Integración de aplicaciones

Tema 4. Sistemas basados en mensajería

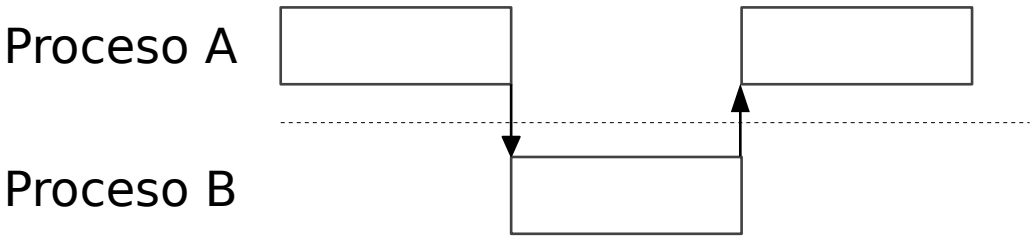
Contenido

- Introducción
- Mensajería
- ZeroMQ
- Sistemas de mensajería
- RabbitMQ

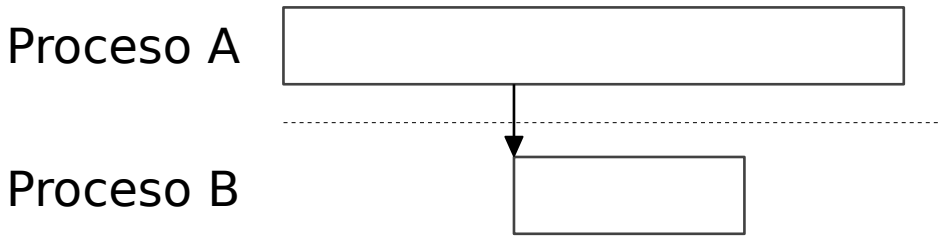
Introducción

Comunicaciones síncronas

- Las arquitecturas SOA utilizan generalmente comunicaciones síncronas, que bloquean al consumidor



Llamada síncrona



Llamada asíncrona

Introducción

Comunicaciones síncronas

- Problemas:
  - Retardos en el consumidor
  - Acoplamiento espacial y temporal entre consumidor-proveedor
  - Comunicaciones 1-a-1
  - El consumidor debe gestionar los errores en las comunicaciones (reintentos): menor cohesión
- Otro modelo de comunicación es posible: mensajería

Mensajería

Sistema de mensajería

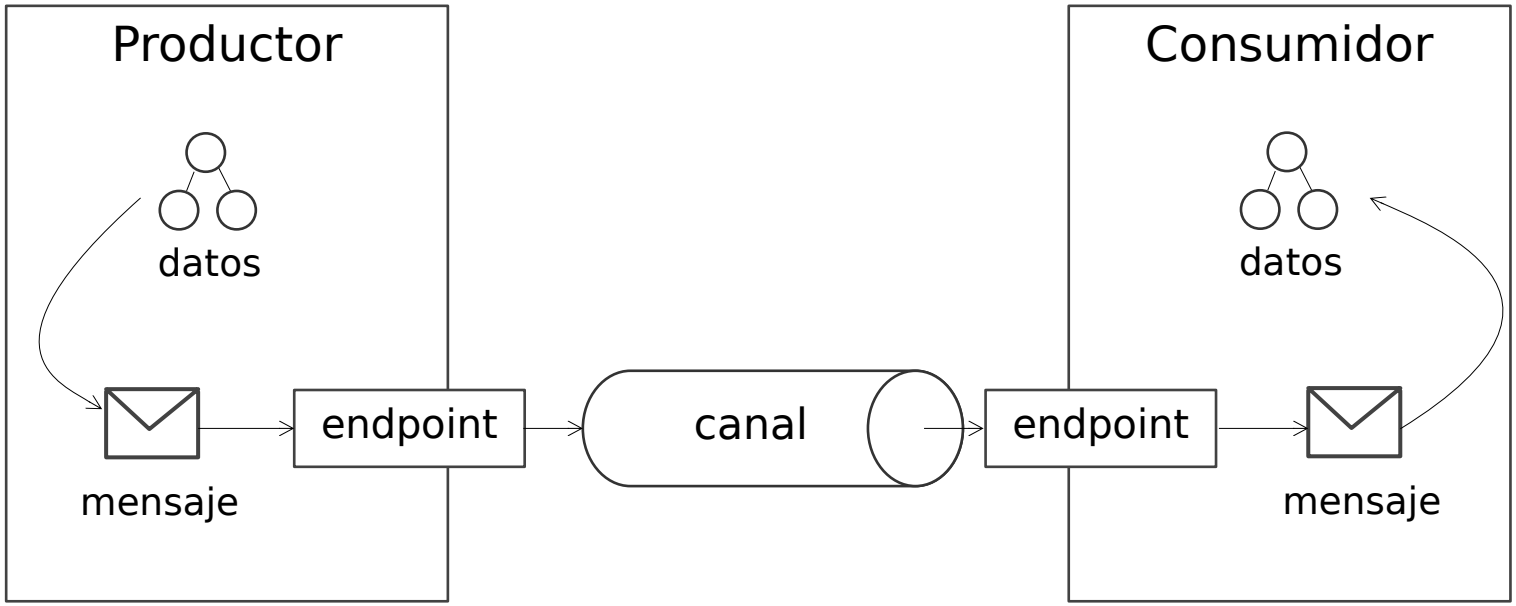
- Un productor envía un mensaje a un consumidor a través de un sistema de mensajería

- El sistema de mensajería puede ser
  - Directo (e.g. ZeroMQ)
  - A través de un intermediario-bróker (e.g. RabbitMQ)

Mensajería

Modelo

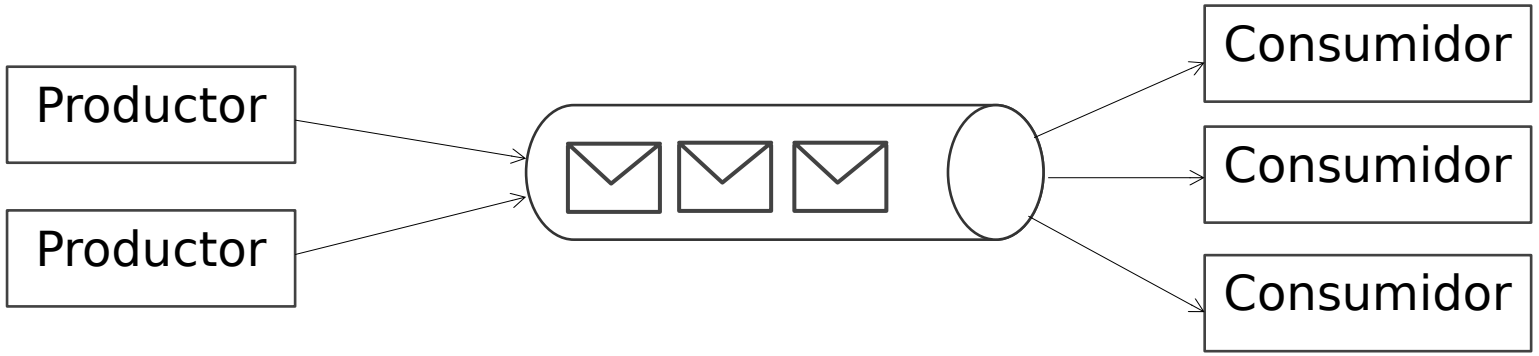
- El productor escribe un mensaje en un canal a través de un endpoint. La comunicación es unidireccional y asíncrona



