

### Taller 3 - Patrón Publicador-Suscriptor (ZEROMQ)



María Paula Rodríguez Ruiz

Daniel Felipe Castro Moreno

Juan Enrique Rozo Tarache

Eliana Katherine Cepeda González

Introducción a Sistemas Distribuidos

Dpto. de Ingeniería de Sistemas

Pontificia Universidad Javeriana

Bogotá, Colombia

1 de Mayo del 2025

Para este taller se implementó una aplicación de cálculo distribuido usando el patrón publicador/suscriptor como medio de comunicación haciendo uso de ZMQ como middleware. La arquitectura básica se muestra en la Figura 1 a continuación.

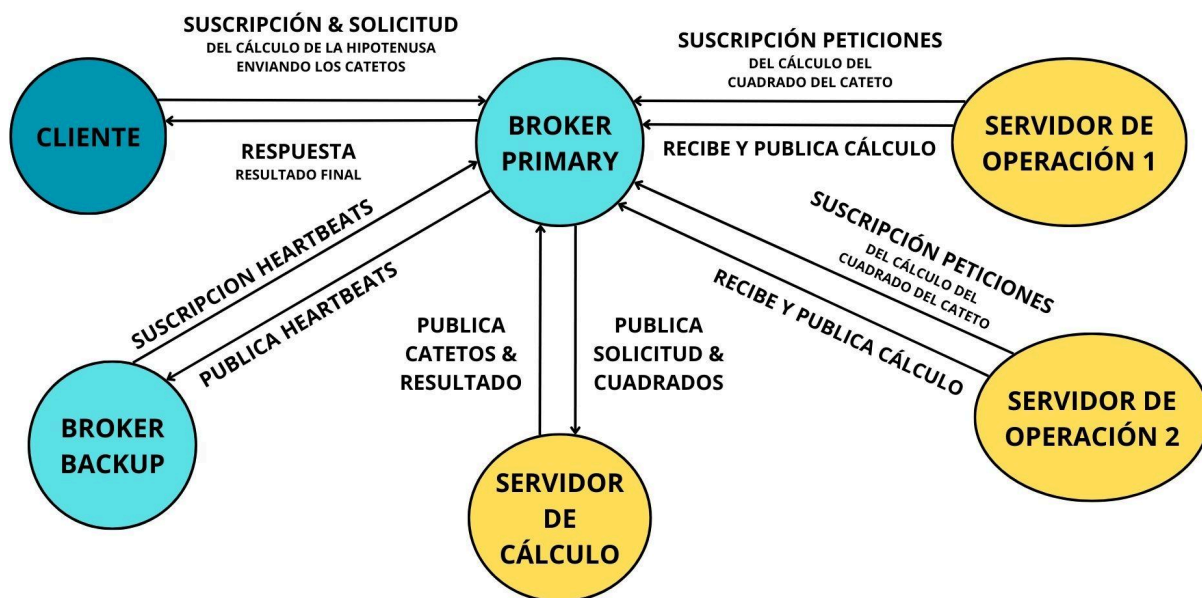


Figura 1. Arquitectura básica

Los tópicos que utilizamos fueron:

- CLIENT\_REQ1 = b"calc.request.1" - Tópico para el primer cateto que envía el cliente al Servidor de Cálculo
- CLIENT\_REQ2 = b"calc.request.2" - Tópico para el segundo cateto que envía el cliente al Servidor de Cálculo
- SQ\_REQ1 = b"calc.square.request.1" - Tópico para el cuadrado del primer cateto entre el Servidor de Cálculo y el Servidor de Operación 1
- SQ\_REQ2 = b"calc.square.request.2" - Tópico para el cuadrado del segundo cateto entre el Servidor de Cálculo y el Servidor de Operación 2

- TOPIC\_SQ\_RES1 = b"calc.square.result.1" - Tópico para el cuadrado del primer cateto entre el Servidor de Cálculo y el Servidor de Operación 1
- TOPIC\_SQ\_RES2 = b"calc.square.result.2" - Tópico para el cuadrado del segundo cateto entre el Servidor de Cálculo y el Servidor de Operación 2
- TOPIC\_HYP = b"calc.hypotenuse.result" - Tópico para retornar la hipotenusa al Cliente desde el Servidor de Cálculo

A continuación un desglose paso a paso de todo nuestro sistema distribuido bajo el patrón

Publicador–Suscriptor con bróker primario y de respaldo:

