

Servidor Middleware java en AWS

Alumnos:

Ayrton Coronado - acoronadoh@uni.pe

Flores Huamani - lfloresh@uni.pe

Barrientos Porras - herlees.barrientos.p@uni.pe

Universidad Nacional de Ingeniería, Facultad de Ciencias,

Curso:

CC462 Sistemas concurrentes y distribuidos

Examen Parcial

Resumen

El presente trabajo es la implementación y deploy de 4 tipos de servidor Middleware Orientado a Mensajes(MOM) Java en AWS que permite la comunicación entre clientes de sistemas heterogéneos que actúan como consumer y producer implementados en lenguajes C++ y Python, con sistemas operativos Windows y GNU/Linux mediante threads y sockets.

Palabras clave: middleware, sockets, cliente, servidor, threads, java, C++, python.

Contenidos

1	Introducción	2
2	Marco teórico	2
2.1	Creación de threads	2
2.2	Cliente-servidor mediante sockets	5
2.3	Los Middlewares desarrollados	
3	Metodología	7
3.1	Manejo de threads	7
3.2	Comunicación entre clientes	7
3.3	Deploy de un Middleware	7
4	Resultados y discusiones	7
5	Conclusiones	10
6	Anexo Código	11
7	Anexo Documentation	11

01. Introducción

Un Middleware es una capa de una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red). El middleware abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas.

02. Marco Teórico

a. Creación de Threads en java.

Hay dos formas de crear un nuevo hilo de ejecución. Una es declarar una clase como una subclase de Thread. Esta subclase debería anular el método run de la clase Thread. Entonces se puede asignar e iniciar una instancia de la subclase. Por ejemplo, un hilo que calcula números primos mayores que un valor establecido podría escribirse de la siguiente manera:

```
class PrimeThread extends Thread {  
  
    long minPrime;  
  
    PrimeThread(long minPrime) {  
  
        this.minPrime = minPrime;  
  
    }  
  
    public void run() {  
  
        // compute primes larger than minPrime  
  
        ...  
  
    }  
  
}
```

El siguiente código crearía un hilo y lo comenzaría a ejecutar:

```
PrimeThread p = nuevo PrimeThread (143);  
  
p.start ();
```

La otra forma de crear un hilo es declarar una clase que implemente la interfaz Runnable. Esa clase luego implementa el método run. Luego se puede asignar una instancia de la clase, pasarla como argumento cuando creamos Thread e iniciarlo. El mismo ejemplo en este otro estilo tiene el siguiente aspecto:

```
class PrimeRun implements Runnable {
```

```

    long minPrime;

    PrimeRun(long minPrime) {

        this.minPrime = minPrime;

    }

    public void run() {

        // compute primes larger than minPrime

        ...

    }

}

```

El siguiente código crea un thread y lo inicia:

```

PrimeRun p = new PrimeRun(143);

new Thread(p).start();

```

b. Cliente-Servidor mediante sockets

Para crear una conexión por sockets entre cliente y servidor usando threads :

En la clase servidor como Middleware, implementamos en java

Class Server implements Runnable {

```

    ServerSocket serverSocket;

```

```

    Socket client;

```

```

    Public static final int SERVERPORT = 32000;

```

```

    ...

```

```

    public void run(){

```

```

        try{

```

```

            #Diferentes conexiones

```

```

            while(true)

```

```

                serverSocket = new ServerSocket(SERVERPORT);

```

```

                client = serverSocket.accept();

```

```

                ...

```

```

            }

```

```

        }catch(Exception e){

```

```

            System.out.println("Error: " + e.getMessage());

```

```

        }

```

```

    }
}

```

En la clase cliente como producer y consumer, implementamos en C++ y python.

```

#C++

#include <iostream>

#include <winsock2.h>

using namespace std;

class Client{
public:
    WSADATA WSADATA;
    SOCKET server;
    SOCKADDR_IN addr;

    Client() {
        cout<<"Conectando al servidor..."<<endl<<endl;
        WSStartup(MAKEWORD(2,0), &WSADATA);
        server = socket(AF_INET, SOCK_STREAM, 0);
        addr.sin_addr.s_addr = inet_addr("13.59.160.76");
        addr.sin_family = AF_INET;
        addr.sin_port = htons(32000);
        connect(server, (SOCKADDR *)&addr, sizeof(addr));
        cout << "Conectado al Servidor!" << endl;
    }
    ...
};

int main() {
    Client *Cliente = new Client();
    ...
}

