JetbsJ009jfP9KqLLWCEBj8Eme5Cvbj"}, "numeroMensaje":1050}el timestamp del receptor es: 1684979851992
{"device_ID":0,"timestamp":1684979852485,"value":{"data":"vWSBDLmWb98em740vb9ZLA1LQH0l0Wwv6ZigNpmlobS"}, "numeroMensaje":1053}el timestamp del receptor es: 1684979852493
{"device_ID":3,"timestamp":1684979852485,"value":{"data":"f6c9wm7UpXqLaJl6CE6VUKkLJH702jG50ltDWrXc"}, "numeroMensaje":1052}el timestamp del receptor es: 1684979852493
{"device_ID":2,"timestamp":1684979852484,"value":{"data":"ayZp"}, "numeroMensaje":1051}el timestamp del receptor es: 1684979852493
{"device_ID":1,"timestamp":1684979852485,"value":{"data":"ZHMOsnoQUTEtdy5UR2PegjfA57nP94Y7WNgpl86x0PF6vaVfmMvNzpV616PlzVce"}, "numeroMensaje":1054}el timestamp del receptor es: 1684979852503
{"device_ID":4,"timestamp":1684979852485,"value":{"data":"Rghxy"}, "numeroMensaje":1055}el timestamp del receptor es: 1684979852503

shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/Kafka$ node consumerKafka.js
{"device_ID":3,"timestamp":1684979629579,"value":{"data":"2gYgkvhfrsmx7cG0pEqISRkef0ZVMld8wdtD8czUsjfxtieZlxb95MFw9u5qhzbQjxXNP1Nb"}, "numeroMensaje":60}el timestamp del receptor es: 1684979712700
{"device_ID":2,"timestamp":1684979629579,"value":{"data":"HUdMzyveAyeGeob5Nm6kY8qBsp6VmJleltnJZ0tK755e01TKp3DAhmo8Xj7xiHdijCr1MS2ll1TdF569J67BaZfoutJplRT"}, "numeroMensaje":57}el timestamp del receptor es: 1684979712705
{"device_ID":1,"timestamp":1684979629579,"value":{"data":"Rnf23Lq6r7ZWtxqn9GrQotkTS0wHA7R18UT5XUHMs5ZInJETZTVbA9E49A4BfowCDDG2I50DMtjc8Ly8cZR1JRtg6nMS1St"}, "numeroMensaje":58}el timestamp del receptor es: 1684979712706
{"device_ID":0,"timestamp":1684979629579,"value":{"data":"cijkl7Nq05758WMZxrwX6XSNKA6tWm"}, "numeroMensaje":56}el timestamp del receptor es: 1684979712706
{"device_ID":3,"timestamp":1684979629579,"value":{"data":"39Pt8d8kgZ100UnTkWb1B5ADmbt75jUgnZns77pqv4"}, "numeroMensaje":59}el timestamp del receptor es: 1684979712706
{"device_ID":4,"timestamp":1684979630579,"value":{"data":"qkUjbsypjmdtcmayNur2U0jm814Rj02k4V1D0cIsWAhajBieP0v0w55dpngnwTc3P1osvw7e55gehqbdb"}, "numeroMensaje":65}el timestamp del receptor es: 1684979712706

```

Figura 5: Realización mensajes con 5 Producers Kafka.

```
{"device_ID":1,"timestamp":1684979629579,"value":{"data":"Rnf23Lq6r7ZWtxqn9GrQotkTS0wHA7R18UT5XUHMs5ZInJETZTVbA9E49A4BfowCDDG2I50DMtjc8Ly8cZR1JRtg6nMS1St"}, "numeroMensaje":58}el timestamp del receptor es: 1684979712706
```

Figura 6: Primer mensaje con 5 Producers Kafka.

```
{"device_ID":1,"timestamp":1684979785420,"value":{"data":"82DYtVao0FhR7SVq88PrhcHG16K2sos21Np"}, "numeroMensaje":719}el timestamp del receptor es: 1684979785429
```

Figura 7: Último mensaje con 5 Producers Kafka.

Ahora se repite lo de arriba, pero con 10 Producers.