### 1. Bróker Primario (10.43.96.50)

- **Programa:** broker\_primary.py
- **Puertos:**
  - 5555 (XSUB) para recibir publicaciones de los publishers.
  - 5556 (XPUB) para enviar mensajes a los subscribers.
  - 5560 (PUB) para emitir heartbeats.
- **Función:** arranca un proxy ZMQ XSUB→XPUB y un hilo que cada segundo publica un latido en tcp://\*:5560.
- **Implementación:**

```

broker_primary.py > ...
1  # broker_primary.py
2  import zmq
3  import threading
4  import time
5
6  def heartbeat_loop(hb_pub):
7      while True:
8          hb_pub.send(b"HB")
9          time.sleep(1)
10
11 def main():
12     context = zmq.Context(1)
13
14     # XSUB recibe de publishers
15     frontend = context.socket(zmq.XSUB)
16     frontend.bind("tcp://*:5555")
17
18     # XPUB envía a subscribers
19     backend = context.socket(zmq.XPUB)
20     backend.bind("tcp://*:5556")
21
22     # PUB para heartbeats
23     hb_pub = context.socket(zmq.PUB)
24     hb_pub.bind("tcp://*:5560")
25
26     # Lanza hilo de heartbeats
27     threading.Thread(target=heartbeat_loop, args=(hb_pub,), daemon=True).start()
28
29     print("Broker PRIMARY escuchando en 5555/5556 (proxy) y 5560 (heartbeat)...")
30     try:
31         zmq.proxy(frontend, backend)
32     except KeyboardInterrupt:
33         pass
34     finally:
35         frontend.close()
36         backend.close()
37         hb_pub.close()
38         context.term()
39
40 if __name__ == "__main__":
41     main()
42

```

*Figura 2. Implementación broker\_primary.py*

## 2. Bróker de Respaldo (10.43.103.58)

- **Programa:** broker\_backup.py
- **Puertos escuchados:**
  - Se conecta a tcp://10.43.96.50:5560 en SUB para heartbeats.
- **Función:**
  - 1. Espera latidos del primario.
  - 2. Si pasan 3 s sin recibir ninguno, asume la caída del primario.
  - 3. Se enlaza entonces en puertos 5555/5556 y ejecuta zmq.proxy, convirtiéndose en el nuevo bróker activo.
- **Implementación:**

```
broker_backup.py > ...
1  # broker_backup.py
2  import zmq
3  import time
4
5  def main():
6      context = zmq.Context(1)
7
8      # SUB para heartbeats del primary
9      hb_sub = context.socket(zmq.SUB)
10     hb_sub.connect("tcp://10.43.96.50:5560")
11     hb_sub.setsockopt(zmq.SUBSCRIBE, b"")
12     hb_sub.setsockopt(zmq.RCVTIMEO, 3000) # 3 s sin latido → primary caído
13
14     print("Broker BACKUP en modo espera (escuchando heartbeats)...")
15     while True:
16         try:
17             hb_sub.recv() # bloquea hasta recibir o timeout
18         except zmq.Again:
19             # Primary cayó, tomo el relevo
20             print(" !! Primary caído. Activando proxy en BACKUP...")
21             frontend = context.socket(zmq.XSUB)
22             frontend.bind("tcp://*:5555")
23             backend = context.socket(zmq.XPUB)
24             backend.bind("tcp://*:5556")
25             print("Broker BACKUP ahora activo como PRIMARY (proxy).")
26             zmq.proxy(frontend, backend)
27             break
28
29     hb_sub.close()
30     context.term()
31
32 if __name__ == "__main__":
33     main()
34
```

Figura 3. Implementación broker\_backup.py

### 3. Servidor de Operación 1 (10.43.103.58)

- **Programa:** ServidorOperacion1.py
- **Conexiones:**
  - SUB a calc.square.request.1 en ambos brókers (10.43.96.50:5556 y 10.43.103.58:5556).
  - PUB para calc.square.result.1 a ambos brókers (10.43.96.50:5555 y 10.43.103.58:5555).
- **Función:** recibe el cateto 1, calcula su cuadrado y publica el resultado en calc.square.result.1.
- **Implementación:**

```

ServidorOperacion.py > ...
1  # operation_server1.py
2  import zmq, time
3
4  BROKER_PUBS = [
5      "tcp://10.43.96.50:5555", # primary
6      "tcp://10.43.103.58:5555", # backup
7  ]
8  BROKER_SUBS = [
9      "tcp://10.43.96.50:5556",
10     "tcp://10.43.103.58:5556",
11 ]
12
13 TOPIC_REQ = b"calc.square.request.1"
14 TOPIC_RES = b"calc.square.result.1"
15
16 def main():
17     ctx = zmq.Context()
18
19     # PUB socket (resultado)
20     pub = ctx.socket(zmq.PUB)
21     pub.setsockopt(zmq.IMMEDIATE, 1)
22     for addr in BROKER_PUBS:
23         pub.connect(addr)
24
25     # SUB socket (peticiones)
26     sub = ctx.socket(zmq.SUB)
27     sub.setsockopt(zmq.IMMEDIATE, 1)
28     for addr in BROKER_SUBS:
29         sub.connect(addr)
30     # ¡Al conectar primero, luego suscribimos!
31     sub.setsockopt(zmq.SUBSCRIBE, TOPIC_REQ)
32
33     # Damos tiempo para que se envíen las SUBSCRIBE a cualquiera de los brokers
34     time.sleep(0.2)
35
36     print("Operation Server 1 listo (cateto 1)...")
37     while True:
38         topic, msg = sub.recv_multipart()
39         value = int(msg)
40         result = value * value
41         print(f"  Cateto1: {value}² = {result}")
42         pub.send_multipart([TOPIC_RES, str(result).encode()])
43
44 if __name__ == "__main__":
45     main()
46