Mensajería

Canal ≡ Cola de mensajes

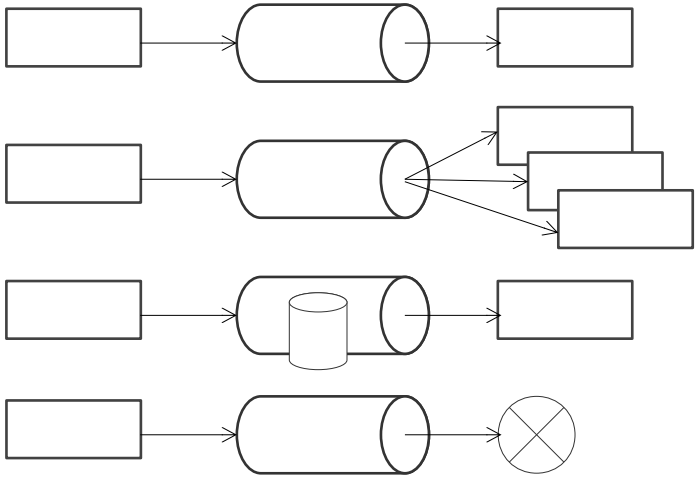
- Tubería lógica de mensajes FIFO
- Añade un nivel adicional de indirección entre el productor-consumidor
- Desacoplamiento espacial y temporal



Mensajería

Tipos de canales

- Punto a punto
- Pub-sub
- Entrega garantizada
- Dead-letter
- etc.



Mensajería

Ventajas

- Rendimiento en productor
- Desacoplamiento velocidades procesamiento
- Desacoplamiento temporal
- Mejora resiliencia y gestión de errores

Mensajería

Retos

- Modelo de programación complejo
- No aplicable en algunos escenarios síncronos
- Entrega de mensajes fuera de orden
- Sobrecarga en la comunicación
- Soporte disponible en la plataforma y lock-in

Mensajería

Patrones de mensajería

- Fire and forget: productor envía mensaje y continúa trabajando
- Request-reply: productor envía una petición y espera una respuesta
- Publish-subscribe: los consumidores se suscriben a un tema, y los productores publican mensajes en dicho tema, que son recibidos por todos los suscriptores

ZeroMQ

- ZeroMQ es una librería muy rápida y eficiente que habilita mensajería
- Cero bróker, cero latencia, cero coste, cero administración
- Soporta múltiples transportes: TCP, intra e inter-proceso
- Instalación

```
> npm install zeromq@5
```

ZeroMQ

Sockets

- Extiende el concepto básico de socket para habilitar mensajería

Consumidor - Servidor

```
const zmq = require('zeromq');
const sock = zmq.socket('<type>');
sock.bind('<spec>');
sock.on('message', (msg) => { ... });
sock.close();
```

Productor - Cliente

```
const zmq = require('zeromq');
const sock = zmq.socket('<type>');
sock.connect('<spec>');
sock.send('<data>');
sock.close();
```

ZeroMQ

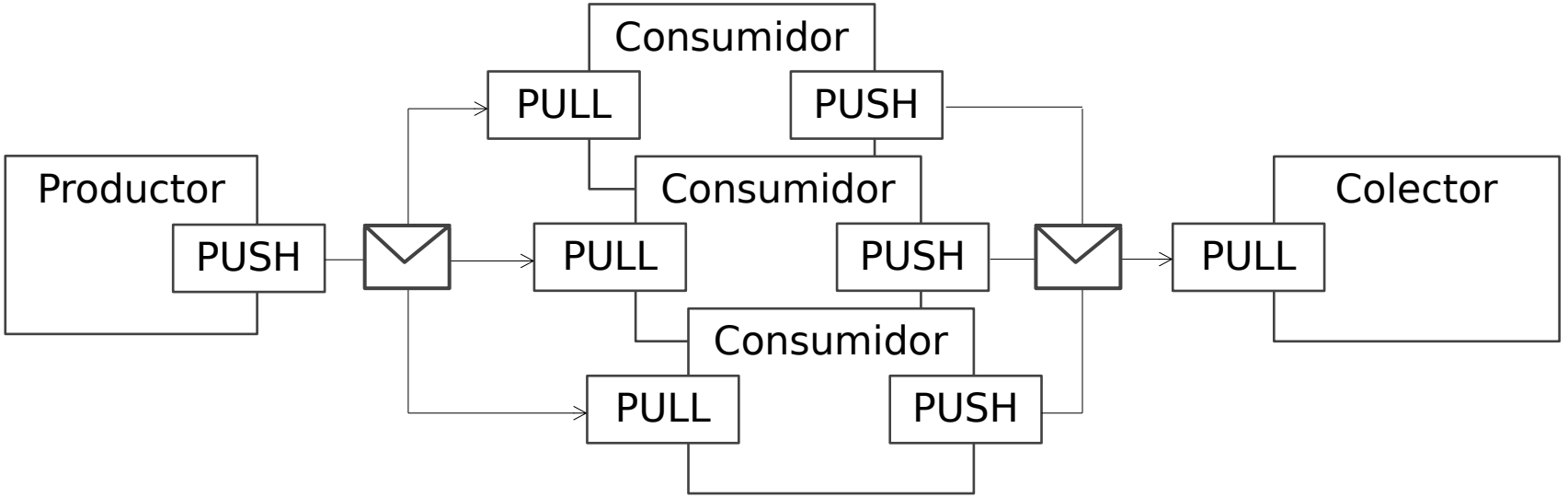
Sockets

- Existen distintos tipos de sockets que habilitan distintos patrones de mensajería. Se utilizan en pares ...
  - Push-pull
  - Request-reply
  - Pub-sub
  - etc.

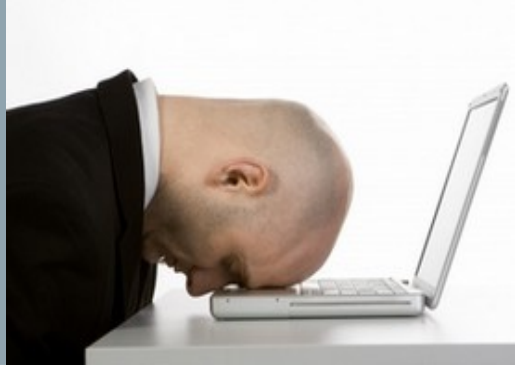
ZeroMQ

Push-pull

- Comunicación asíncrona unidireccional (fire & forget)
- Sockets PUSH/PULL, múltiples con round robin
- En arquitecturas pipeline