```

shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/Kafka$ node producer
{"device_ID":4,"timestamp":1684980336820,"value":{"data":"C3JDW2zsGY64frjxKNoYHR6wqs0p6jhm7HEKcqlu80NEFH58LoyF8tbl3zir4rkf5ImvFhs301f"}, "numeroMensaje":887}el timestamp del receptor es: 1684980336820
{"device_ID":3,"timestamp":1684980336819,"value":{"data":"ULDlzcG13ddasMmqfcySkEKIBV1ezWc8"}, "numeroMensaje":881}el timestamp del receptor es: 1684980336828
{"device_ID":6,"timestamp":1684980336820,"value":{"data":"AgvnQ800cPLCoogh8L0NUFSwsUR9T5go2gfhUu5h7auUrCvGmPPQg4UALn97hZqkgo"}, "numeroMensaje":885}el timestamp del receptor es: 1684980336828
{"device_ID":8,"timestamp":1684980336820,"value":{"data":"EWgbitzalm80zUsXL9u8a6VPG784DRhSaflc1FzNJTfBmBAnA3"}, "numeroMensaje":884}el timestamp del receptor es: 1684980336828
{"device_ID":2,"timestamp":1684980336820,"value":{"data":"tKobJ2Y19jRN30K9Jay2FR3mp2qmK"}, "numeroMensaje":883}el timestamp del receptor es: 1684980336828
{"device_ID":7,"timestamp":1684980336820,"value":{"data":"sq3B0o0j1CtyR44KcwXznmaswrlTyjsLwLNonfSuWrYB"}, "numeroMensaje":888}el timestamp del receptor es: 1684980336828
{"device_ID":8,"timestamp":1684980336821,"value":{"data":"5Kdf"}, "numeroMensaje":890}el timestamp del receptor es: 1684980336828
{"device_ID":9,"timestamp":1684980336821,"value":{"data":"4k0KI5feNYMhCAM6rf3PXz9q9tvmuIN4xR9ojWjh8raIfPEJvhP9iKjIJjp6RCJthf57W4zF9XfsiZov4zp9LWQ44AWNy"}, "numeroMensaje":889}el timestamp del receptor es: 1684980336828
```
shsnow@shsnow-Modern-14-B5M: ~$ con 10 productores y 2 consumidores kafka
```
clK0ftk6kxqitMutob532UfaCrBdXhgk13z1CnnRp5test6XYBz7D0oQpjXndr9EWNN1w1"}, "numeroMensaje":1564}el timestamp del receptor es: 1684980405399
{"device_ID":8,"timestamp":1684980404892,"value":{"data":"BYTThvxq7NHfkuuW0EAg7r7Cs0iixBN0JvEZJ1WpJCLIDtczvJQUNOLWTAIIahWhH5k4Zp3y8"}, "numeroMensaje":1570}el timestamp del receptor es: 1684980405399
{"device_ID":4,"timestamp":1684980404891,"value":{"data":"vobK0g2fjq9xtCiXjWibcffff6055I9niuz3gsxe0fbanZhEhAy6n"}, "numeroMensaje":1566}el timestamp del receptor es: 1684980405400
{"device_ID":9,"timestamp":1684980404891,"value":{"data":"wXqzDy54WMjULrT1ulfFa9REllbo1Dko0K7lvM154a18AOUxmcbNv80pbnDu5S"}, "numeroMensaje":1569}el timestamp del receptor es: 1684980405400
{"device_ID":5,"timestamp":1684980404891,"value":{"data":"Pjnq9sUijOC2qGm53YaJDMQP07fL2Rmr"}, "numeroMensaje":1567}el timestamp del receptor es: 1684980405400
{"device_ID":6,"timestamp":1684980404891,"value":{"data":"Y2Htidis9s40wqUllCgL52KjW9bw8VYp2"}, "numeroMensaje":1565}el timestamp del receptor es: 1684980405400
{"device_ID":1,"timestamp":1684980404891,"value":{"data":"CHKTNmwyYIdg62YgrTfSpNJHTDfYhIK0o0jY8GHPaaevbdv5064RjEx4NKz470SpzntUSDStM39fIxq4HzGp3s"}, "numeroMensaje":1562}el timestamp del receptor es: 1684980405400
{"device_ID":7,"timestamp":1684980404891,"value":{"data":"2D5syG2EV208Ch2FkbIvJ0RkhmV2BYMKR75f0f0g7hLNj1xM4P8KGcaeIbgppVfjuP6ZV46gnC4l3"}, "numeroMensaje":1568}el timestamp del receptor es: 1684980405400
{"device_ID":3,"timestamp":1684980405892,"value":{"data":"5sSxjBxH"}, "numeroMensaje":1571}el timestamp del receptor es: 1684980405901
{"device_ID":4,"timestamp":1684980405893,"value":{"data":"mlnRd"}, "numeroMensaje":1572}el timestamp del receptor es: 1684980405902
```
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/Kafka$ 