```

Figura 4. Implementación operation\_server1.py

#### 4. Servidor de Operación 2 (10.43.96.60)

- **Programa:** ServidorOperacion2.py
- **Conexiones:**
  - SUB a calc.square.request.2 en ambos brókers (10.43.96.50:5556 y 10.43.103.58:5556).
  - PUB para calc.square.result.2 a ambos brókers (10.43.96.50:5555 y 10.43.103.58:5555).
- **Función:** recibe el cateto 2, calcula su cuadrado y publica el resultado en calc.square.result.2.
- **Implementación:**

```

ServidorOperacion2.py > main
1  # operation_server2.py
2  import zmq, time
3
4  BROKER_PUBS = [
5      "tcp://10.43.96.50:5555", # primary
6      "tcp://10.43.103.58:5555", # backup
7  ]
8  BROKER_SUBS = [
9      "tcp://10.43.96.50:5556",
10     "tcp://10.43.103.58:5556",
11 ]
12
13 TOPIC_REQ = b"calc.square.request.2"
14 TOPIC_RES = b"calc.square.result.2"
15
16 def main():
17     ctx = zmq.Context()
18
19     # PUB socket (resultado)
20     pub = ctx.socket(zmq.PUB)
21     pub.setsockopt(zmq.IMMEDIATE, 1)
22     for addr in BROKER_PUBS:
23         pub.connect(addr)
24
25     # SUB socket (peticiones)
26     sub = ctx.socket(zmq.SUB)
27     sub.setsockopt(zmq.IMMEDIATE, 1)
28     for addr in BROKER_SUBS:
29         sub.connect(addr)
30     # ¡Al conectar primero, luego suscribimos!
31     sub.setsockopt(zmq.SUBSCRIBE, TOPIC_REQ)
32
33     # Damos tiempo para que se envíen las SUBSCRIBE a cualquiera de los brokers
34     time.sleep(0.2)
35
36     print("Operation Server 2 listo (cateto 2)...")
37     while True:
38         topic, msg = sub.recv_multipart()
39         value = int(msg)
40         result = value * value
41         print(f"  Cateto2: {value}^2 = {result}")
42         pub.send_multipart([TOPIC_RES, str(result).encode()])
43
44 if __name__ == "__main__":
45     main()
46

```

Figura 5. Implementación `operation_server2.py`

### 5. Servidor de Cálculo (10.43.96.50)

- **Programa:** ServidorCalculo.py
- **Conexiones:**
  - SUB en ambos brókers a los cuatro tópicos:
    - calc.request.1 y calc.request.2 (peticiones originales del cliente).
    - calc.square.result.1 y calc.square.result.2 (respuestas de operation servers).
  - PUB en ambos brókers para:

- reenviar a operation servers en `calc.square.request.1/.2`.
- publicar hipotenusa en `calc.hypotenuse.result`.

- **Flujo interno:**

- 1. Lee un mensaje en `calc.request.1` y otro en `calc.request.2` (los dos catetos).
- 2. Re-publica esos valores a `calc.square.request.1` y `calc.square.request.2`.
- 3. Espera resultados de los operation servers (hasta 2 s), descarta duplicados y hace “fallback” local si alguno no responde.
- 4. Calcula la hipotenusa y la publica en `calc.hypotenuse.result`.

