Instituto Politécnico Nacional

Escuela Superior de Cómputo

Asignatura:

Desarrollo de sistemas distribuidos

Grupo:4CM3

Tarea 4. Token Ring

Alumno: Garcia Garcia Rafael

Profesor: Carlos Pineda Guerrero

Capturas de la ejecución del programa:

Lo primero que tenemos que realizar es la transcripción de nuestros algoritmos a nuestro programa **Token.java** y compilarlo para ver que efectivamente no tenemos ningún error de compilación esta parte se puede ver en la **imagen 1**.

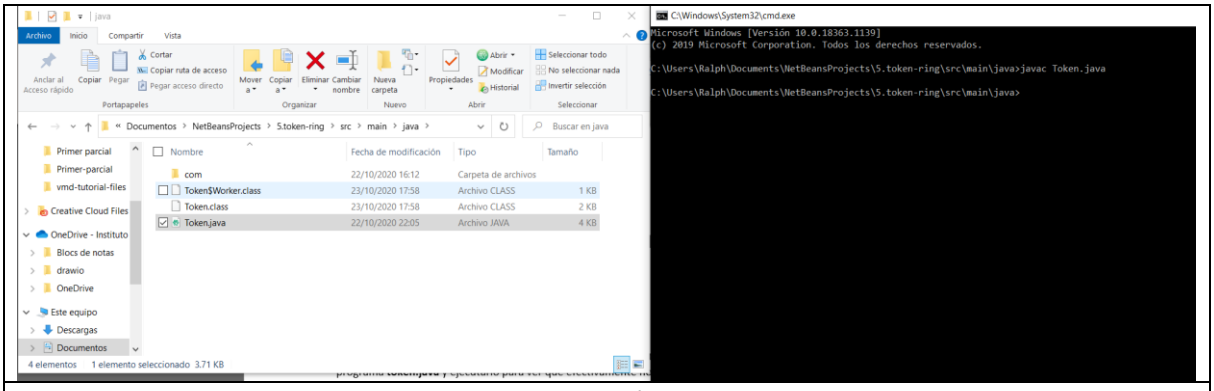

Imagen 1. Compilación del programa

Una vez generados nuestros archivos **.class** vamos a mandarlos a cada una de nuestras maquinas virtuales además de que vamos a tener que instalar java en cada una de ellas para ejecutar los programas. Lo primero a realizar en esta parte es crear las maquinas virtuales en **Azure** con **Ubuntu,** posterior a esto vamos a abrir el puerto 50,000 en **la imagen 2** se muestran las maquinas creadas y en la **imagen 3** se muestra la apertura del puerto 50,000