```

Figura 8: Realización mensajes con 10 Producers Kafka.

```
{"device_ID":2,"timestamp":1684980248731,"value":{"data":"rlgXb66ydWszaiv8C0BKQpyCB"}, "numeroMensaje":3}el timestamp del receptor es: 1684980272707
```

Figura 9: Primer mensaje con 10 Producers Kafka.

```
{"device_ID":9,"timestamp":1684980336821,"value":{"data":"4k0KI5feNYMhCAM6rf3PXz9q9tvmuIN4xR9ojWjh8raIfPEJvhP9iKjIJjp6RCJthf57W4zF9XfsiZov4zp9LWQ44AWNy"}, "numeroMensaje":889}el timestamp del receptor es: 1684980336828
```

Figura 10: Último mensaje con 10 Producers Kafka.

Con los datos obtenidos de las pruebas en Kafka se puede ver la siguiente tabla:

|                | 5 Producers   |               | 10 Producers  |               |
|----------------|---------------|---------------|---------------|---------------|
| Mensaje        | Primero       | Último        | Primero       | Último        |
| tiempo inicial | 1684979629579 | 1684979785420 | 1684980248731 | 1684980336821 |
| tiempo final   | 1684979712706 | 1684979785429 | 1684980272707 | 1684980336828 |
| diferencia     | 83127         | 9             | 23976         | 7             |

Y por último, todo el proceso que se realizó en Kafka se realiza en RabbitMQ.

The image shows two terminal windows on a Linux system. The top window is titled 'shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWato...' and the bottom window is titled 'shsnow@shsnow-Modern-14-B5M: ~'. Both windows are running a Python script to interact with a RabbitMQ broker.

```

shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWato... python3 producer.py
[...] Received {"device": 3, "message_count": 3, "timestamp": 1684980794.207630
shsnow@shsnow-Modern-14-B5M: ~$ con 5 productores y 2 consumidores rabbitmq
[...] Received {"device": 3, "message_count": 253, "timestamp": 1684980796.221209

```

The top window displays the output of a producer script, showing messages being sent from multiple devices. The bottom window displays the output of a consumer script, showing messages being received by two consumers. The messages contain device IDs, message counts, and timestamps.

Figura 11: Realización de mensajes con 5 Producers Rabbit.

```
[x] Received {"device": 3, "message_count": 3, "timestamp": 1684980744.8433442,
  "value": "Qq3knvdHhhySBGGv0LlXQEkwgx3M9JVIIsEbGySN1FhgSjccw"}el timestamp del receptor es: 16849807448494
```

Figura 12: Primer mensaje con 5 Producers Rabbit.

```
[x] Received {"device": 3, "message_count": 253, "timestamp": 1684980796.262148
  , "value": "1xEF463AAuiZWNj1HqEYUu5Yf3yLedkXFGQRah2Fe68ev8c1KsxWB04jvpwm9J4feZ
  \l4Wi5hY224Is"}el timestamp del receptor es: 1684980796266
```

Figura 13: Último mensaje con 5 Producers Rabbit.

Figura 14: Realización mensajes con 10 Producers Rabbit.

```
[x] Received {"device": 1, "message_count": 1, "timestamp": 1684981022.5233824, "value": "DN9uOLDFVHaVcmZe9uvX6uuoxm4Xsn2VA8S9ACopvvyNltrleS9Nd15HAC8kYGvldFzxWkTCN55h0NfRZCCauHt1"} el timestamp del receptor es: 1684981022532
```

Figura 15: Primer mensaje con 10 Producers Rabbit.

```
[x] Received {"device": 7, "message_count": 427, "timestamp": 1684981067.0735896, "value": "wDIgCDDgYgGS8nac"} el timestamp del receptor es: 1684981067077
```

Figura 16: Último mensaje con 10 Producers Rabbit.

