Tarea 1

Sistemas Distribuidos

Camilo Núñez - Hugo Sepúlveda

Universidad Técnica Federico Santa María

5 de octubre de 2019

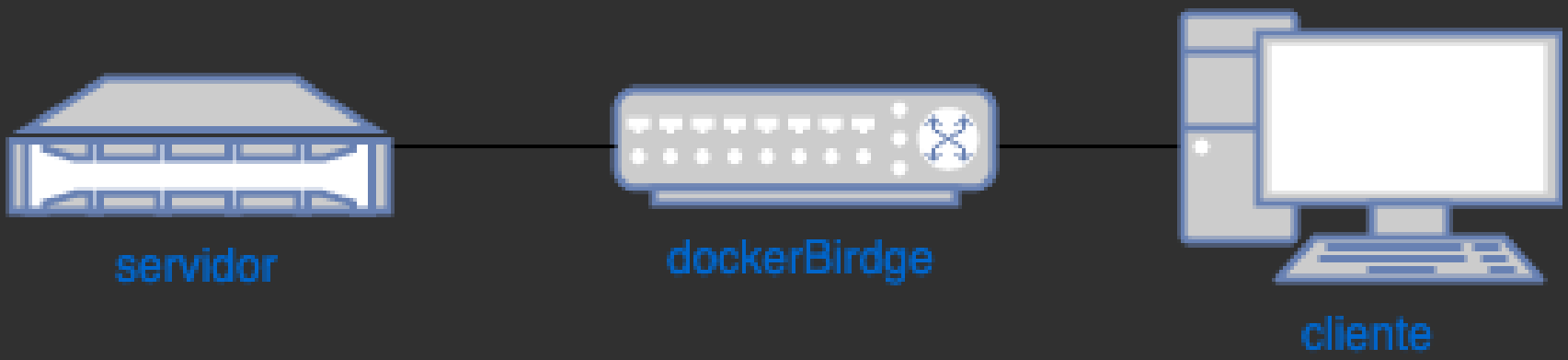
Arquitectura y servicios

Arquitectura y servicios

- Arquitectura: Docker
- Programación : Python

Actividad 1

Topología



Servidor - Código

```
class ServerThread(Thread):
...
    def run(self):

        # Recibe peticion de conexion
        data = self.conn.recv(bufferSize).decode("utf-8")
        logging.info(data)
...

        while True:
            # Recibe mensaje
            data = conn.recv(bufferSize).decode("utf-8")
            logging.info(data)
            self.conn.send(data.encode("utf-8"))

serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("Socket de Servidor iniciado.")
serverSocket.bind( (IP, PORT) )
...
while i < 100:
    (conn, (ip, port)) = serverSocket.accept()
...

# se cierra servidor despues de 100 clientes
serverSocket.close()
```

Servidor - Características

- Se utilizó la librería `logging` para guardar los output como archivos.
- El objetivo del servidor es generar una conexión TCP/5000 por cada cliente, utilizando un Thread por cliente conectado.
- Se fijó un máximo de 100 clientes para conectar, además se aceptan 100 conexiones simultáneas.

Cliente - Código

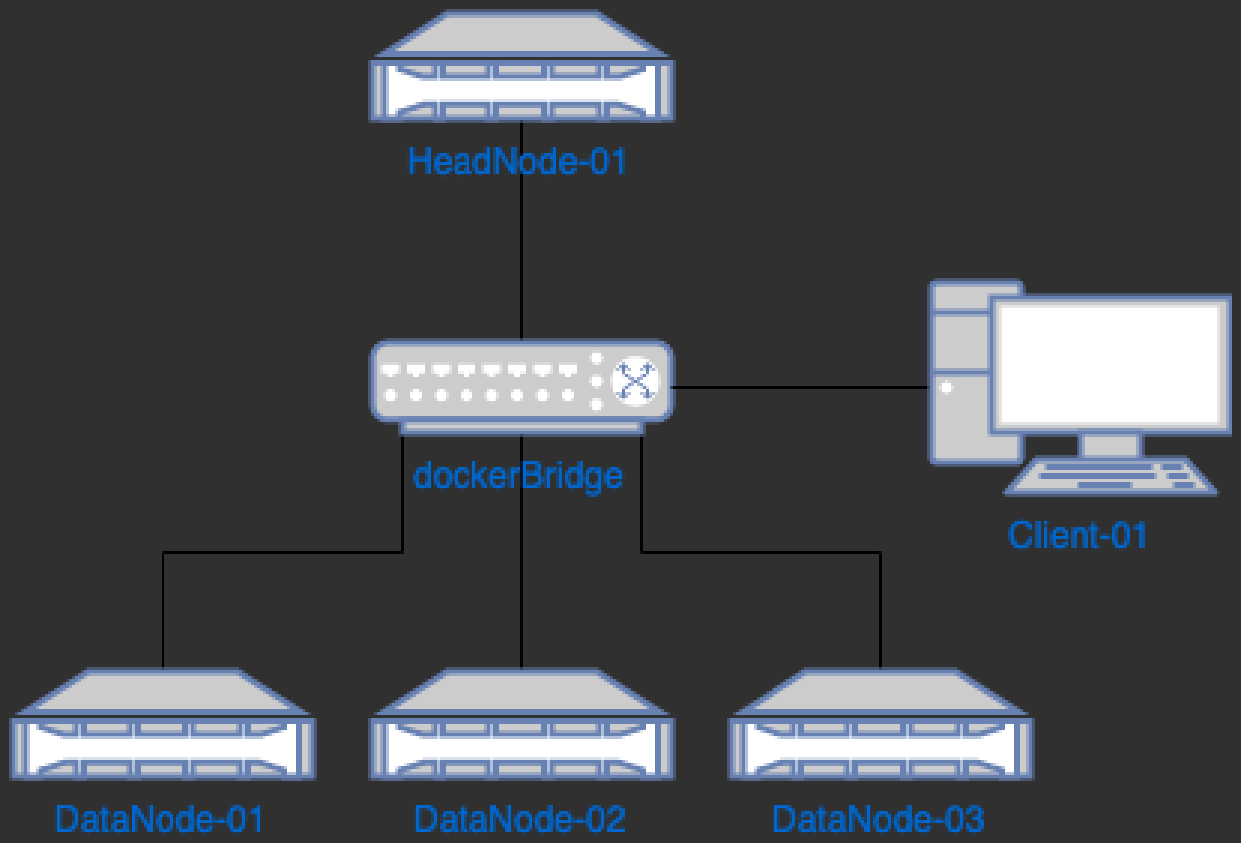
```
...

# manda el mensaje de solicitud al servidor
cliente.send(b"peticion")

# recibe la confirmacion
data = cliente.recv(bufferSize).decode("utf-8")
logging.info(data)

def randomMSG():
    Timer(3.0, randomMSG).start()
    numero = random.randint(0, 100)
    msg = "numero random: " + str(numero)
    cliente.send(msg.encode("utf-8"))

    data = cliente.recv(bufferSize).decode("utf-8")
    logging.info("Servidor recibio: " + data)

randomMSG()
```