```

#Python

```
import sys
import socket as sk

host = "13.59.160.76"

port = 32000

sCliente = sk.socket()

sCliente.connect((host, port))

print("Conectado")

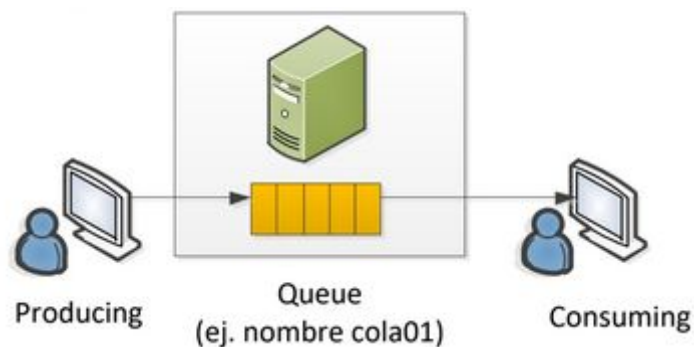
...

sCliente.close()

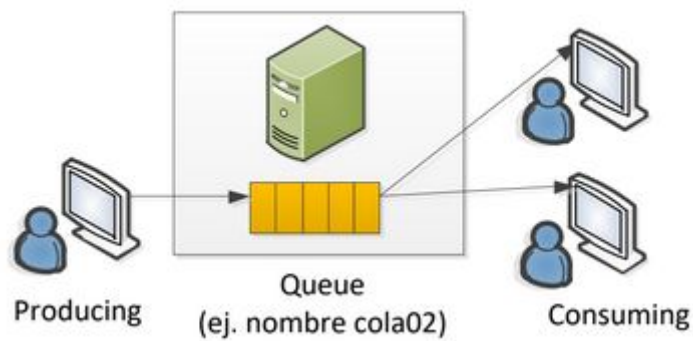
print("Terminado")
```

2.3. Los Middleware Desarrollados

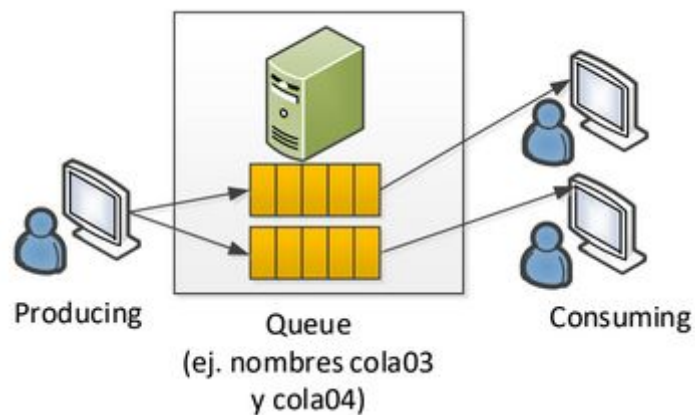
- i. **Middleware basado con una Cola** , En este Middleware está un servidor java que espera los usuarios Productor en python y el Consumidor en c++, donde administra sus mensajes con una cola.



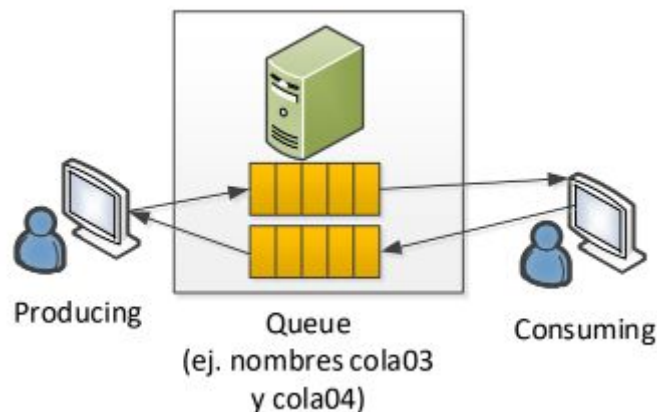
- ii. **Middleware con varios Consuming**, En este Middleware está un servidor java que espera los usuarios Productor en c++ y el Consumidor en python , la cual difiere de la anterior al esta poseer varios consumidores.



- iii. **Middleware con varias Colas**, En este Middleware está un servidor java que espera los usuarios Productor en c++ y los Consumidores en python , además de haber más de una cola de mensajes para sus respectivos Consumidores.



- iv. **Middleware basado en RPC**, En este Middleware está un servidor java que espera los usuarios Productor en c++, los Consumidores en python, varias colas y manejan el envío y recibo de mensajes de los respectivos consumidores y productores a través de las colas



03. Metodología

3.1 Manejo de threads

Para la implementación del middleware se requiere trabajar estrictamente con threads ya que se generan procesos independientes para el correcto funcionamiento.