Con los datos obtenidos de las pruebas en RabbitMQ se puede ver la siguiente tabla:

|                | 5 Producers   |               | 10 Producers  |               |
|----------------|---------------|---------------|---------------|---------------|
| Mensaje        | primero       | último        | primero       | último        |
| tiempo inicial | 1684980744843 | 1684980796262 | 1684981022523 | 1684981067073 |
| tiempo final   | 1684980744849 | 1684980796266 | 1684981022532 | 1684981067077 |
| diferencia     | 6             | 4             | 9             | 4             |

Teniendo en cuenta los resultados anteriores, podemos decir que Kafka mejora su escalabilidad con el tiempo, en cambio con RabbitMQ este siempre tiene unos tiempos bajos con respecto a Kafka.

## Tolerancia a fallos

¿Qué mecanismos ofrecen Kafka y RabbitMQ para garantizar la disponibilidad y la recuperación ante errores? Entre Kafka y RabbitMQ ¿Cuál entrega mejores herramientas para el manejo de errores? ¿De qué depende?

Para dar respuestas a las preguntas planteadas, nos pusimos en la situación de detener el docker el cual mantiene funcionando el sistema para observar qué pasaba.

Para el sistema de Kafka, luego de detener el docker el programa se congela y al momento de levantarla nuevamente se actualizan todos los tópicos de golpe. Como se puede ver en la figura 17 y lo que se observa en la figura 18 es el mensaje que muestra la terminal del Producer/Consumer con el docker caído.

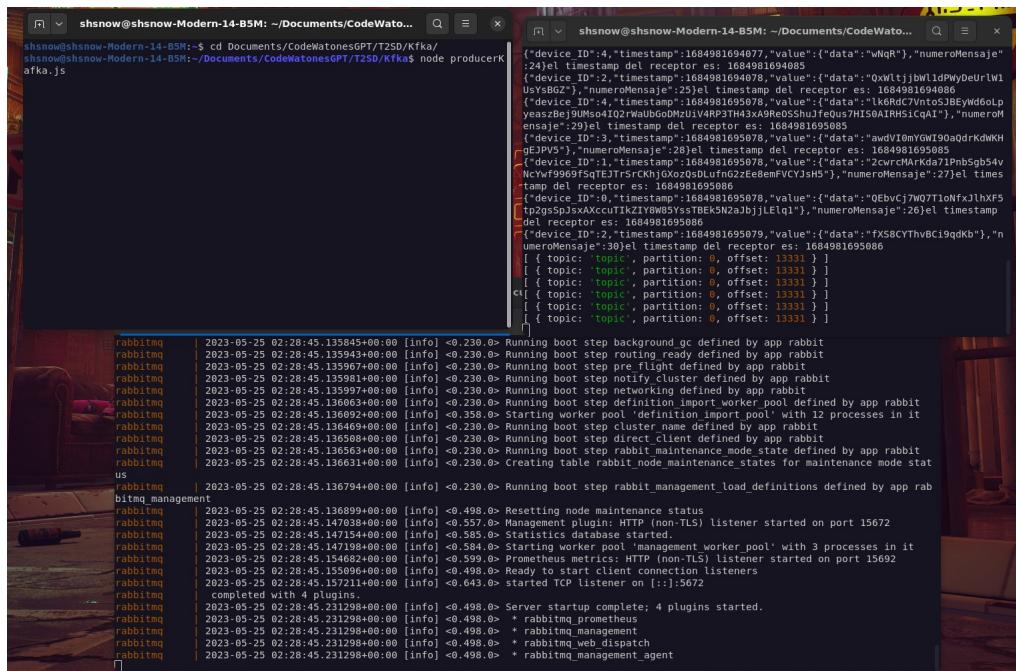


Figura 17: Botar docker Kafka.