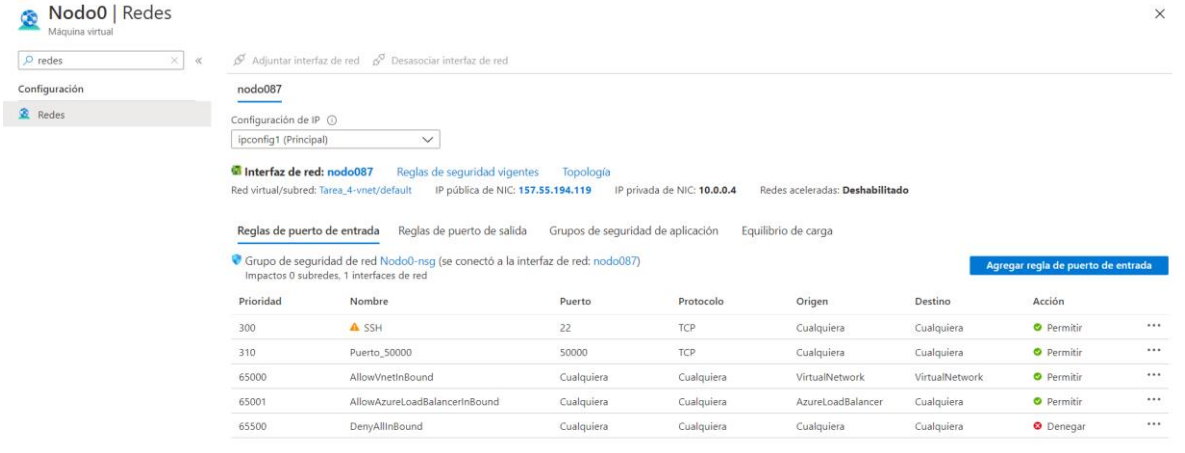

Imagen 2. Creación del nodo 0 al 3 en Microsoft Azure

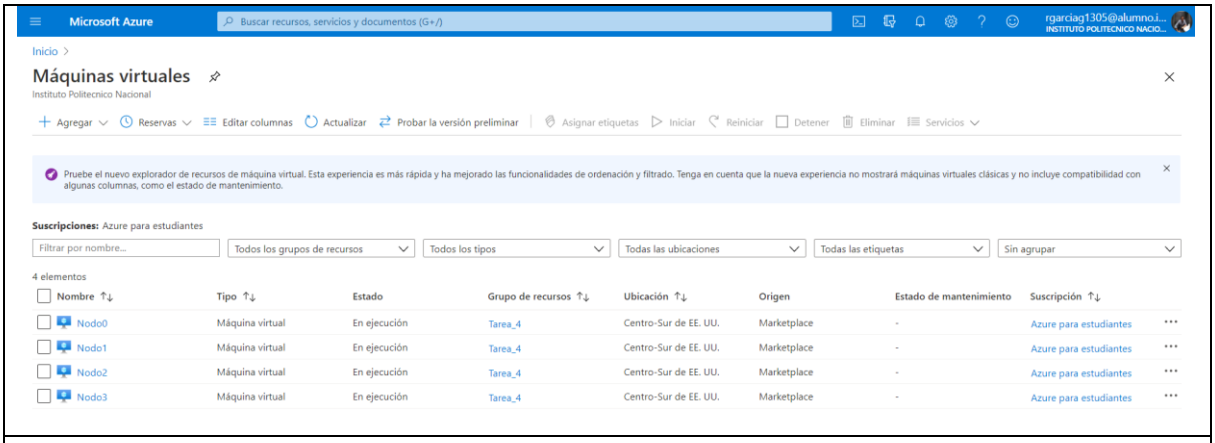

Imagen 3. Apertura del puerto 50,000 en un nodo

Ya que tenemos nuestras máquinas virtuales ejecutándose en la nube vamos a tener que transferir nuestros archivos **.class** en las máquinas además de que tenemos que instalar **Java** en cada una de ellas para que los programas puedan ejecutarse sin ningún problema. En la **imagen 4** se muestra la transferencia de los archivos **.class** a una maquina virtual mediante nuestro programa **PuTTY**. Para esta parte tenemos que conseguir la IP pública de nuestras máquinas para conectarnos remotamente. Haremos este proceso para cada una de las máquinas virtuales que tenemos en nuestra nube.

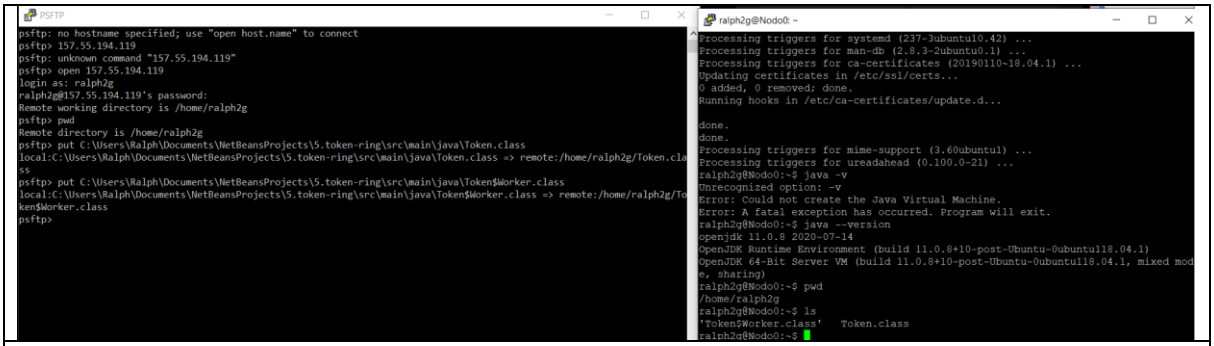

Imagen 4. Transferencia de archivos .class a una maquina virtual

Para ejecutar el programa instalaremos **Java** como se mencionó anteriormente en cada una de nuestras máquinas este proceso se ilustra en la **imagen 5**.

