



Departamento de Informática
Universidad Técnica Federico Santa María

Presentación Tarea 1

Sistemas Distribuidos

Cristian Navarrete
Benjamin Seider

2019-2




Departamento de Informática
Universidad Técnica Federico Santa María

Estructura de Archivos

✓ parte1

✓ client

 client.py

 Dockerfile

✓ server


 Dockerfile

 server.py

 docker-compose.yml


✓ parte2

✓ client

 client.py


 Dockerfile

✓ datanode1

 datanode1.py


 Dockerfile

✓ datanode2

 datanode2.py

 Dockerfile


✓ datanode3

 datanode3.py


 Dockerfile


✓ headnode

 Dockerfile

 headnode.py

9+

 cliente_opcional.py

 docker-compose.yml



Parte 1

```
1 version: "3"
2 networks:
3   sdtareal:
4     ipam:
5       driver: default
6       config:
7         - subnet: 172.16.1.0/24
8 services:
9   server:
10    build: ./server
11    volumes:
12      - ./server:/app/
13    networks:
14      sdtareal:
15        ipv4_address: "172.16.1.100"
16    ports:
17      - "5000:5000"
```

```
18 client:
19   build: ./client
20   volumes:
21     - ./client:/app/
22   networks:
23     sdtareal:
24       ipv4_address: "172.16.1.101"
25   depends_on:
26     - server
27
```

Docker Compose

```
1 FROM      python:3.7.4-alpine3.9
2 WORKDIR   /app
3 EXPOSE    5000
4 ENV       PYTHONUNBUFFERED 1
5 CMD       ["python", "./server.py"]
```

Dockerfile Servidor

```
1
2 import socket
3 from _thread import *
4 import threading
5
6
7 TCP_IP = '0.0.0.0'
8 TCP_PORT = 5000
9
10 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11 s.bind((TCP_IP, TCP_PORT))
12 s.listen(10)
13 write_lock = threading.Lock()
14
15
```

```
15
16 def handle_client(c, client_address):
17     while True:
18
19         # data received from client
20         data = c.recv(1024)
21         if not data:
22             c.close()
23             break
24         data = data.decode('utf-8').strip()
25         write_lock.acquire()
26         logs = open("log.txt", "a")
27         logs.write(":".join(map(str, client_address)) + " | "+data+"\n")
28         print(data)
29         logs.close()
30         write_lock.release()
31
32         respuesta = "Exito\n"
33         # Respuesta
34         c.send(respuesta.encode("utf-8"))
35
36     # connection closed
37
38
```

```
38
39 while True:
40     connection, client_address = s.accept()
41     print('connection from', client_address)
42     start_new_thread(handle_client, (connection, client_address,))
43
```

Servidor

```
1 FROM      python:3.7.4-alpine3.9
2 WORKDIR   /app
3 ENV       PYTHONUNBUFFERED 1
4 CMD       ["python", "./client.py"]
```

Dockerfile Cliente

```
1 import socket
2
3
4 TCP_IP = '172.16.1.100'
5 TCP_PORT = 5000
6 BUFFER_SIZE = 1024
7
8 log = open("respuestas.txt", "a")
9
10 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11 s.connect((TCP_IP, TCP_PORT))
12 s.send("Saludos\n".encode("utf-8"))
13 data = s.recv(BUFFER_SIZE)
14 data = data.decode('utf-8').strip()
15 log.write(data+"\n")
16 log.close()
17 s.close()
18
```

Cliente



Parte 2

```
1 version: "3"
2 networks:
3   sdtareal:
4     ipam:
5       driver: default
6       config:
7         - subnet: 172.16.1.0/24
8 services:
9   headnode:
10    build: ./headnode
11    volumes:
12      - ./headnode:/app/
13    networks:
14      sdtareal:
15        ipv4_address: "172.16.1.100"
16    ports:
17      - "5000:5000"
```

```
18   datanode1:
19     build: ./datanode1
20     volumes:
21       - ./datanode1:/app/
22     networks:
23       sdtareal:
24         ipv4_address: "172.16.1.101"
25     depends_on:
26       - headnode
```

```
45   client:
46     build: ./client
47     volumes:
48       - ./client:/app/
49     networks:
50       sdtareal:
51         ipv4_address: "172.16.1.104"
52     depends_on:
53       - headnode
```

Docker Compose

```
1 FROM      python:3.7.4-alpine3.9
2 WORKDIR   /app
3 EXPOSE    5000
4 ENV       PYTHONUNBUFFERED 1
5 CMD       ["python", "./headnode.py"]
```

Dockerfile HEADNODE

```
1 import socket
2 from _thread import *
3 import threading
4 from random import randrange
5 import struct
6 import time
7 from datetime import datetime
8
9
10 TCP_IP = '0.0.0.0'
11 TCP_PORT = 5000
12
13 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14 s.bind((TCP_IP, TCP_PORT))
15 s.listen(10)
16 send_lock = threading.Lock()
17
18 BUFFER_SIZE = 1024
19
20 ips_nodes = ['172.16.1.101', '172.16.1.102', '172.16.1.103']
21 nodes_active = [False, False, False]
22
23
24 def handle_client(c, client_address):
```

```
58
59 def operational():
60     global nodes_active
61     message = b'Operativo?'
```

```
100
101 start_new_thread(operational, ())
102
103 while True:
104     connection, client_address = s.accept()
105     print('connection from', client_address)
106     start_new_thread(handle_client, (connection, client_address,))
107
```