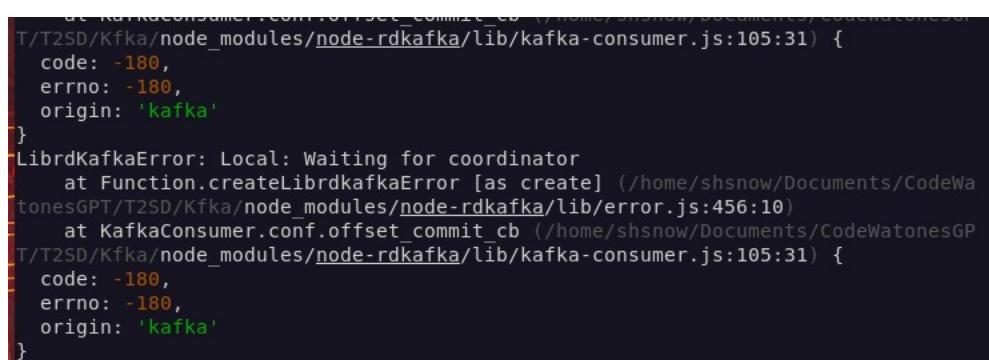


Figura 18: Mensaje fallo kafka.

Para RabbitMQ al detener el docker el Consumer se cae totalmente y el Producer entra en un estado de espera, en el instante en que se levanta el docker de nuevo el Producer vuelve a la normalidad mientras que el Consumer no, para este dispositivo vuelva a la normalidad tendría que ejecutarse el código nuevamente, como se puede observar en las figuras 19 y 20.

```

shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWat... Q = x shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWat...
raise self._reap_last_connection_workflow_error(error)
connection = pika.BlockingConnection(pika.ConnectionParameters('localhost')) 7149
File "/home/shsnow/.local/lib/python3.10/site-packages/pika/adapters/blocking_
connection.py", line 366, in __init__
    connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
File "/home/shsnow/.local/lib/python3.10/site-packages/pika/adapters/blocking_
connection.py", line 366, in __init__
    self._impl = self._create_connection(parameters, _impl_class)
ika.exceptions.AMQPConnectionError
self._impl = self._create_connection(parameters, _impl_class)
File "/home/shsnow/.local/lib/python3.10/site-packages/pika/adapters/blocking_
connection.py", line 451, in _create_connection
File "/home/shsnow/.local/lib/python3.10/site-packages/pika/adapters/blocking_
connection.py", line 451, in _create_connection
    self._impl = self._create_connection(parameters, _impl_class)
File "/home/shsnow/.local/lib/python3.10/site-packages/pika/adapters/blocking_
connection.py", line 451, in _create_connection
    raise self._reap_last_connection_workflow_error(error)
ika.exceptions.AMQPConnectionError
raise self._reap_last_connection_workflow_error(error)
ika.exceptions.AMQPConnectionError
raise self._reap_last_connection_workflow_error(error)
ika.exceptions.AMQPConnectionError
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/rabbitMQs

```

Figura 19: Botar docker RabbitMQ.

```

shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWat... Q = x shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWat...
KufkcUc1KSHoB89NEzbD1ffpdquE7tYjf115GxeS}
device 1 sent Message 351: {"device": 1, "message_count": 351, "timestamp": 1684 7149
82162.1192887, "value": "uhUJgONNlS"}
device 4 sent Message 354: {"device": 4, "message_count": 354, "timestamp": 1684 7149
82162.1228487, "value": "vl704vdkn0igEKLT8FaWreDSgpFwWzsXibk3YwIcKH8P6cmhjx7zi
xfdf9"}
device 3 sent Message 353: {"device": 3, "message_count": 353, "timestamp": 1684 7149
82162.122394, "value": "19fqxFJC15Rw3GftfMN"}
device 2 sent Message 352: {"device": 2, "message_count": 352, "timestamp": 1684 7149
82162.1218681, "value": "o4rb0v5kb0cGaxoJojfn6ukMMy7610"}
device 5 sent Message 360: {"device": 5, "message_count": 360, "timestamp": 1684 7149
82163.1475422, "value": "PalAldvUF"}
device 1 sent Message 356: {"device": 1, "message_count": 356, "timestamp": 1684 7149
82163.1496475, "value": "nkT2o0508IHnlp4WPAySEpeoZ044KF4msCsAa"}
device 2 sent Message 357: {"device": 2, "message_count": 357, "timestamp": 1684 7149
82163.1515126, "value": "VLJXYj0OKyHISJB0l7ubwy15mFLeq91655r088aBnX4vgAFWxpH
233v2RReyyiyQnYq71JYvD"}
device 3 sent Message 358: {"device": 3, "message_count": 358, "timestamp": 1684 7149
82163.150828, "value": "WKV2RV7Ig3zdPxPcpVQd9yIhpjdROMNBcLHGvgrmbfdWnQV45inXP
Vf9nE"}
device 4 sent Message 359: {"device": 4, "message_count": 359, "timestamp": 1684 7149
82163.1540844, "value": "Q2P9BPVv1XqAheD7XYv1pt8KNdBV3jgjEW4KEM4V2UBCfxHTGbVt
rd0"}
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/rabbitMQs