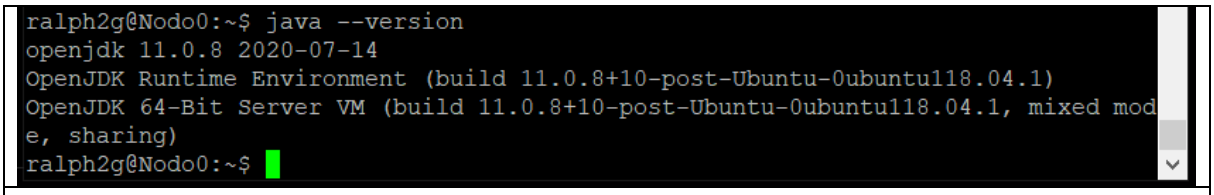

Imagen 5. Instalación de Java en una máquina virtual

Terminado estos dos puntos clave vamos a tener que ejecutar el programa en nuestras máquinas mandando como primer parámetro el nodo de la máquina virtual y como segundo parámetro la IP

del nodo que le sigue, así posteriormente podremos hacer que se comuniquen una con otras remotamente en la imagen 6 se ve como se ejecuta el programa realizado en cada nodo mandando como parámetro el número de nodo actual y la IP del nodo que sigue, en este caso cuando se ejecuta el programa se queda en un estado de espera cada nodo hasta que se termine de enlazar la comunicación con el nodo 0.

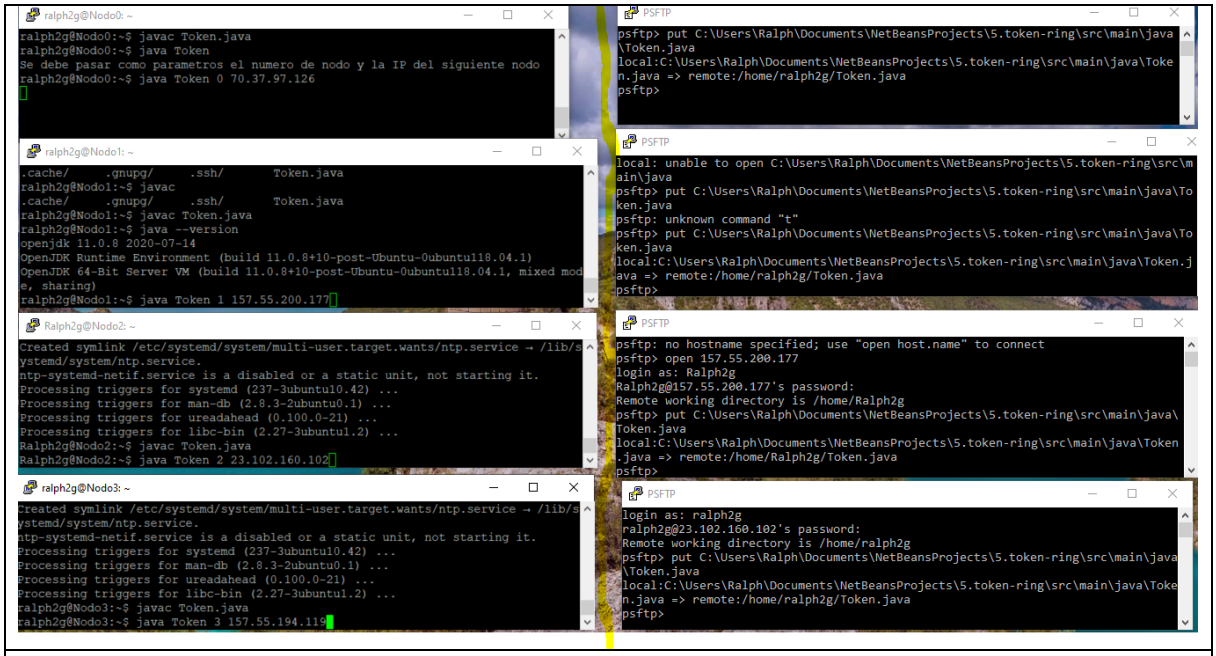

Imagen 6. Inicialización del programa en cada uno de los nodos

En la imagen 7 podemos ver que sucede cuando se termina de ejecutar el programa en cada uno de nuestros nodos, la salida que se muestra en nuestros nodos es el token incrementado de uno en uno en cada nodo, además se encuentra una observación bastante lógica en la ejecución la cual se ve cuando se interrumpe el programa en el nodo número 3, este nodo alcanza a pasar el token aumentado al nodo que le sigue pero en el momento en que el nodo 2 quiere pasar el token al nodo 3 como este ya no está en funcionamiento produce una excepción la cuál posteriormente se ve reflejado en el nodo 0 y 1.

