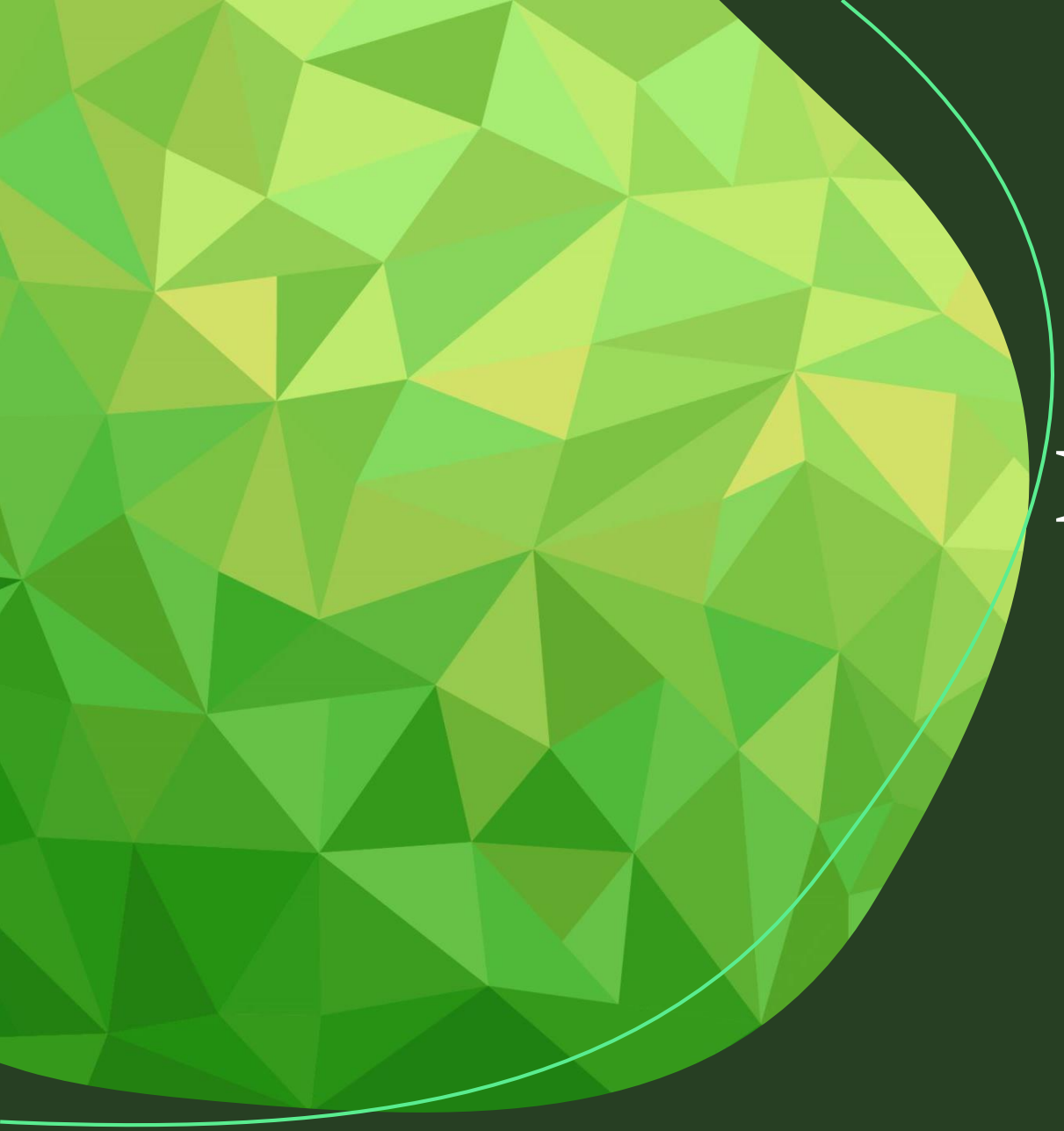


ZeroMQ – Publicador - Suscriptor

Material realizado por:

Juliana García Mogollón

Otro ejemplo por M. Curiel

¿Qué es ZeroMQ?