```

Figura 20: Levantar docker RabbitMQ.

Por lo que luego de este análisis, se concluye que el sistema de Kafka tiene una mejor tolerancia a fallos que RabbitMQ porque permite que el envío y recepción de mensajes continúe una vez levantado el sistema nuevamente cosa que RabbitMQ no pudo realizar.

## Latencia y rendimiento

¿Cómo se comportan Kafka y RabbitMQ en términos de latencia y rendimiento en un escenario de alta carga y tráfico de mensajes?

En el caso de Kafka, este sistema la conexión se establece más tarde que lo normal, pero una vez establecida la entrada de mensajes es casi inmediata, una vez detenido el proceso de enviar mensajes, el Consumer deja de imprimir por terminal los mensajes recibidos por lo que se puede concluir que no quedan mensajes en la cola. Cabe destacar que para este caso la variable  $\Delta T$  se dejó en 0.

Figura 21: Latencia Kafka.

Para el caso de RabbitMQ este sistema tiene una conexión inmediata al igual que la recepción de mensajes a pesar de haber definido la variable  $\Delta T$  en 100 milisegundos. Después de finalizar el proceso de envíos de mensajes, el Consumer muestra todos los mensajes por terminal concluyendo que no resta información en la cola.

```
[+] shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWato... = x shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWato... = x
device 3 sent Message 773: {"device": 3, "message_count": 773, "timestamp": 168483530.854560
83530.8514032, "value": "LFWmBFoUyTh"} [x] Received {"device": 1, "message_count": 771, "timestamp": 1684983530.854560
6, "value": "BEOqWRl7"}el timestamp del receptor es: 1684983530856
device 5 sent Message 775: {"device": 5, "message_count": 775, "timestamp": 168483530.852270
83530.8522706, "value": "OlZbw7qgszwMGLOvko5u8hp5psS"} [x] Received {"device": 5, "message_count": 775, "timestamp": 1684983530.852270
6, "value": "OlZbw7qgszwMGLOvko5u8hp5psS"}el timestamp del receptor es: 1684983530.852270
device 2 sent Message 772: {"device": 2, "message_count": 772, "timestamp": 168483530.854918
83530.8549189, "value": "h"} [x] Received {"device": 2, "message_count": 772, "timestamp": 1684983530.854918
5, "value": "h"}el timestamp del receptor es: 1684983530856
device 1 sent Message 771: {"device": 1, "message_count": 771, "timestamp": 168483530.857623
83530.8545606, "value": "BEOqWRl7"} [x] Received {"device": 4, "message_count": 774, "timestamp": 1684983530.857623
8, "value": "yzEkBhUTBw6p40Dohtr8fNH1lpqlccnjhOBGbnzsmqDfpXgkF3alp"} [x] Received {"device": 3, "message_count": 778, "timestamp": 1684983530.867851
83530.8576238, "value": "yzEkBhUTBw6p40Dohtr8fNH1lpqlccnjhOBGbnzsmqDfpXgkF3alp"}el timestamp del receptor es: 1684983530860
device 1 sent Message 776: {"device": 1, "message_count": 776, "timestamp": 168483530.868041
83530.8685606, "value": "tJfhWRDpb4oc4yAZRCVbtAfNljqfnw5qgnvaTsF2Y130PnRnZSN"} [x] Received {"device": 4, "message_count": 779, "timestamp": 1684983530.868041
7, "value": "wur75"}el timestamp del receptor es: 1684983530869
device 4 sent Message 779: {"device": 4, "message_count": 779, "timestamp": 1684983530.868560
83530.86860413, "value": "tJlbDecWmu0G01ST4xV8G6Zc4d4wTuJpdjq47zsNyvriUkrx8kLl6A"} [x] Received {"device": 1, "message_count": 776, "timestamp": 1684983530.868560
6, "value": "tJlbDecWmu0G01ST4xV8G6Zc4d4wTuJpdjq47zsNyvriUkrx8kLl6A"}el timestamp del receptor es: 1684983530869
device 3 sent Message 778: {"device": 3, "message_count": 778, "timestamp": 1684983530.868803
83530.8678517, "value": "wur75"} [x] Received {"device": 2, "message_count": 777, "timestamp": 1684983530.868803
device 2 sent Message 777: {"device": 2, "message_count": 777, "timestamp": 1684983530.870214
83530.868803, "value": "o0EnYzg0hcublwWjLzyZcf7hq0z"} [x] Received {"device": 5, "message_count": 780, "timestamp": 1684983530.870214
device 5 sent Message 780: {"device": 5, "message_count": 780, "timestamp": 1684983530.870214
83530.8702145, "value": "Bb"} [x] Received {"device": 5, "message_count": 780, "timestamp": 1684983530.870214
5, "value": "Bb"}el timestamp del receptor es: 1684983530871
```