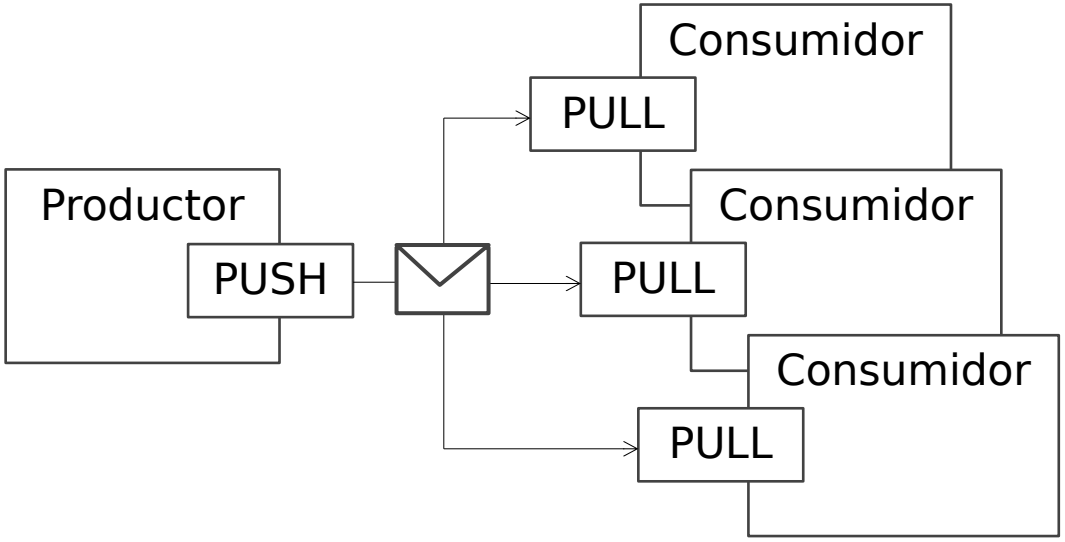


ZeroMQ



Push-pull > Ejercicio 1

- Crear productor/consumidor usando push-pull



ZeroMQ

Push-pull > Ejercicio 1

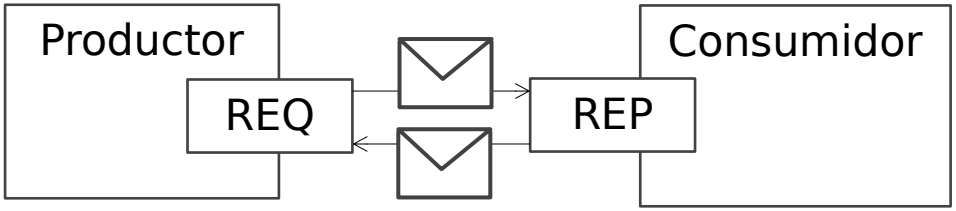
```
// productor
const zmq = require("zeromq");
const sock = zmq.socket('push');
sock.connect('tcp://127.0.0.1:3000');
sock.send('Hello');
sock.close();
```

```
// consumidor
const zmq = require("zeromq");
const sock = zmq.socket('pull');
sock.bind("tcp://127.0.0.1:3000", (err) => {
  if (err) console.log(err.stack);
  else console.log('Listening on 3000');
});
sock.on('message', (msg) => {
  console.log('Received message: ' +
  msg.toString());
});
```

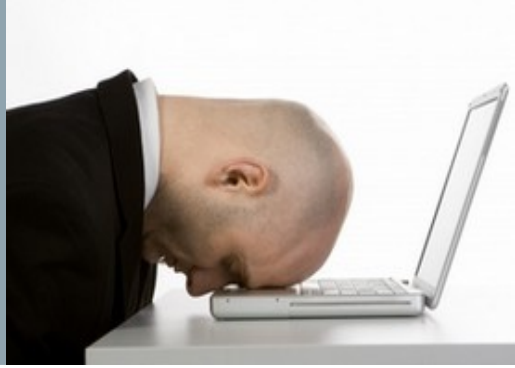
ZeroMQ

Request-reply

- Habitual en SOA
- Sockets REQ/REP
- Soporta múltiples secuencias petición-respuesta consecutivas

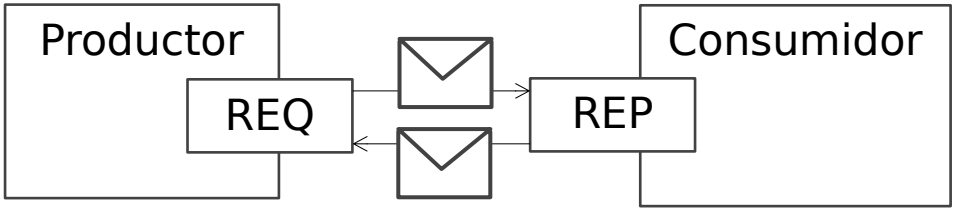


ZeroMQ

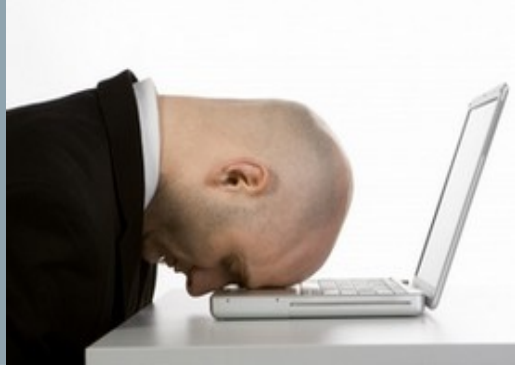


Request-reply > Ejercicio 2

- Crear productor/consumidor usando request-reply



ZeroMQ



Request-reply > Ejercicio 2

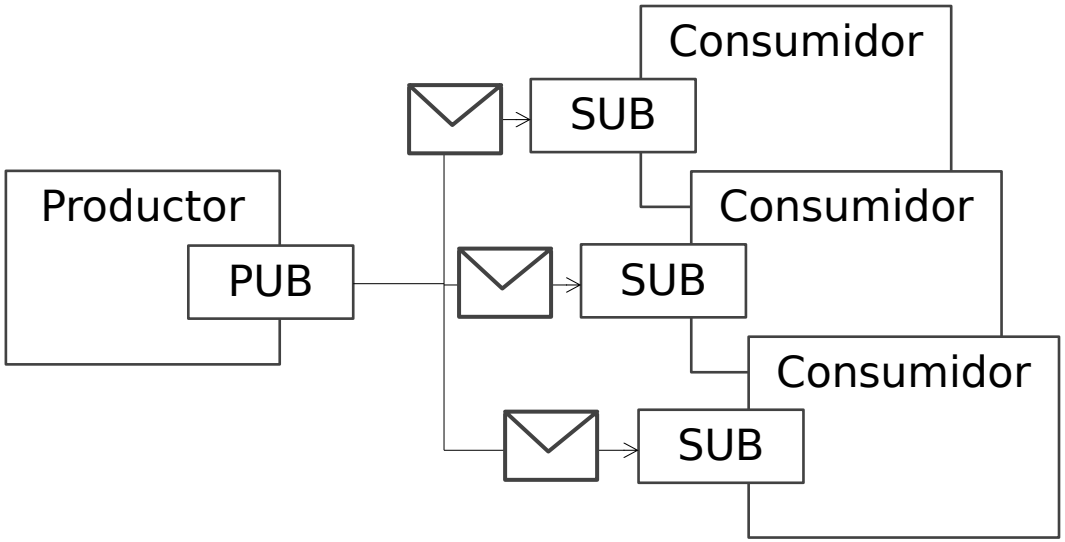
```
// productor
const zmq = require("zeromq");
const sock = zmq.socket('req');
sock.connect('tcp://127.0.0.1:3000');
sock.send('Hello');
sock.on('message', (msg) => {
  console.log('Received reply: ' + msg.toString());
  sock.close();
  process.exit(0);
});
```

```
// consumidor
const zmq = require("zeromq");
const sock = zmq.socket('rep');
sock.bind("tcp://127.0.0.1:3000", (err) => {
  if (err) console.log(err.stack);
  else console.log('Listening on 3000');
});
sock.on('message', (msg) => {
  console.log('Received message: ' + msg.toString());
  sock.send(msg + ' world!');
});
```

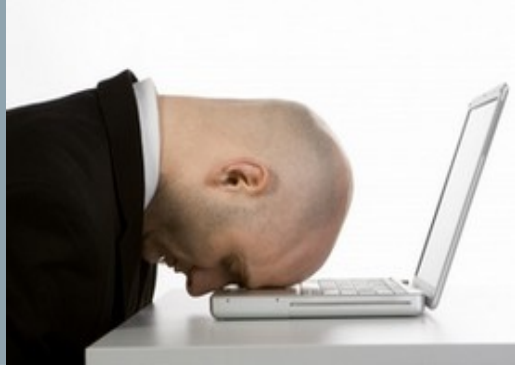
ZeroMQ

Pub-sub

- Difusión de mensajes, sin garantías de entrega
- Sockets PUB/SUB, suscripción en temas