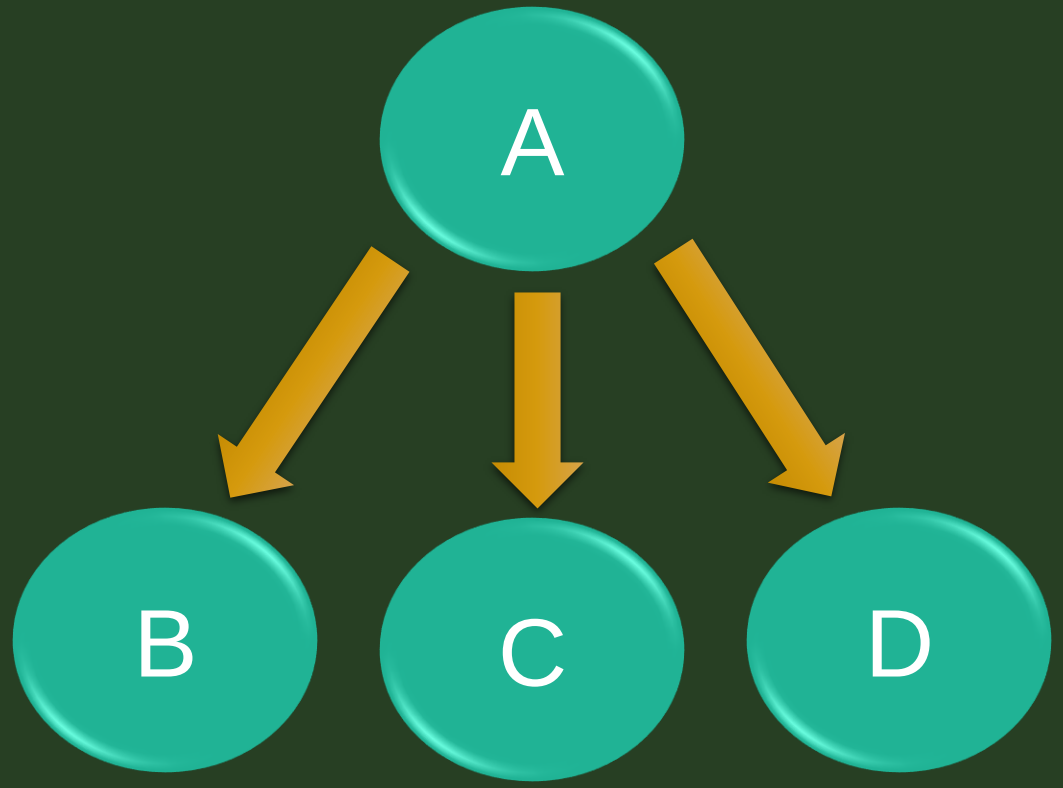
- ZeroMQ es un conjunto de librerías de mensajería asíncrona orientada al desarrollo de aplicaciones distribuida.
- It provides a message queue, but unlike message-oriented middleware, a ZeroMQ system can run without a dedicated message broker.
- Está disponible para distintos lenguajes de programación.