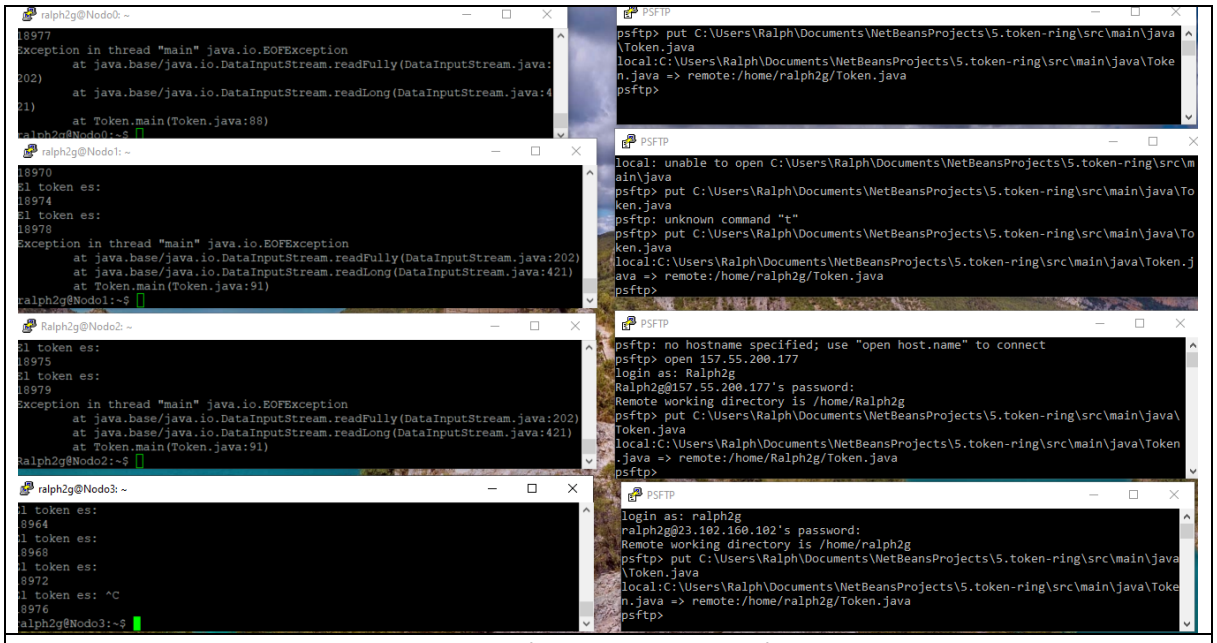

Imagen 7. Interrupción de la comunicación entre nodos

Como **conclusión** podemos ver que nuestro programa funcionó de manera correcta, logrando una conexión entre cada una de nuestras máquinas virtuales en la nube conectando los nodos con las ip's y puertos de nuestras máquinas. En realidad, me asombró la velocidad en que trabajaron las maquinas ya que se logró incrementar el token **18,979** veces mientras estuvieron comunicándose. Creo que esta práctica es la introducción perfecta al siguiente tema que es el "**Algoritmo de elección del abusón"** y después de haber hecho esto me siento mucho más preparado para la siguiente práctica.

Explicación del código del programa:

Nuestro programa se ejecuta sobre la clase **Token** la cual va a tener un funcionamiento de cliente y de servidor al mismo tiempo ya que va a recibir respuestas de un cliente-servidor n-1 va a aumentar el valor que reciba en 1 y lo mandará a un cliente-servidor n+1. Este proceso se realizará conectando cada servidor uno con otro en forma de anillo para que el token se pase indefinidamente entre ellos.

Lo primero que se hará es declarar algunas variables que compartirán nuestros servidores como atributos de la clase Token y además se declara una susubclase "Worker" la cual será el hilo que esperará la conexión de un nodo para asignar el buffer de entrada.

Para la parte del **main** tenemos que tener claro que recibiremos 2 parámetros, para ello vamos a poner una condicional para que esto suceda y en caso contrario el usuario introduzca el número de nodo y la IP del nodo que le sigue de manera correcta. Cuando tengamos la IP y el nodo vamos a inicializar nuestra clase "Worker" la cual va a ser un hilo y para esto lo haremos con el método ***start*** una vez tengamos la conexión vamos a realizar nuevamente una conexión de salida al puerto 50,000 para poder tener nuestro buffer de salida. Para controlar errores y poder garantizar que estas conexiones de entrada y salida se hagan de manera correcta vamos a ejecutar el método ***join.*** Finalmente, teniendo nuestro buffer de salida y entrada vamos a revisar si somos el nodo 0, si

este el caso vamos a tener que desactivar nuestra bandera “primera_vez” para que en el for en el que nos encontramos podamos pasar a leer nuestro buffer de entrada de valores y así obtener el token que anteriormente pasó en nuestra red, aumentar este token en 1 y mandarlo al nodo siguiente. En caso de que no sea el nodo 0 inmediatamente vamos a esperar a que un nodo nos mande un token en nuestro buffer de entrada para hacer la suma en el token y mandarlo al siguiente nodo.