Figura 22: Latencia RabbitMQ

## Persistencia de mensajes

¿Qué opciones ofrecen ambos sistemas para el almacenamiento y recuperación de mensajes?

Para la persistencia, tanto Kafka como rabbitMQ almacenan los mensajes en el broker bajo sus propios criterios. Por ejemplo, kafka permite almacenar los mensajes durante X tiempo o por el tamaño. Por otro lado RabbitMQ tiene una política de retención de colas.

## Facilidad de uso y administración

¿Cuál de los dos sistemas es más fácil de configurar y administrar?

Tomando los sistemas vistos anteriormente podemos decir sin ninguna duda que Kafka es más complicado de configurar por su tolerancia a fallos y seguridad por otro lado RabbitMQ es más sencillo pero cuenta con menos funcionalidades que kafka. Por otro lado ambas poseen una alta escalabilidad, aunque rabbitMQ tuvo un mejor rendimiento en esta tarea por su velocidad, fue la que flexibilidad tenía a la hora de intentar cosas nuevas. En otras palabras Kafka requiere un mayor conocimiento en el campo y RabbitMQ está más orientado a ser amigable con el usuario, sacrificando funcionalidades por flexibilidad.

## 4. Cuarto caso

Por último, se plantea un escenario específico donde los dispositivos IoT tienen 5 categorías asignadas por nosotros y se tiene que enviar información a través de distintos canales dependiendo de la categoría que se encuentre en el dispositivo.

La pregunta que se presenta en este último caso es ¿Cómo puede enviar esta información teniendo un sólo broker?

La solución a este problema es crear tópicos en los sistemas que nos permiten crear una especie de filtro para los mensajes recibidos y enviarlos a distintos Consumers dependiendo del tópico al que estén inscritos por así decirlo, teniendo como resultado lo que se ve en las imágenes 23 y 24.

```

shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/dockerFinal/kafka$ node producerKafka.js
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/dockerFinal/kafka$ node consumer1
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/dockerFinal/kafka$ node consumer2
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/dockerFinal/kafka$ node consumer3
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/dockerFinal/kafka$ node consumer4
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/dockerFinal/rabbitMQ$ node consumer1
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/dockerFinal/rabbitMQ$ node consumer2
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/dockerFinal/rabbitMQ$ node consumer3
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/dockerFinal/rabbitMQ$ node consumer4

```

Figura 23: Tópicos Kafka.

```

shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/dockerFinal/rabbitMQ$ node consumer1
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/dockerFinal/rabbitMQ$ node consumer2
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/dockerFinal/rabbitMQ$ node consumer3
shsnow@shsnow-Modern-14-B5M: ~/Documents/CodeWatonesGPT/T2SD/dockerFinal/rabbitMQ$ node consumer4

```

Figura 24: Tópicos RabbitMQ.

Lamentablemente para el sistema de RabbitMQ no se pudo realizar, la lógica para solucionar este problema sigue siendo la misma que para Kafka.