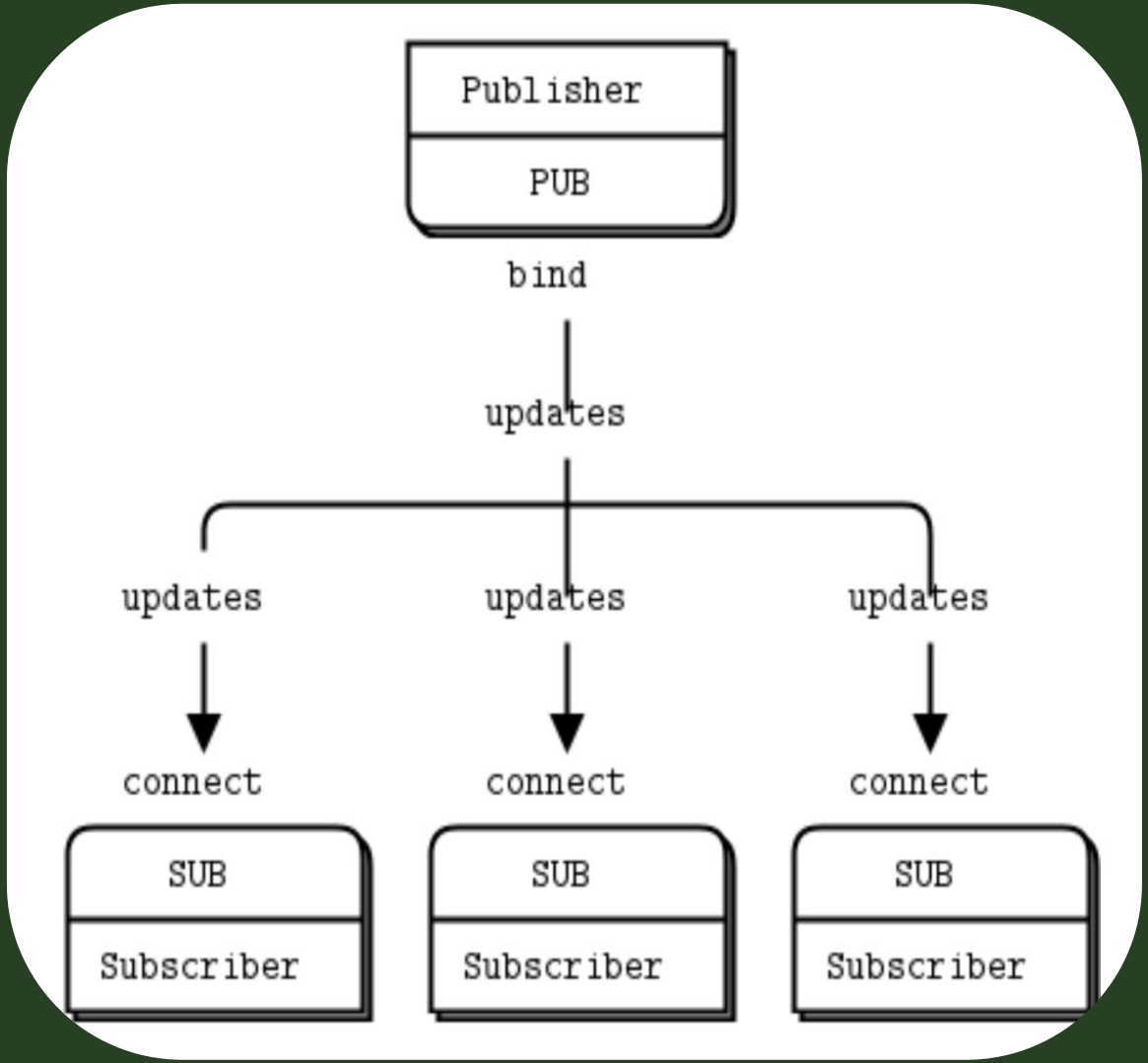
Patrones Básicos

- Request-Reply: usado en la comunicación tradicional cliente-servidor, similar a un RPC
- Publicador-Suscriptor: los clientes se suscriben a mensajes específicos que son publicados por servidores.
- Push-Pull (pipeline): un proceso desea “expulsar” resultados, con la certeza de que otros procesos desean consumir estos resultados.

Patrón Publicador - Suscriptor

- Patrón de distribución ***one-way-data.***
- Establece una conexión **asíncrona** entre un publicador y un conjunto de suscriptores.
- Es un patrón de distribución de datos ***one-to-many*** (uno a muchos).