Este proceso se hará de manera infinita hasta que encuentre una interrupcion en alguno de nuestros servidores.

Código del programa:

```
1.  /*
2.   * To change this license header, choose License Headers in Project Properties.
3.   * To change this template file, choose Tools | Templates
4.   * and open the template in the editor.
5.   */
6.  import java.net.Socket;
7.  import java.net.ServerSocket;
8.  import java.io.DataOutputStream;
9.  import java.io.DataInputStream;  /**
10. /**
11.  *
12.  * @author Ralph
13.  */
14. public class Token {
15.   static DataInputStream entrada;
16.   static DataOutputStream salida;
17.   static boolean primera_vez = true;
18.   static String ip;
19.   static long token = 0;
20.   static int nodo;
21.
22.   static class Worker extends Thread{
23.     public void run(){
24.         //=====IMPLEMENTACIÓN ALGORITMO 1
25.         try{
26.         //1. Declarar la variable servidor de tipo ServerSocket,
27.         //2. Asignar a la variable servidor el objeto: new ServerSocket(50000)
28.         ServerSocket servidor = new ServerSocket(50000);
29.         //3. Declarar la variable conexion de tipo Socket.
30.         //4. Asignar a la variable conexion el objeto servidor.accept().
31.         Socket conexion = servidor.accept();
32.         //5. Asignar a la variable entrada el objeto new DataInputStream(conexion.
    getInputStream()).
33.         entrada = new DataInputStream(conexion.getInputStream());
34.
35.         }catch(Exception e){
36.             e.printStackTrace();
37.         }
38.     }
39.   }
40.
41.   public static void main(String[] args) throws Exception{
42.     if (args.length != 2){
43.       System.err.println("Se debe pasar como parametros el numero de nodo y la IP
    del siguiente nodo");
44.       System.exit(1);
```

```
45.     }
46.
47.     nodo = Integer.valueOf(args[0]);  // el primer parametro es el numero de nodo

48.     ip = args[1];  // el segundo parametro es la IP del siguiente nodo en el anill
    o
49.     //============IMPLEMENTACIÓN Algoritmo 2
50.     //1. Declarar la variable w de tipo Worker.
51.     //2. Asignar a la variable w el objeto new Worker().
52.     Worker w = new Worker();
53.     //3. Invocar el método w.start().
54.     w.start();
55.     //4. Declarar la variable conexion de tipo Socket. Asignar null a la variable
    conexion.
56.     Socket conexion = null;
57.     //5. En un ciclo:
58.     for(;;){
59.     //5.1 En un bloque try:
60.       try{
61.         //5.1.1 Asignar a la variable conexion el objeto Socket(ip,50000).
62.         conexion = new Socket(ip,50000);
63.         //5.1.2 Ejecutar break para salir del ciclo.
64.         break;
65.       }catch(Exception e){
66.           //5.2 En el bloque catch:
67.           //5.2.1 Invocar el método Thread.sleep(500).
68.           Thread.sleep(500);
69.       }
70.     }
71.     //6. Asignar a la variable salida el objeto new DataOutputStream(conexion.getO
    utputStream()).
72.     salida = new DataOutputStream(conexion.getOutputStream());
73.     //7. Invocar el método w.join().
74.     w.join();
75.     //8. En un ciclo:
76.     for(;;){
77.         //8.1 Si la variable nodo es cero:
78.         if (nodo == 0){
79.             //8.1.1 Si la variable primera_vez es true:
80.             //8.1.1.1 Asignar false a la variable primera_vez.
81.             if(primera_vez == true)
82.                 primera_vez = false;
83.             //8.1.2 Si la variable primera_vez es false:
84.             //8.1.2.1 Asignar a la variable token el resultado del método entrada.
    readLong().
85.             else
86.                 token = entrada.readLong();
87.         }else{//8.2 Si la variable nodo no es cero:
88.             //8.2.1 Asignar a la variable token el resultado del método entrada.re
    adLong().
89.             token = entrada.readLong();
90.         }
91.         //8.3 Incrementar la variable token.
92.         token++;
93.         //8.4 Desplegar el valor de la variable token.
94.         System.out.println("El token es: ");
95.         System.out.println(token);
96.         //8.5 Invocar el método salida.writeLong(token).
97.         salida.writeLong(token);
98.
99.     }
```

```
100.
101.         }
102.       }
```