- **Implementación:**



```

ServidorCalculo.py > ...
1  # calculation_server.py
2  import zmq, time, math
3
4  BROKER_PUBS = [
5      "tcp://10.43.96.50:5555",
6      "tcp://10.43.103.58:5555"
7  ]
8  BROKER_SUBS = [
9      "tcp://10.43.96.50:5556",
10     "tcp://10.43.103.58:5556"
11 ]
12
13 CLIENT_REQ1 = b"calc.request.1"
14 CLIENT_REQ2 = b"calc.request.2"
15 SQ_REQ1     = b"calc.square.request.1"
16 SQ_REQ2     = b"calc.square.request.2"
17 TOPIC_SQ_RES1 = b"calc.square.result.1"
18 TOPIC_SQ_RES2 = b"calc.square.result.2"
19 TOPIC_HYP     = b"calc.hypotenuse.result"
20
21 def main():
22     ctx = zmq.Context()
23
24     # PUB socket: reenviar a op1/op2 y enviar hipotenusa
25     pub = ctx.socket(zmq.PUB)
26     pub.setsockopt(zmq.IMMEDIATE, 1)
27     for addr in BROKER_PUBS:
28         pub.connect(addr)
29
30     # SUB socket: recibir del cliente y de los operation servers
31     sub = ctx.socket(zmq.SUB)
32     sub.setsockopt(zmq.IMMEDIATE, 1)
33     for addr in BROKER_SUBS:
34         sub.connect(addr)
35     # Suscribimos **tras** conectarnos
36     for topic in (CLIENT_REQ1, CLIENT_REQ2, TOPIC_SQ_RES1, TOPIC_SQ_RES2):
37         sub.setsockopt(zmq.SUBSCRIBE, topic)
38
39     # Esperamos breve para que las SUBSCRIBE lleguen al broker activo
40     time.sleep(0.2)
41
42     poller = zmq.Poller()
43     poller.register(sub, zmq.POLLIN)
44
45     print("Calculation Server esperando pareja de catetos...")

```

Figura 6. Implementación calculation\_server.py

```

ServidorCalculo.py > ...
21 def main():
45     print("Calculation Server esperando pareja de catetos...")
46     while True:
47         # 1) Recoger los dos catetos del cliente
48         reqs = {}
49         while len(reqs) < 2:
50             topic, msg = sub.recv_multipart()
51             if topic == CLIENT_REQ1:
52                 reqs['a'] = int(msg); print(f"Recibido cateto1: {reqs['a']}")
53             elif topic == CLIENT_REQ2:
54                 reqs['b'] = int(msg); print(f"Recibido cateto2: {reqs['b']}")
55
56         # 2) Reenviar cada cateto a su operation server
57         pub.send_multipart([SQ_REQ1, str(reqs['a']).encode()])
58         pub.send_multipart([SQ_REQ2, str(reqs['b']).encode()])
59
60         # 3) Esperar resultados con timeout y sin duplicados
61         results = {}
62         start = time.time()
63         timeout_ms = 2000
64         while len(results) < 2:
65             remaining = timeout_ms - int((time.time() - start) * 1000)
66             if remaining <= 0:
67                 break
68             socks = dict(poller.poll(remaining))
69             if sub in socks:
70                 topic, msg = sub.recv_multipart()
71                 if topic == TOPIC_SQ_RES1 and 'a2' not in results:
72                     results['a2'] = int(msg); print(f"Recibido square1: {results['a2']}")
73                 elif topic == TOPIC_SQ_RES2 and 'b2' not in results:
74                     results['b2'] = int(msg); print(f"Recibido square2: {results['b2']}")
75
76         # 4) Fallback local si falta alguno
77         a2 = results.get('a2', reqs['a']**2)
78         if 'a2' not in results:
79             print("⚠ op1 no respondió, calculando a² local")
80         b2 = results.get('b2', reqs['b']**2)
81         if 'b2' not in results:
82             print("⚠ op2 no respondió, calculando b² local")
83
84         # 5) Calcular y publicar hipotenusa
85         hyp = math.sqrt(a2 + b2)
86         print(f"❄ Publicando hipotenusa: {hyp:.4f}")
87         pub.send_multipart([TOPIC_HYP, f"{hyp:.4f}".encode()])
88
89         # Pequeña pausa antes de volver a escuchar
90         time.sleep(0.1)