Por ejemplo:

- . Proceso principal para iniciar el servidor y que reciba y envíe información.
- . Procesos principales para iniciar cada cliente, que reciban y envíen información.
- . Proceso para aceptar nuevas conexiones por socket en el servidor.
- . Proceso para cada conexión por socket creada por cada cliente.
- . Proceso para cada objeto nuevo creado con comportamiento independiente : minas, etc

3.2 Comunicación entre clientes

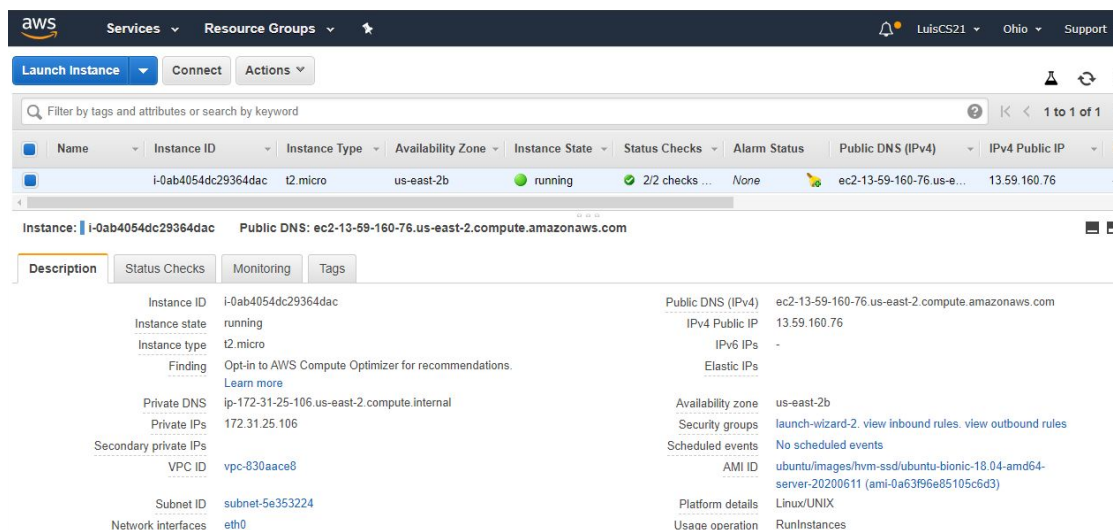
Para lograr que diferentes clientes tanto consumer como producer se comuniquen de forma asíncrona, el servidor encola los mensajes que recibe de producer y luego desencola cuando el consumer lo requiera.

3.3 Deploy del servidor middleware

Necesitamos tener una cuenta en AWS, luego creamos una instancia EC2 (con IP pública habilitada) como servidor virtual en la nube de AWS, donde clonamos el repositorio del grupo y ejecutamos nuestro servidor Middleware, al que accederemos remotamente con su IP pública.

04. Results and Discussion

Instancia AWS



The screenshot displays the AWS Management Console interface for an EC2 instance. The instance is named 'i-0ab4054dc29364dac', is of type 't2.micro', and is currently in a 'running' state. It is located in the 'us-east-2b' availability zone. The public DNS is 'ec2-13-59-160-76.us-east-2.compute.amazonaws.com' and the public IP is '13.59.160.76'. The instance is associated with the 'launch-wizard-2' security group and the 'ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-20200611' AMI. The platform details show 'Linux/UNIX' and 'RunInstances' usage operation.

Category	Value
Instance ID	i-0ab4054dc29364dac
Instance state	running
Instance type	t2.micro
Finding	Opt-in to AWS Compute Optimizer for recommendations. Learn more
Private DNS	ip-172-31-25-106.us-east-2.compute.internal
Private IPs	172.31.25.106
Secondary private IPs	
VPC ID	vpc-830aace8
Subnet ID	subnet-5e353224
Network interfaces	eth0
Public DNS (IPv4)	ec2-13-59-160-76.us-east-2.compute.amazonaws.com
IPv4 Public IP	13.59.160.76
IPv6 IPs	-
Elastic IPs	
Availability zone	us-east-2b
Security groups	launch-wizard-2 . view inbound rules . view outbound rules
Scheduled events	No scheduled events
AMI ID	ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-20200611 (ami-0a63f96e85105c6d3)
Platform details	Linux/UNIX
Usage operation	RunInstances

servidor middleware

```
ubuntu@ip-172-31-25-106:~/CC462Concurrentes/Parcial/ServerI$ java Server
Esperando clientes ...
New client connected 190.234.38.130
New client connected 190.237.28.194
```

i. Middleware basado con una Cola

Producer

```
TRIBUIDOS-20-1/GIT/Parcial/ServerI$ python3 productor.py
Conectado
Texto para enviar: mensaje 1 prueba
Texto para enviar: █
```