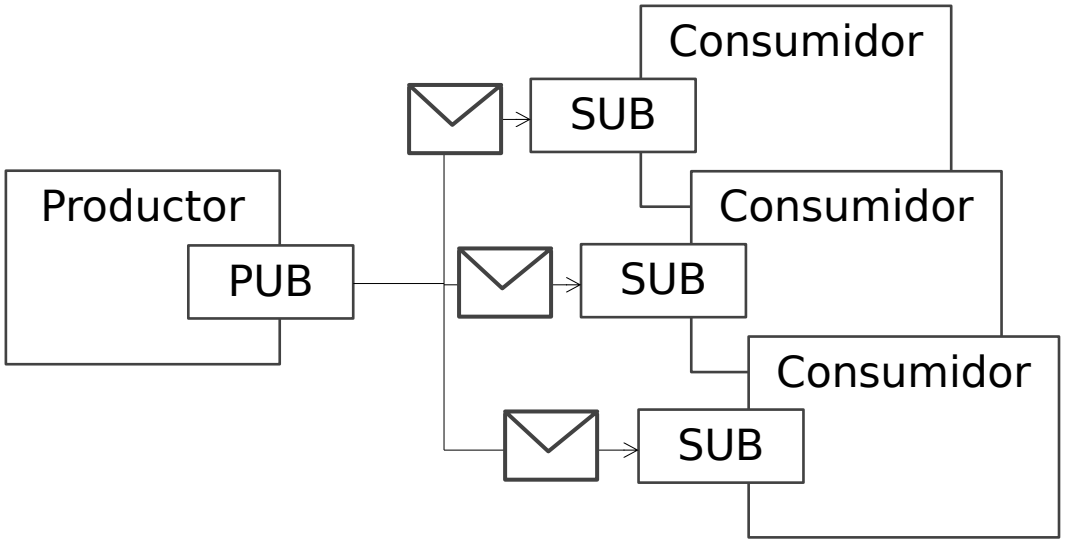


ZeroMQ



Pub-sub > Ejercicio 3

- Crear publicador/suscriptor usando pub-sub, enviando el tiempo actual



ZeroMQ

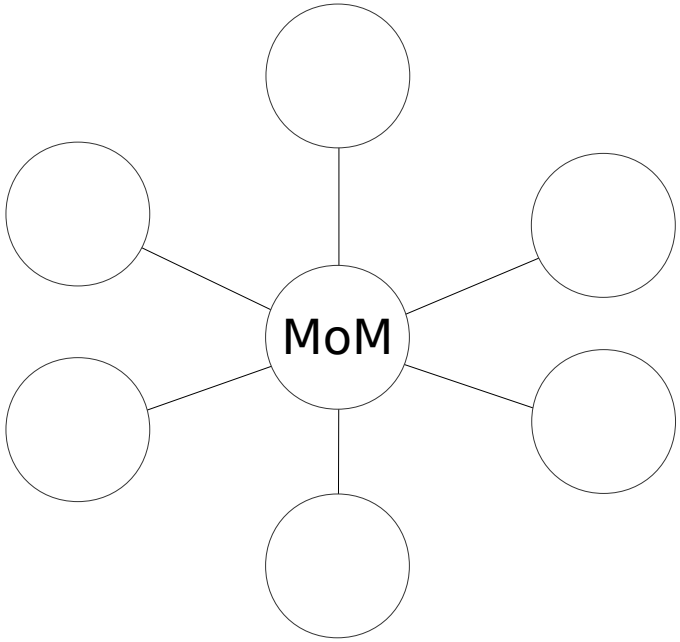
Pub-sub > Ejercicio 3

```
// publicador
const zmq = require("zeromq");
const sock = zmq.socket('pub');
sock.bindSync("tcp://127.0.0.1:3000");
setInterval(() => {
  sock.send(['time','Current time is: ' + new Date().toLocaleString()]);
},1000);
```

```
// suscriptor
const zmq = require("zeromq");
const sock = zmq.socket('sub');
sock.subscribe('time');
sock.on('message', (topic, msg) => {
  console.log('Received: ' + msg.toString());
});
sock.connect('tcp://127.0.0.1:3000');
```

Sistemas de mensajería

- Sistema que centraliza las comunicaciones de mensajería (MoM, bróker, ESB)
- Arquitectura hub and spoke



Sistemas de mensajería

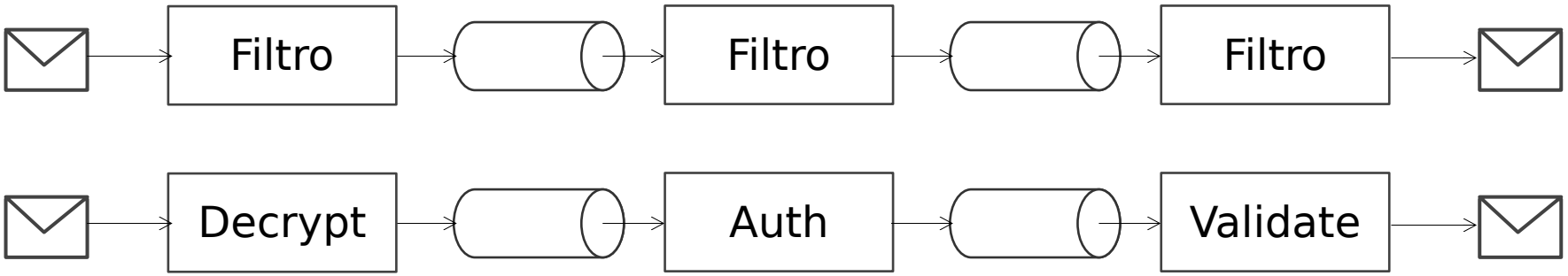
Ventajas

- Simplificación del código de integración
- Desacoplamiento espacial
- Mejora en escalado
- Gestión y procesamiento de mensajes avanzado

Sistemas de mensajería

Arquitectura

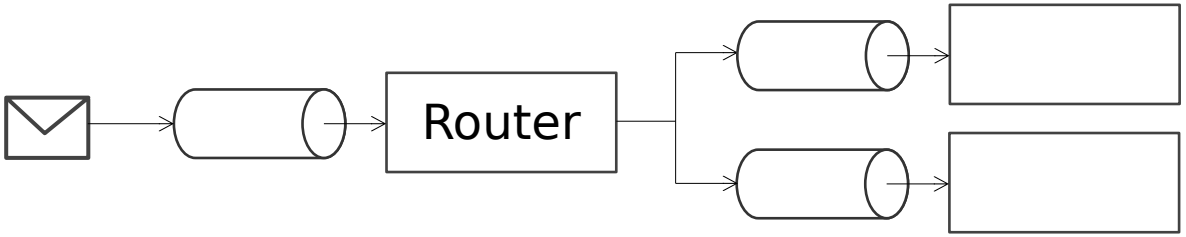
- Pipe-and-filter
- El mensaje es procesado por filtros
- Se conectan con tuberías (canales)



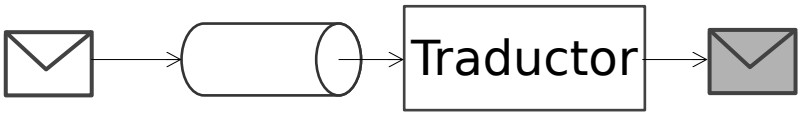
Sistemas de mensajería

Filtros

- Enrutador de mensajes: basados en contenido, dinámicos, splitters, agregadores, ...