```

Figura 7. Implementación calculation\_server.py

## 6. Cliente (10.43.103.51)

- **Programa:** client.py
- **Conexiones:**
  - PUB a ambos brokers en calc.request.1 y calc.request.2.
  - SUB a ambos brokers en calc.hypotenuse.result.
- **Flujo:**
  - 1. Publica los dos catetos.
  - 2. Se queda a la espera de un único mensaje en calc.hypotenuse.result.
  - 3. Imprime la hipotenusa recibida.
- **Implementación:**

```
Cliente.py > run
1  # client.py
2  import zmq, time
3
4  BROKER_PUB = "tcp://10.43.96.50:5555"
5  BROKER_SUB = "tcp://10.43.96.50:5556"
6
7  TOPIC_REQ1 = b"calc.square.request.1"
8  TOPIC_REQ2 = b"calc.square.request.2"
9  TOPIC_HYP  = b"calc.hypotenuse.result"
10
11 def run(a, b):
12     ctx = zmq.Context()
13     pub = ctx.socket(zmq.PUB)
14     pub.connect(BROKER_PUB)
15
16     sub = ctx.socket(zmq.SUB)
17     sub.connect(BROKER_SUB)
18     sub.setsockopt(zmq.SUBSCRIBE, TOPIC_HYP)
19
20     time.sleep(0.5) # pausa para asentarse PUB/SUB
21
22     # Envía catetos
23     pub.send_multipart([TOPIC_REQ1, str(a).encode()])
24     pub.send_multipart([TOPIC_REQ2, str(b).encode()])
25
26     # Recibe hipotenusa
27     topic, msg = sub.recv_multipart()
28     print(f"Hipotenusa recibida: {msg.decode()}")
29
30 if __name__ == "__main__":
31     run(12, 13)
32
```

Figura 8. Implementación Cliente.py

## Comunicación entre componentes

### Cliente → Servidor de Cálculo

- Tópicos calc.request.1 y calc.request.2 via bróker (primario o backup).

```
● estudiante@NGEN67:~/Documents/Distri$ python3 cliente.py
→ Enviando cateto1=12 en calc.request.1
→ Enviando cateto2=13 en calc.request.2
← Hipotenusa recibida: 17.6918
```

*Figura 9. comunicación cliente-servidor de cálculo*

### Servidor de Cálculo → Servidor de Operación 1 y 2

- Mismos valores, republicados en calc.square.request.1 y 2.

```
○ estudiante@NGEN319:~/Documents/Distribu$ python3 ServidorCalculo.py
Calculation Server esperando pareja de catetos...
Recibido cateto1: 12
Recibido cateto2: 13
Recibido square2: 169
Recibido square1: 144
```

*Figura 10 . Comunicación Servidor de cálculo a Servidores de operación*

### Servidores de Operación 1 y 2 → Servidor de Cálculo

- Publican resultados en calc.square.result.1 y 2.

```
○ estudiante@NGEN74:~/Documents/taller3$ python3 ServidorOperacion.py
Operation Server 1 listo (cateto 1)...
Cateto1: 122 = 144
```

*Figura 11. Comunicación Servidor operación 1 a Servidor de cálculo*

```
estudiante@NGEN329:~/Documents/taller3$ python3 ServidorOperacion2.py
Operation Server 2 listo (cateto 2)...
Cateto2: 132 = 169
```

Figura 12. Comunicación Servidor operación 2 a Servidor de cálculo

#### Servidor de Cálculo → Cliente

- Publica hipotenusa en calc.hypotenuse.result.

```
estudiante@NGEN319:~/Documents/Distribu$ python3 ServidorCalculo.py
Calculation Server esperando pareja de catetos...
Recibido cateto1: 12
Recibido cateto2: 13
Recibido square2: 169
Recibido square1: 144
Publicando hipotenusa: 17.6918
```

Figura 13. Comunicación Servidor de cálculo a cliente

#### Fail-over

##### Servidor Operación → Servidor de cálculo

- Si el servidor de cálculo 1 y/o 2 caen, el servidor de cálculo no recibe la respuesta y hace el cálculo correspondiente al servidor que cayó

El servidor de operación cae

```
estudiante@NGEN74:~/Documents/taller3$ python3 ServidorOperacion.py
Operation Server 1 listo (cateto 1)...
Cateto1: 122 = 144
^CTraceback (most recent call last):
  File "/home/estudiante/Documents/taller3/ServidorOperacion.py", line 31, in <module>
    main()
  File "/home/estudiante/Documents/taller3/ServidorOperacion.py", line 23, in main
    topic, msg = sub.recv_multipart()
  File "/home/estudiante/.local/lib/python3.10/site-packages/zmq/sugar/socket.py", line 799, in recv_multipart
    parts = [self.recv(flags, copy=copy, track=track)]
  File "_zmq.py", line 1147, in zmq.backend.cython._zmq.Socket.recv
  File "_zmq.py", line 1182, in zmq.backend.cython._zmq.Socket.recv
  File "_zmq.py", line 1337, in zmq.backend.cython._zmq._recv_copy
  File "_zmq.py", line 169, in zmq.backend.cython._zmq._check_rc
KeyboardInterrupt

estudiante@NGEN74:~/Documents/taller3$
```

Figura 14. Servidor de operación cae

El servidor de operación 2 hace su trabajo

```
estudiante@NGEN329:~/Documents/taller3$ python3 ServidorOperacion2.py
Operation Server 2 listo (cateto 2)...
Cateto2: 132 = 169
Cateto2: 132 = 169
█
```