Consumer

```
PS C:\Users\Usuario\Desktop\CC343\CC462Concurrentes\Parcial\ServerI>
Conectando al servidor...

Conectado al Servidor!
Enter para recibir:
Recibido: mensaje 1 prueba
Enter para recibir:
Recibido: No hay mas mensajes en la cola!
Enter para recibir: █
```

ii. Middleware con varios Consuming

Producer

```
PS C:\Users\Usuario\Desktop\CC343\CC462Concurrentes\Parcial\ServerII> .\a.exe
Conectando al servidor...

Conectado al Servidor!
Texto a enviar: hola
Texto a enviar: soy el productor
Texto a enviar: █
```

Consumer 1

```
TRIBUIDOS-20-1/GIT/Parcial/ServerII$ python3 consumidor.py
Conectado
Enter para recibir elemento:
Texto recibido: hola
Enter para recibir elemento:
Texto recibido: soy el productor
Enter para recibir elemento:
Texto recibido: No hay mas elementos en la cola!
Enter para recibir elemento: █
```

Consumer2

```
TRIBUIDOS-20-1/GIT/Parcial/ServerII$ python3 consumidor.py
Conectado
Texto recibido: hola
Texto recibido: soy el productor
Enter para recibir elemento: █
```


iii. Middleware con varias Colas

Producer

```
PS C:\Users\Usuario\Desktop\CC343\CC462Concurrentes\Parcial\
Conectando al servidor...

Conectado al Servidor!

A que consumidor enviar?:2
Texto a enviar: hola consumer 2
hola consumer 2
A que consumidor enviar?:3
Texto a enviar: hola consumer 3
hola consumer 3
A que consumidor enviar?:
```

Consumer 1

```
TRIBUIDOS-20-1/GIT/Parcial/ServerIII$ python3 consumidor.py
Ingrese un numero identificador: 2
Conectado
Enter para recibir elemento:
Texto recibido: hola consumer 2
Enter para recibir elemento:
Texto recibido: No hay mas elementos en la cola!
Enter para recibir elemento:
```

Consumer 2

```
TRIBUIDOS-20-1/GIT/Parcial/ServerIII$ python3 consumidor.py
Ingrese un numero identificador: 3
Conectado
Enter para recibir elemento:
Texto recibido: hola consumer 3
Enter para recibir elemento:
Texto recibido: No hay mas elementos en la cola!
Enter para recibir elemento:
```

iv. Middleware basado en RPC

Producer 1

```
PS C:\Users\Usuario\Desktop\CC343\CC462Concurrentes\Par
cial\ServerIV> .\a.exe
Conectando al servidor...

Conectado al Servidor!
Ingrese un numero identificador:1
Consumir(1) o Enviar(2): 2
Ingrese texto a enviar: hola consumer 1
Consumir(1) o Enviar(2): 1
Recibido: Hola Producer
Consumir(1) o Enviar(2):
```

Producer 2

```
PS C:\Users\Usuario\Desktop\CC343\CC462Concurrentes\Parcial\ServerIV> .\a.exe
Conectando al servidor...

Conectado al Servidor!
Ingrese un numero identificador:2
Consumir(1) o Enviar(2): 2
Ingrese texto a enviar: hola consumer 2
Consumir(1) o Enviar(2): 1
Recibido: Hola Producer 2
Consumir(1) o Enviar(2): █
```

Consumer 1

```
TRIBUIDOS-20-1/GIT/Parcial/ServerIV$ python3 consumidor.py
Ingrese un numero identificador: 1
Conectado
Consumir(1) o Enviar(2): 1
Texto recibido:  Hola consumer 2
Consumir(1) o Enviar(2): 1
Texto recibido:  hola consumer 1
Consumir(1) o Enviar(2): 2
Ingrese el texto a enviar: Hola Producer
Consumir(1) o Enviar(2): █
```

Consumer 2

```
TRIBUIDOS-20-1/GIT/Parcial/ServerIV$ python3 consumidor.py
Ingrese un numero identificador: 2
Conectado
Consumir(1) o Enviar(2): 1
Texto recibido:  hola consumer 2
Consumir(1) o Enviar(2): 1
Texto recibido:  No hay mas elementos en la cola!
Consumir(1) o Enviar(2): 2
Ingrese el texto a enviar: Hola Producer 2
Consumir(1) o Enviar(2): █
```

05. Conclusiones

- El servidor Middleware resulta muy útil en la comunicación entre sistemas heterogéneos, ya que abstrae la complejidad de las redes adyacentes .
- El uso de colas permite una comunicación asíncrona entre consumer y producer.
- El uso de sockets es una forma fácil de comunicación remota.

- Desplegar el Middleware en la nube permite una conexión global y veloz entre clientes.

06. Anexo Código

- <https://github.com/lfloresh/CC462Concurrentes>

07. Anexo Documentación

- <https://docs.oracle.com/en/java/javase/14/>
 - <https://www.youtube.com/watch?v=hPDL9yIlZEK>
 - <https://www.redhat.com/es/topics/middleware/what-is-middleware>
-