- Traductor de mensajes: enriquecedores, filtros, ...

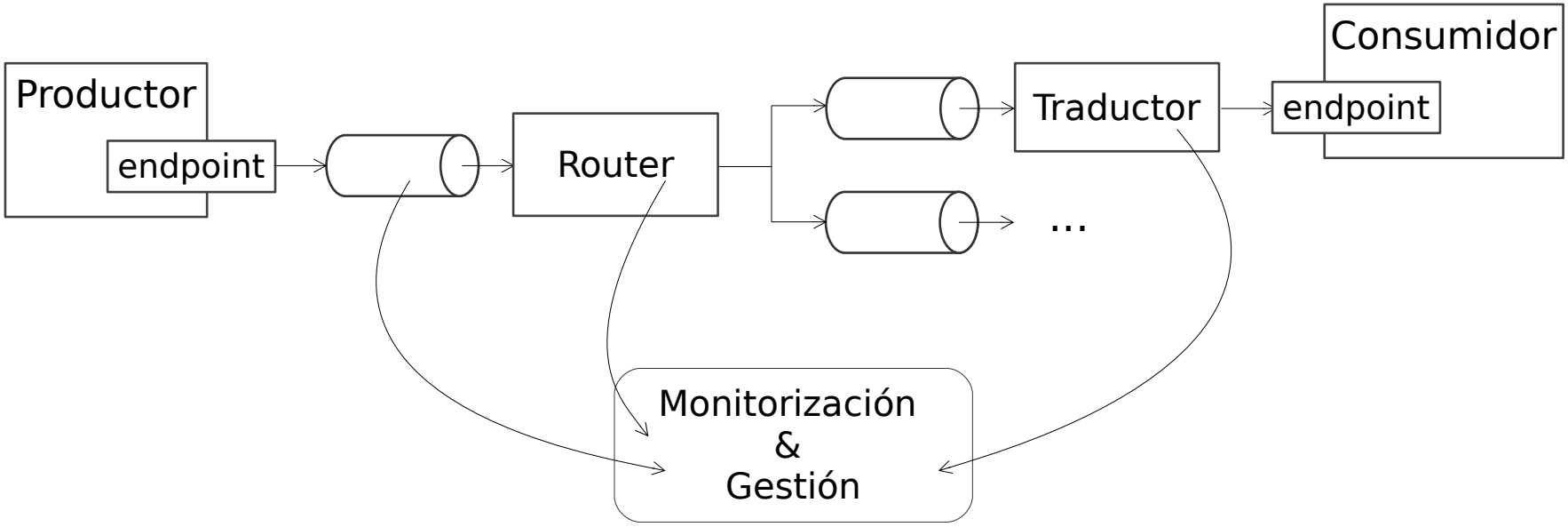


- etc.

Sistemas de mensajería

Resumen

- Arquitectura pipe-and-filter con canales, enrutadores, traductores, monitorización



RabbitMQ

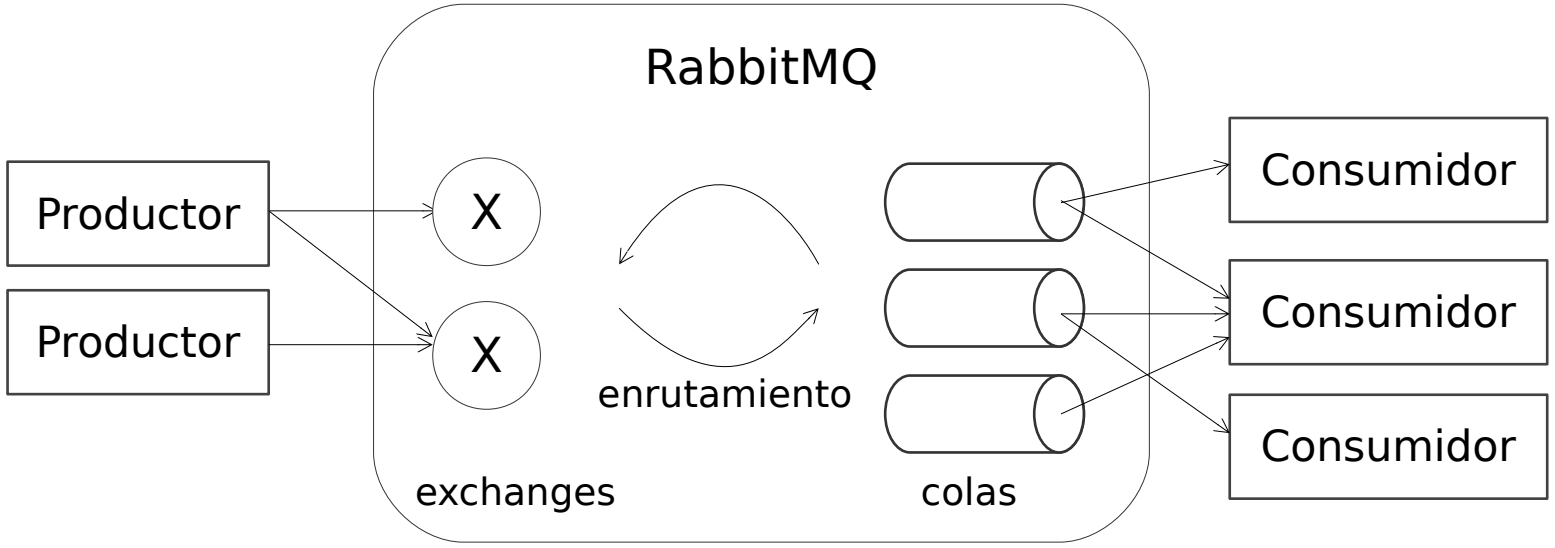
- RabbitMQ es un bróker de mensajería ligero, open source, desarrollado en Erlang
- Soporta múltiples protocolos de mensajería
- Disponible para múltiples plataformas

```
> sudo apt install rabbitmq-server
> sudo systemctl start rabbitmq-server
> sudo rabbitmqctl status
```

RabbitMQ

AMQP 0-9-1

- Protocolo de mensajería núcleo de RabbitMQ
- Comunicación productor-consumidor con bróker
- Entidades: exchanges, colas, bindings



RabbitMQ

AMQP 0-9-1

- Productores publican mensajes en exchanges
- Las colas se registran en exchanges
- Exchanges enrutan a las colas
- Consumidores reciben mensajes de colas suscritas
- Para comunicarnos con el bróker en Node.js usaremos la librería cliente amqplib

```
> npm install amqplib
```

RabbitMQ

Conexiones y canales

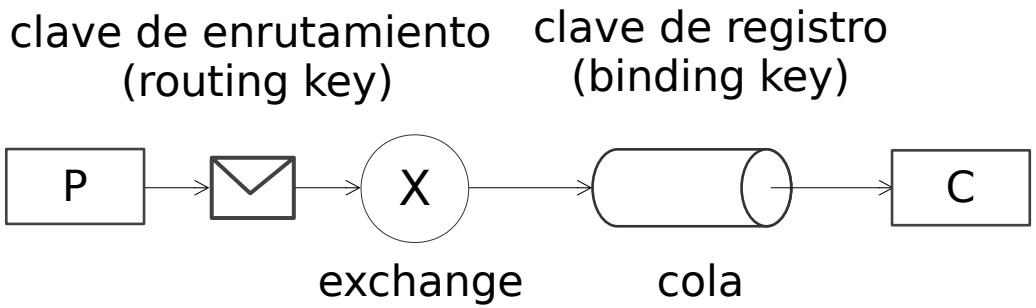
- Una conexión contiene múltiples canales
- Solicitamos operaciones a través del canal

```
var amqp = require('amqplib/callback_api');
amqp.connect('amqp://localhost', function(err, con){
  if (err) console.log(err.stack);
  else {
  con.createChannel(function(err, channel) {
    if (err) console.log(err.stack);
    else console.log('Channel created.');
    /* execute operations */
    con.close();
  });
}});
```