Figura 15. Servidor de operación 2 funcionando

EL servidor de cálculo hace el trabajo

```
Recibido cateto1: 12
Recibido cateto2: 13
Recibido square2: 169
⚠ op1 no respondió, calculando a2 local
🏁 Publicando hipotenusa: 17.6918
█
```

Figura 16. Respuesta servidor de cálculo

El cliente recibe su respuesta

```
estudiante@NGEN67:~/Documents/Distri$ python3 cliente.py
→ Enviando cateto1=12 en calc.request.1
→ Enviando cateto2=13 en calc.request.2
← Hipotenusa recibida: 17.6918
estudiante@NGEN67:~/Documents/Distri$
```

Figura 17. Respuesta a cliente

**Bróker Primario → Bróker de Respaldo**

Ambos Brokers deben estar activos inicialmente

Broker principal

```
estudiante@NGEN319:~/Documents/Distribu$ python3 broker_primary.py  
Broker PRIMARY escuchando en 5555/5556 (proxy) y 5560 (heartbeat)...  
█
```

*Figura 18. Broker principal funcionando*

Broker de respaldo

```
estudiante@NGEN319:~/Documents/Distribu$ python3 broker_primary.py  
Broker PRIMARY escuchando en 5555/5556 (proxy) y 5560 (heartbeat)...  
█
```

*Figura 19. Broker de respaldo funcionando*

- Si el bróker primario (10.43.96.50) cae, el backup (10.43.103.58) detecta la ausencia de latidos tras 3 s y toma la dirección de proxy en los mismos puertos.

```
● estudiante@NGEN319:~/Documents/Distribu$ python3 broker_primary.py  
Broker PRIMARY escuchando en 5555/5556 (proxy) y 5560 (heartbeat)...  
○ ^Cestudiante@NGEN319:~/Documents/Distribu$ █
```

*Figura 20. Broker principal cae*

```
estudiante@NGEN74:~/Documents/taller3$ python3 broker_backup.py
Broker BACKUP en modo espera (escuchando heartbeats)...
!! Primary caído. Activando proxy en BACKUP...
Broker BACKUP ahora activo como PRIMARY (proxy).
```

Figura 21. Broker de respaldo activo

- Todos los nodos (cliente, servidor de cálculo y servidores de operación 1 y 2) están conectados a ambos bróker, por lo que el cambio resulta transparente y el flujo continúa sin interrupción perceptible.

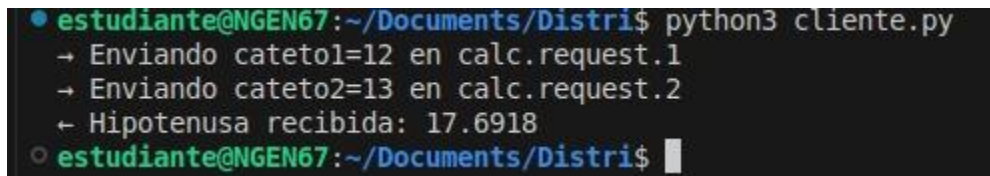
```
estudiante@NGEN329:~/Documents/taller3$ python3 ServidorOperacion2.py
Operation Server 2 listo (cateto 2)...
Cateto2:  $13^2 = 169$ 
Cateto2:  $13^2 = 169$ 
Cateto2:  $13^2 = 169$ 
```

Figura 22. Servidor de operación 2 funcionando

```
Recibido cateto1: 12
Recibido cateto2: 13
Recibido square2: 169
⚠ op1 no respondió, calculando  $a^2$  local
🚩 Publicando hipotenusa: 17.6918
```

Figura 23. Servidor de cálculo recibe la operación 2 y realiza la 1



A terminal window with a dark background and light-colored text. The prompt is 'estudiante@NGEN67:~/Documents/Distri\$'. The command 'python3 cliente.py' has been executed. The output shows two lines of data being sent: '→ Enviando cateto1=12 en calc.request.1' and '→ Enviando cateto2=13 en calc.request.2'. This is followed by a response line: '← Hipotenusa recibida: 17.6918'. The prompt returns to 'estudiante@NGEN67:~/Documents/Distri\$' with a cursor at the end.

```
● estudiante@NGEN67:~/Documents/Distri$ python3 cliente.py
→ Enviando cateto1=12 en calc.request.1
→ Enviando cateto2=13 en calc.request.2
← Hipotenusa recibida: 17.6918
○ estudiante@NGEN67:~/Documents/Distri$
```

*Figura 24. Respuesta a cliente.*

Para visualizar y comprender la dinámica de la comunicación entre los componentes de nuestro sistema de cálculo distribuido, basado en el patrón Publicador-Suscriptor con zeroMQ y una arquitectura de brokers primario/backup, se presentan los siguientes diagramas de secuencia.