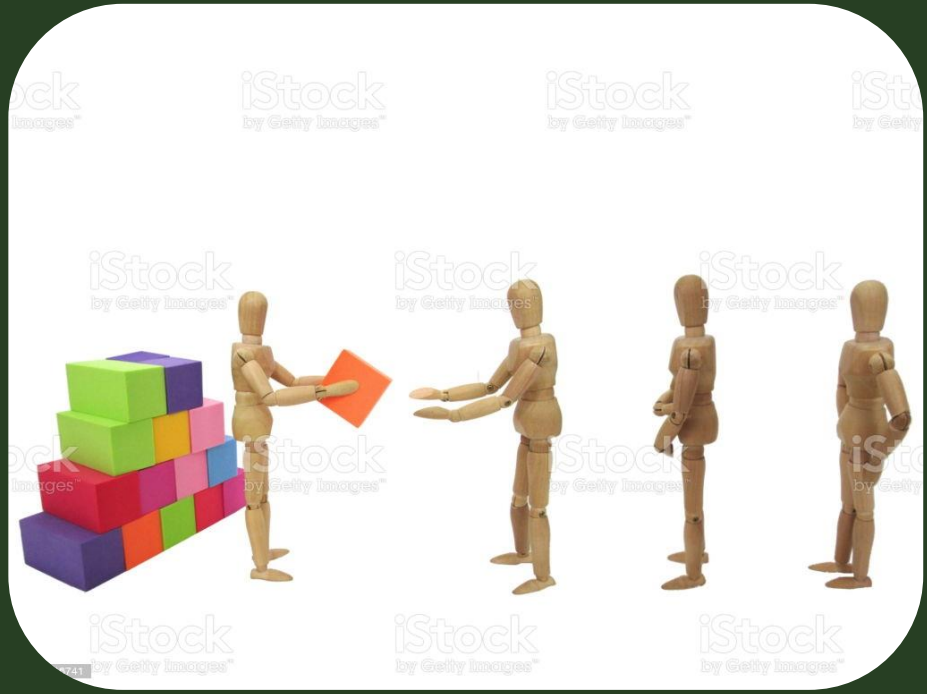




- El publicador **transmite mensajes sin fin;** no se preocupa por la existencia de los suscriptores, simplemente deja los mensajes.
- Cuando un suscriptor se **conecta** y todavía no hay un publicador, el suscriptor debe **esperar**.

Sockets

- PUB: Es el socket usado para **distribuir datos**. Este tipo de socket es unidireccional y sólo puede **enviar**.
- SUB: Es el socket usado para **suscribirse a cierto tipo de datos**. Este tipo de socket es unidireccional y sólo puede **recibir**.



Tópicos

- Los tópicos se expresan en un **arreglo de bytes** o **cadena de caracteres**.
- El publicador debe incluir el tópico en el **primer marco del mensaje**.
- Para recibir mensajes, el suscriptor debe haberse **suscrito anteriormente a un tópico**.

Pasos para establecer la conexión

Pasos	Publicador	Suscriptor
1	Establece el ambiente o contexto ZeroMQ.	Establece el ambiente o contexto ZeroMQ.
2	Crea un socket tipo PUB.	Crea un socket tipo SUB
3	Ata el socket a un puerto determinado.	Conecta el socket a un puerto determinado
4		Se suscribe por lo menos a un tópico.
5	Crea el mensaje.	
6	Transmite el mensaje.	
7		Recibe el mensaje.
8		Se asegura de que el mensaje pertenezca a uno de los tópicos suscritos.
9		Procesa el mensaje.

Demostración

```
//  Pubsub envelope publisher
//  Note that the zhelpers.h file also provides s_sendmore

#include "zhelpers.h"
#include <unistd.h>

int main (void)
{
    //  Prepare our context and publisher
    void *context = zmq_ctx_new ();
    void *publisher = zmq_socket (context, ZMQ_PUB);
    zmq_bind (publisher, "tcp://*:5563");

    while (1) {
        //  Write two messages, each with an envelope and content
        s_sendmore (publisher, "A");
        s_send (publisher, "We don't want to see this");
        s_sendmore (publisher, "B");
        s_send (publisher, "We would like to see this");
        sleep (1);
    }
    //  We never get here, but clean up anyhow
    zmq_close (publisher);
    zmq_ctx_destroy (context);
    return 0;
}
```