RabbitMQ

Exchanges y colas

- Las colas se registran (binding) en los exchanges usando una clave de registro (binding key)
- Los mensajes se envían a los exchanges usando una clave de enrutamiento (routing key)
- El exchange determina las colas a enviar comparando ambas claves



RabbitMQ

Exchanges y colas > Ejemplo productor

```
var amqp = require('amqplib/callback_api');
amqp.connect('amqp://localhost', function (err, con) {
  if (err) console.log(err.stack);
  else {
    con.createChannel(function (err, channel) {
      if (err) console.log(err.stack);
      else {
        var msg = 'Hello world';
        channel.assertExchange('hello-exchange', 'direct',
                               {durable: false});
        channel.publish('hello-exchange', 'greetings',
                        Buffer.from(msg));
        setTimeout( () => {
          con.close();process.exit(0);}, 500);
      }});
}});
```

RabbitMQ

Exchanges y colas > Ejemplo consumidor

```
var amqp = require('amqplib/callback_api');
amqp.connect('amqp://localhost', function (err, con) {
  if (err) console.log(err.stack);
  else {
    con.createChannel((err, channel) => {
      if (err) console.log(err.stack);
      else {
        channel.assertQueue('hello-queue',{ durable: false });
        channel.bindQueue('hello-queue','hello-exchange',
                          'greetings');
        console.log('Waiting for messages');
        channel.consume('hello-queue', (msg) => {
          console.log('Received ' + msg.content.toString());
        },{ noAck: true });
    }});
}});
```

RabbitMQ

Exchanges y colas > Comandos útiles

```
> sudo rabbitmqctl list_exchanges
> sudo rabbitmqctl list_queues
> sudo rabbitmqctl list_bindings
```

RabbitMQ

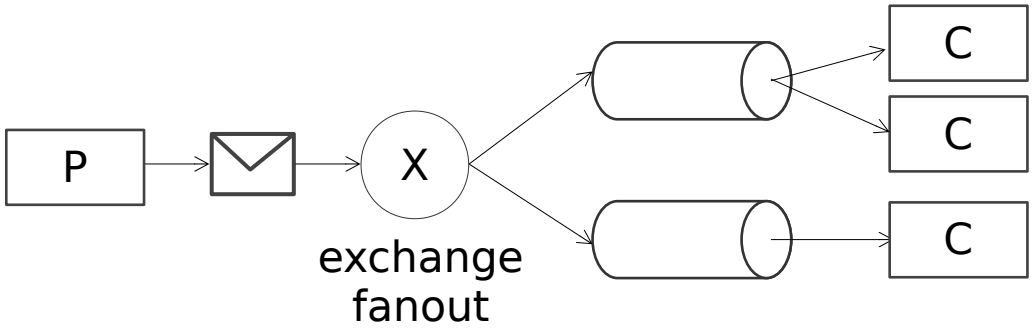
Exchanges y colas > Tipos de exchanges

- Habilitan distintos comportamientos en el enrutamiento de mensajes
  - Fanout
  - Direct
  - Topic
  - etc.

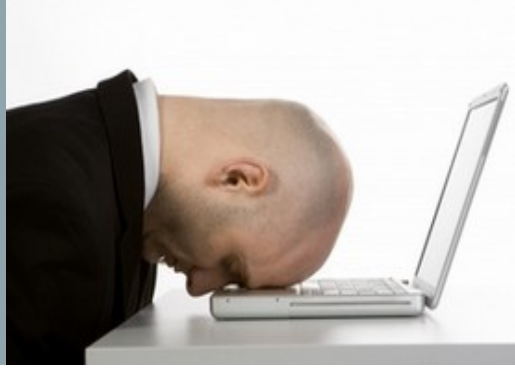
RabbitMQ

Exchange fanout

- Implementa patrón pub/sub: el exchange difunde mensaje a todas las colas suscritas
- No se hace uso de las claves
- Múltiples consumidores por cola: round robin

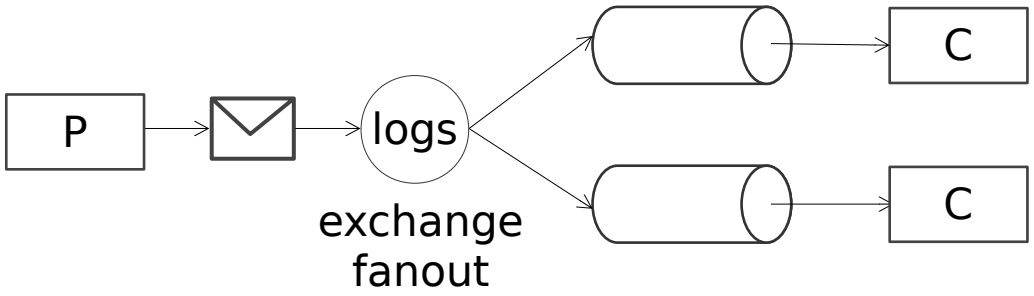


RabbitMQ

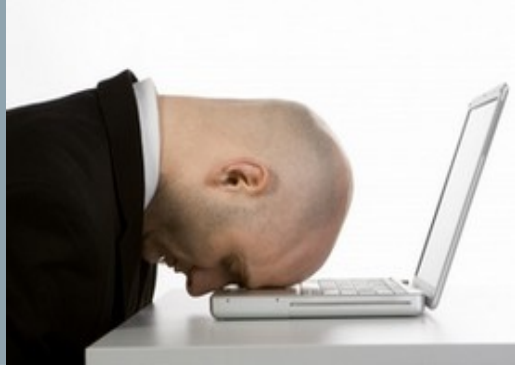


Exchange fanout > Ejercicio 4

- Crear sistema de log con un exchange fanout



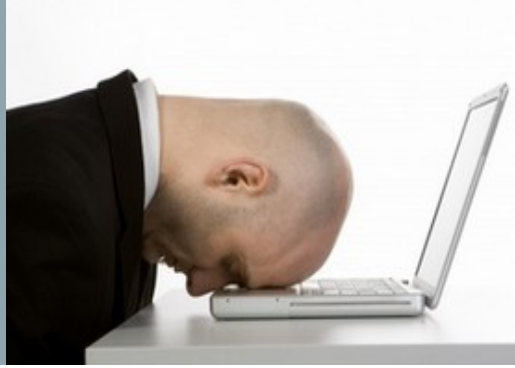
RabbitMQ



Exchange fanout > Ejercicio 4 > productor

```
/* connect to channel */
...
var msg = 'Hello world';
channel.assertExchange('logs', 'fanout', {durable: false});
channel.publish('logs', '', Buffer.from(msg));
setTimeout( () => {
  con.close();process.exit(0);}, 500);
...
```