Estos diagramas ilustran la interacción y el intercambio de mensajes entre los participantes clave (Cliente, Broker Primario, Broker Backup, Calculation Server, Operation Server 1 y Operation Server 2) a lo largo del tiempo. Se han elaborado dos diagramas principales para cubrir los escenarios fundamentales de operación:

**Flujo de Cálculo de Hipotenusa con Broker Primario Activo:** Este diagrama detalla la secuencia de eventos que ocurren durante una solicitud de cálculo de hipotenusa bajo condiciones normales de operación, donde el Broker Primario está activo y funcionando correctamente. Incluye la publicación inicial de la solicitud por parte del cliente, el reenvío por el broker, la interacción entre el Calculation Server y los Operation Servers para obtener los cuadrados de los catetos, el posible fallback local en caso de fallo de los servidores de operación, y finalmente la publicación y recepción del resultado de la hipotenusa.

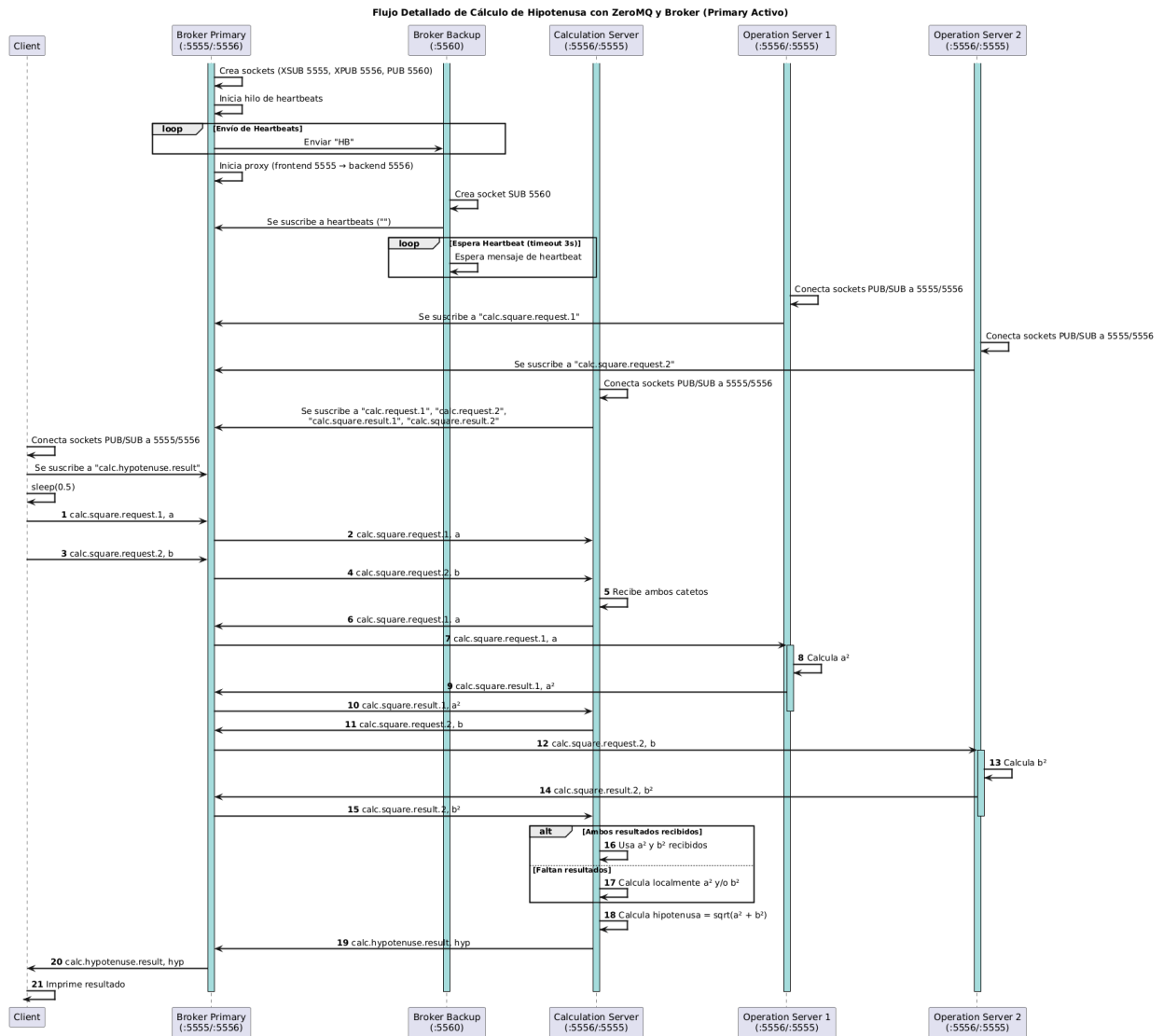


Figura 25. Diagrama de secuencia con broker primario activo.