En General

```
1 def handle_client(c, client_address):
2     global nodes_active
3     while True:
4         # Leer data del cliente
5         data = c.recv(1024)
6         if not data:
7             print("Closing connection with", client_address)
8             c.close()
9             break
10        # Elegir nodo activo
11        node = randrange(3)
12        while not nodes_active[node]:
13            #print("Cheking if node ", node, "is active")
14            node = randrange(3)
15        print("Seleccionado nodo ", node)
16        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17
18        s.connect((ips_nodes[node], TCP_PORT))
19
20        s.send(data)
21
```

```
22
23        response = s.recv(BUFFER_SIZE)
24        response = response.decode('utf-8').strip()
25        if response == "SUCCESS":
26            logs = open("registro_server.txt", "a")
27            now = datetime.now()
28            dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
29            logs.write("[ "+dt_string+" ] Guardado en datanode " +
30                      (str(node+1))+ "\n")
31            logs.close()
32            respuesta = "Guardado en datadone " +(str(node+1))+ "\n"
33            # Respuesta
34            c.send(respuesta.encode("utf-8"))
```

función handle_client (Gestión del mensaje)

```
1 def operational():
2     global nodes_active
3     message = b'Operativo?'
4     multicast_group = ('224.3.29.71', 10000)
5
6     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
7
8     sock.settimeout(1)
9
10    ttl = struct.pack('b', 2)
11    sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)
12
13    try:
14        while True:
15            time.sleep(5)
16            for i in range(len(nodes_active)):
17                nodes_active[i] = False
18            sent = sock.sendto(message, multicast_group)
19            while True:
20
```

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

try:

data, server = sock.recvfrom(1024)

response = data.decode('utf-8').strip()

server = response.split("-D")

server = int(server[-1])-1

nodes_active[server] = True

now = datetime.now()

dt_string = now.strftime("%d/%m/%Y %H:%M:%S")

logs2 = open("heartbeat_server.txt", "a")

logs2.write "[" + dt_string + " " + response + "\n")

logs2.close()

except socket.timeout:

break

finally:

print('closing socket')

logs2.close()

sock.close()

41

Heartbeat Headnode

```
1 FROM      python:3.7.4-alpine3.9
2 WORKDIR   /app
3 ENV PYTHONUNBUFFERED 1
4 CMD       ["python", "./datanode1.py"]
```

Dockerfile DATANODES

Procesamiento “Archivo” datanode

```
1 def handle_client(c, client_address):
2     while True:
3
4         # data received from client
5         data = c.recv(1024)
6         if not data:
7             c.close()
8             break
9         data = data.decode('utf-8').strip()
10        write_lock.acquire()
11        logs = open("data.txt", "a")
12        logs.write(data+"\n")
13        print(data)
14        logs.close()
15        write_lock.release()
16
17        respuesta = "SUCCESS"
18        # Respuesta
19        c.send(respuesta.encode("utf-8"))
20
21    # connection closed
```

```
1 def operational():
2     multicast_group = '224.3.29.71'
3     server_address = ('', 10000)
4
5     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6
7     sock.bind(server_address)
8
9     group = socket.inet_aton(multicast_group)
10    mreq = struct.pack('4sL', group, socket.INADDR_ANY)
11    sock.setsockopt(
12        socket.IPPROTO_IP,
13        socket.IP_ADD_MEMBERSHIP,
14        mreq)
15
16    while True:
17        data, address = sock.recvfrom(1024)
18        sock.sendto(b'PONG-D1', address)
19
```

Response a heartbet datanode

```
1 FROM      python:3.7.4-alpine3.9
2 WORKDIR   /app
3 ENV PYTHONUNBUFFERED 1
4 CMD       ["python", "./client.py"]
```

Dockerfile Cliente

```
1 import socket
2
3
4 TCP_IP = '172.16.1.100'
5 TCP_PORT = 5000
6 BUFFER_SIZE = 1024
7
8 log = open("registro_cliente.txt", "a")
9
10 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11 s.connect((TCP_IP, TCP_PORT))
12 s.send("Mensaje\n".encode("utf-8"))
13 data = s.recv(BUFFER_SIZE)
14 data = data.decode('utf-8').strip()
15 log.write(data+"\n")
16 log.close()
17 s.close()
```

Código Cliente



Extra: cliente_opcional.py

```
1 import socket
2
3
4 TCP_IP = '127.0.0.1'
5 TCP_PORT = 5000
6 BUFFER_SIZE = 1024
7
8 log = open("registro_cliente.txt", "a")
9
10 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11 s.connect((TCP_IP, TCP_PORT))
12 mensaje = input("Ingrese mensaje o SALIR para salir: ")
13 while mensaje != "SALIR":
14     s.send((mensaje).encode("utf-8"))
15     data = s.recv(BUFFER_SIZE)
16     data = data.decode('utf-8').strip()
17     log.write(data+"\n")
18
19     mensaje = input("Ingrese mensaje o SALIR para salir: ")
20 log.close()
21 s.close()
22
```

cliente_opcional.py



Departamento de Informática
Universidad Técnica Federico Santa María

Gracias, ¿Consultas?