RabbitMQ



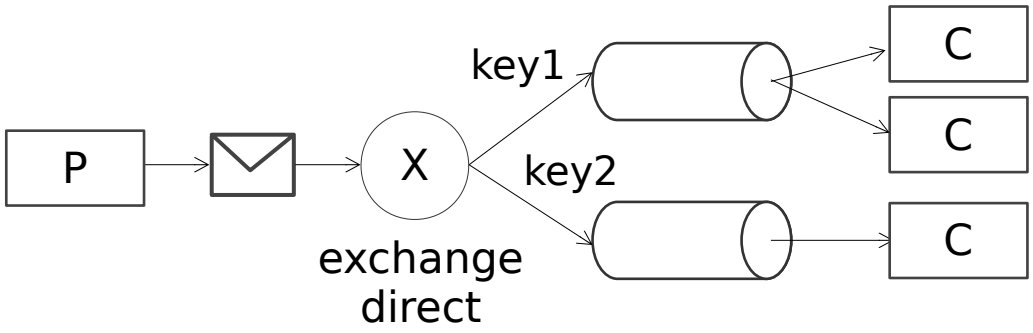
Exchange fanout > Ejercicio 4 > consumidor

```
/* connect to channel */
...
channel.assertExchange('logs', 'fanout', { durable: false });
channel.assertQueue('', { exclusive: true }, (err, q) => {
  if (err) console.log(err.stack);
  else {
    console.log('Waiting for messages');
    channel.bindQueue(q.queue, 'logs', '');
    channel.consume(q.queue, function (msg) {
      console.log('Received ' + msg.content.toString());
    }, { noAck: true });
  }
});
...
```

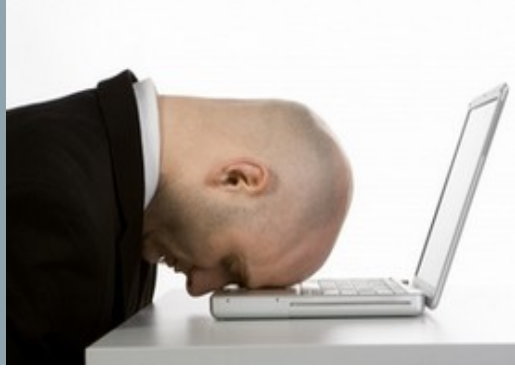
RabbitMQ

Exchange direct

- Añade capacidades básicas de enrutamiento
- Comprobación de claves de registro/enrutamiento
- El mensaje se copia en todas las colas con claves coincidentes (multicast!)
- Múltiples consumidores por cola: round robin

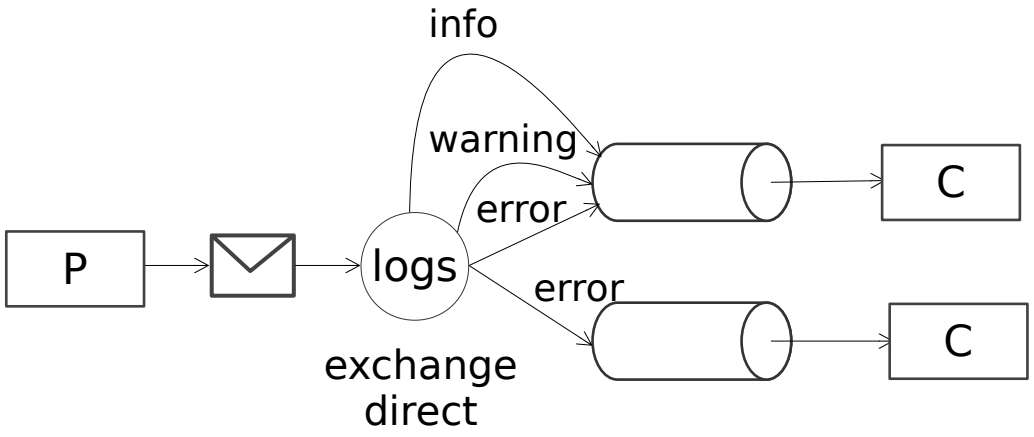


RabbitMQ

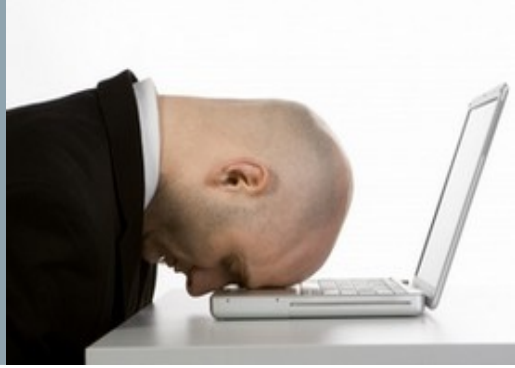


Exchange direct > Ejercicio 5

- Sistema de log con entrega selectiva según la severidad (e.g. ‘info’, ‘warning’, ‘error’) del mensaje con exchange direct



RabbitMQ

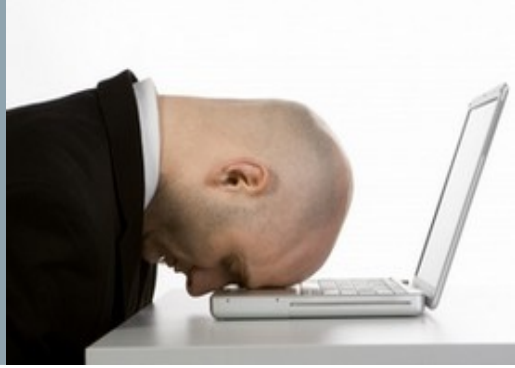


Exchange direct > Ejercicio 5 > productor

```
/* connect to channel */
...
var severity = process.argv.length > 3? process.argv[2]: 'info';
var msg = process.argv.length > 2? process.argv.slice(2).join(' '):
          'Hello world!';
channel.assertExchange('logs', 'direct', { durable: false });
channel.publish('logs', severity, Buffer.from(msg));
setTimeout( () => {
  con.close();process.exit(0);}, 500);
...
```

```
> node producer info Hello world
> node producer error My god
```

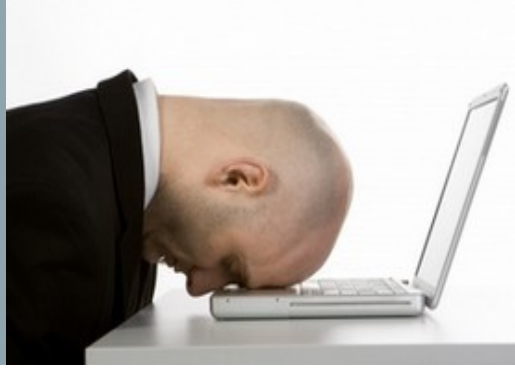
RabbitMQ



Exchange direct > Ejercicio 5 > consumidor

```
/* connect to channel */
...
channel.assertExchange('logs', 'direct', { durable: false });
channel.assertQueue('', { exclusive: true }, (err, q) => {
  if (err) console.log(err.stack);
  else {
    console.log('Waiting for messages');
    var severities = process.argv.length > 2?
                     process.argv.slice(2): ['info'];
    severities.forEach((severity) => {
      channel.bindQueue(q.queue, 'logs', severity);
    });
    channel.consume(q.queue, function (msg) {
      console.log('Received [' + msg.fields.routingKey + '] ' +
                  msg.content.toString());
    }, { noAck: true });
  }
});
```