**Escenario de Failover:** Activación del Broker Backup tras Falla del Primario: Este diagrama se enfoca en la resiliencia del sistema, mostrando cómo el Broker Backup detecta la falla del Broker Primario a través de la ausencia de heartbeats. Se ilustra el proceso de toma de control por parte del Broker Backup, asumiendo el rol de proxy principal. Posteriormente, se muestra cómo los clientes y servidores, previamente conectados a ambos brokers, redirigen su comunicación automáticamente al Broker Backup activo para completar una nueva solicitud de cálculo.

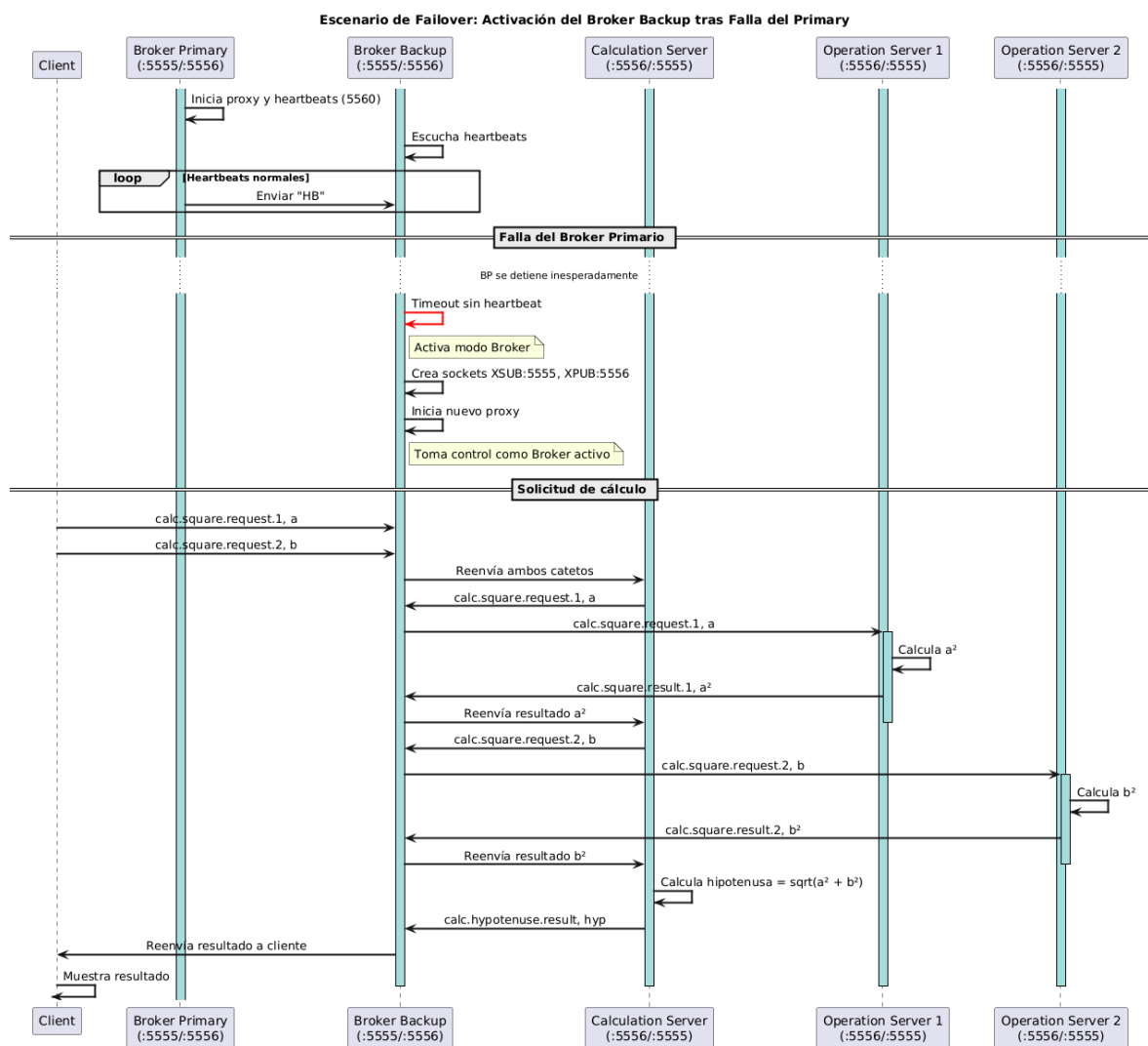


Figura 26. Diagrama de secuencia de escenario failover.