contexto

Se ata a un puerto

Se envía información sobre 2 tópicos, A y B

Se cierra el socket y se destruye el contexto

```
//  Pubsub envelope subscriber

#include "zhelpers.h"

int main (void)
{
    //  Prepare our context and subscriber
    void *context = zmq_ctx_new ();
    void *subscriber = zmq_socket (context, ZMQ_SUB);
    zmq_connect (subscriber, "tcp://localhost:5563");
    zmq_setsockopt (subscriber, ZMQ_SUBSCRIBE, "B", 1);

    while (1) {
        //  Read envelope with address
        char *address = s_recv (subscriber);
        //  Read message contents
        char *contents = s_recv (subscriber);
        printf ("[%s] %s\n", address, contents);
        free (address);
        free (contents);
    }
    //  We never get here, but clean up anyhow
    zmq_close (subscriber);
    zmq_ctx_destroy (context);
    return 0;
}
```

contexto

Conexión a puerto

Se suscribe a un tópico

Recibe los mensajes

Cierra el socket y destruye

```
//  Pubsub envelope subscriber

#include "zhelpers.h"

int main (void)
{
    //  Prepare our context and subscriber
    void *context = zmq_ctx_new ();
    void *subscriber = zmq_socket (context, ZMQ_SUB);
    zmq_connect (subscriber, "tcp://localhost:5563");
    zmq_setsockopt (subscriber, ZMQ_SUBSCRIBE, "A", 1);

    while (1) {
        //  Read envelope with address
        char *address = s_recv (subscriber);
        //  Read message contents
        char *contents = s_recv (subscriber);
        printf ("[%s] %s\n", address, contents);
        free (address);
        free (contents);
    }
    //  We never get here, but clean up anyhow
    zmq_close (subscriber);
    zmq_ctx_destroy (context);
    return 0;
}
```

Otro proceso se suscribe a otro tópico

El publicador pasa estructuras de datos

```
//  Pubsub envelope publisher

#include "empleados.h"
#include "zhelpers.h"
#include <unistd.h>

int main (void) {
  // InformaciÃ³n a Enviar a los suscriptores
    int rc;
    emple e, e1;
    e.edad = 18;
    e.ttrabajo= 5;
    strcpy(e.nombre, "Maria Perez");

    e1.edad = 30;
    e1.ttrabajo= 20;
    strcpy(e1.nombre, "Jose Gonzalez");

    //  Prepare our context and publisher
    void *context = zmq_ctx_new ();
    void *publisher = zmq_socket (context, ZMQ_PUB);
    zmq_bind (publisher, "tcp://*:5563");
```

El publicador pasa estructuras de datos

```
    while (1) {
        //  Write two messages, each with an envelope and content
        rc= zmq_send (publisher, "A", 1, ZMQ_SNDMORE);
        rc= zmq_send (publisher, (char *)&e, sizeof(e), 0);
        rc= zmq_send (publisher, "B", 1, ZMQ_SNDMORE);
        rc= zmq_send (publisher, (char *)&e1, sizeof(e1), 0);
        sleep (1);
    }
    //  We never get here, but clean up anyhow
    zmq_close (publisher);
    zmq_ctx_destroy (context);
    return 0;
}
```

El suscriptor

```
int main (void) {
  int error;
  emple e;

    //  Prepare our context and subscriber
    void *context = zmq_ctx_new ();
    void *subscriber = zmq_socket (context, ZMQ_SUB);
    zmq_connect (subscriber, "tcp://192.168.0.17:5563");
    zmq_setsockopt (subscriber, ZMQ_SUBSCRIBE, "A", 1);

    while (1) {
        //  Read envelope with address
        char *address = s_recv (subscriber);
        //  Read message contents
        error = zmq_recv(subscriber, (char *)&e, sizeof(e),0);
        assert(error != -1);
        printf ("[%s] %s\n", address, e.nombre);
        free (address);
}
    //  We never get here, but clean up anyhow
    zmq_close (subscriber);
    zmq_ctx_destroy (context);
    return 0;
}
```

La deje igual para recibir el tópico

Se recibe la estructura de datos.

Links importantes

- Iniciación ZeroMQ: https://zeromq.org/get-started/
- Descargar ZeroMQ: https://zeromq.org/download/
- Guía: http://zguide.zeromq.org/java:all
- Patrones de mensajería: https://zeromq.org/socket-api/#messaging-patterns

Muchas Gracias !