Cliente - Características

- Se utilizó la librería `logging` para guardar los output como archivos.
- El objetivo del cliente es conectarse por TCP/5000 al servidor y enviar un número aleatorio entre $[0, 100]$.

Actividad 2

Topología


HeadNode-01
dockerBridge
Client-01
DataNode-01
DataNode-02
DataNode-03

headNode

```
class ServerThread(Thread):
...
    def run(self):
        while True:
            # se recibe de dataNode
            data = self.conn.recv(bufferSize).decode("utf-8")
            if data == "recibido":
                registro.info("recibido en dataNode " + threads[eleccionData].getIP())
                msg = str(threads[eleccionData].getIP()) + " , mensaje: " + data
                conn.send(msg.encode("utf-8"))
            else:
                registro.info("Problema: mensaje no fue guardado en dataNode")
...
class ClientThread(Thread):
...
    def run(self):
        while True:
            # recibe mensajes de cliente
            data = self.conexion.recv(bufferSize).decode("utf-8")
            # se elige dataNode random
            eleccionData = random.randint(0, len(threads))
            # se distribuye
            msg = "mensaje: " + data
            threads[eleccionData].getConn().send(msg.encode("utf-8"))
...
```

hearbeat

```
def heartbeat():
    Timer(5.0, heartbeat).start()

    for i in range(len(threads)):
        estado = os.system("ping -c 1 > /dev/null " + threads[i].getIP())

        if estado == 0:
          hearbeat.info(threads[i].getIP() + " disponible")
        else:
          hearbeat.info(threads[i].getIP() + " no disponible, eliminado")
          threads.remove(threads[i])

    if len(threads) == 0:
        registro.info("no hay dataNodes")
```

dataNode

```
class ReceiveThread(Thread):
...
    def run(self):
        while True:
            # mensaje a guardar
            data = dataNode.recv(bufferSize).decode("utf-8")
            # se registra en data.txt
            registroData.info(data)
            # avisa que fue recibido
            dataNode.send(b"recibido")

dataNode = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
dataNode.connect( (host, port) )
registroData.info('Conexion establecida con ' + str(dataNode.getsockname()) + ' y ' + str(dataNode.getpeername(
# handshake 3 pasos
dataNode.send(b"peticion")
data = dataNode.recv(bufferSize).decode("utf-8")

if data != "confirmacion":
    registroData.info("Servidor no conecta.")
    dataNode.close()
    exit(0)

registroData.info(data)
dataNode.send(b"dataNode")

# entrada para el usuario
receiveThread = ReceiveThread()
receiveThread.start()
```

Cliente - Código

```
...

# manda el mensaje de solicitud al servidor
cliente.send(b"peticion")

# recibe la confirmacion
data = cliente.recv(bufferSize).decode("utf-8")
logging.info(data)

def randomMSG():
    Timer(3.0, randomMSG).start()
    numero = random.randint(0, 100)
    msg = "numero random: " + str(numero)
    cliente.send(msg.encode("utf-8"))

    data = cliente.recv(bufferSize).decode("utf-8")
    logging.info("Servidor recibio: " + data)

randomMSG()
```

Características Generales

- Al igual que en la Actividad 1, se utilizó la librería `logging` para guardar los output como archivos, el cliente se conectó por TCP/5000 al servidor y enviando número aleatorio entre $[0, 100]$.
- Se utilizó el thread temporizador `Timer` para realizar los `hearbeat`.

Fin