RabbitMQ



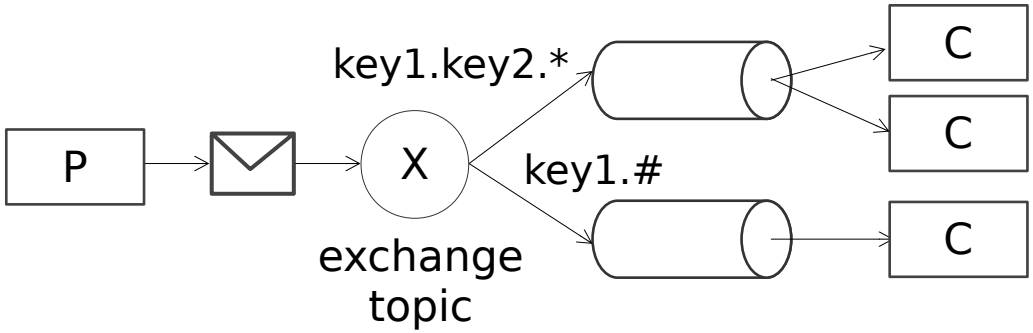
Exchange direct > Ejercicio 5 > consumidor

```
> node consumer info
> node consumer info warning error
```

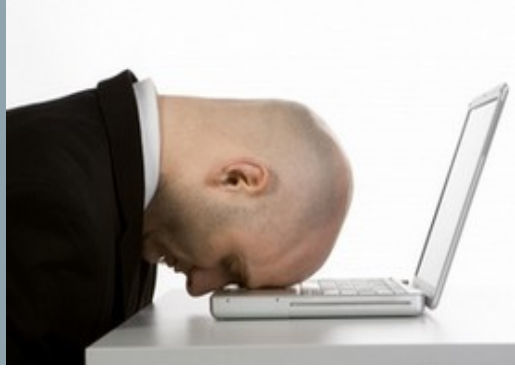
RabbitMQ

Exchange topic

- Añade capacidades multicriterio en enrutamiento
- Comprobación de claves de registro/enrutamiento, ahora con patrones (e.g. ‘hello.cruel.world’)
- Soporta wildcards ‘*’ (1 palabra), ‘#’ (0+ palabras)

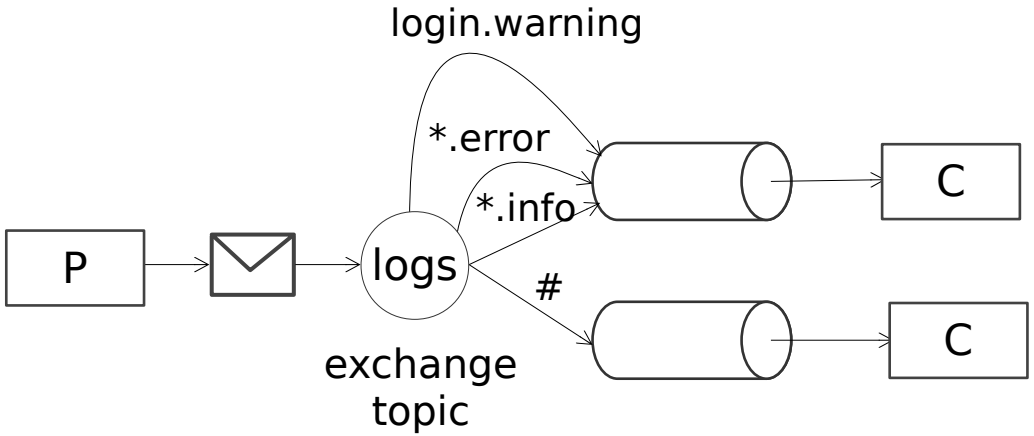


RabbitMQ

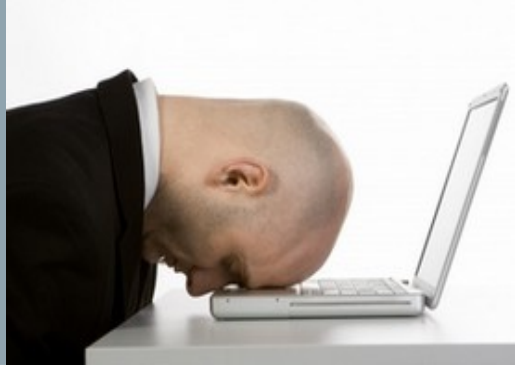


Exchange topic > Ejercicio 6

- Sistema de log con entrega selectiva según origen (e.g. ‘login’, ‘db’, etc.) y severidad (e.g. ‘info’, ‘warning’, ‘error’) del mensaje con exchange topic



RabbitMQ

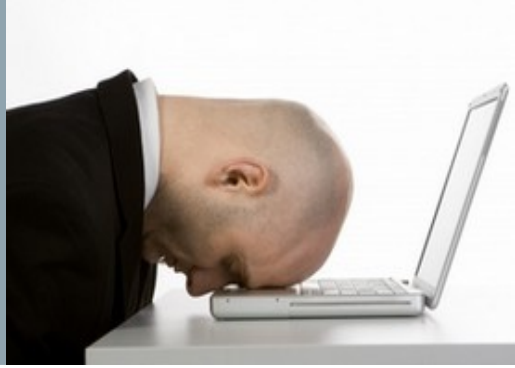


Exchange topic > Ejercicio 6 > productor

```
/* connect to channel */
...
var key = process.argv.length > 3? process.argv[2]: 'unknown.info';
var msg = process.argv.length > 2? process.argv.slice(2).join(' '):
          'Hello world!';
channel.assertExchange('logs', 'topic', { durable: false });
channel.publish('logs', key, Buffer.from(msg));
setTimeout( () => {
  con.close();process.exit(0);}, 500);
...
```

```
> node producer login.error Authenticate error
```

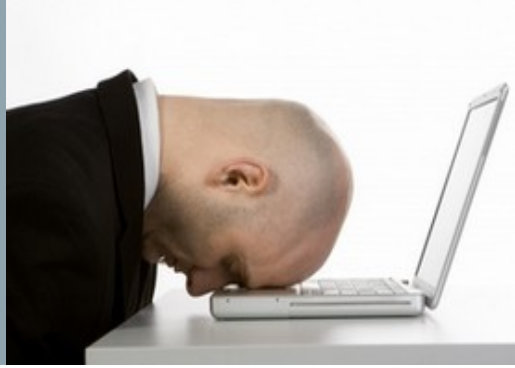
RabbitMQ



Exchange topic > Ejercicio 6 > consumidor

```
/* connect to channel */
...
channel.assertExchange('logs', 'topic', { durable: false });
channel.assertQueue('', { exclusive: true }, (err, q) => {
  if (err) console.log(err.stack);
  else {
    console.log('Waiting for messages');
    var keys = process.argv.length > 2? process.argv.slice(2):
               ['unknown.info'];
    keys.forEach((key) => {
      channel.bindQueue(q.queue, 'logs', key);
    });
    channel.consume(q.queue, function (msg) {
      console.log('Received [' + msg.fields.routingKey + '] ' +
                  msg.content.toString());
    }, { noAck: true });
  }
});
```

RabbitMQ



Exchange topic > Ejercicio 6 > consumidor

```
> node consumer 'login.error'
> node consumer '*.info'
> node consumer '*.*'
> node consumer '#'
```

RabbitMQ

Confirmaciones

- El bróker elimina de la cola el mensaje tras transmitirlo por el canal
- Si el consumidor no procesa el mensaje podría perderse
- Nivel adicional de seguridad con confirmaciones-acknowledgements
- Usamos la opción opts.noAck == false al suscribir el consumidor a la cola con channel.consume()

RabbitMQ

Confirmaciones > Ejemplo

```
/* connect to channel */
...
channel.assertQueue('hello-queue', { durable: false });
channel.consume('hello-queue', function (msg) {
  console.log('Received ' + msg.content.toString());
  /* make some process */
  channel.ack(msg);
}, { noAck: false });
...
```