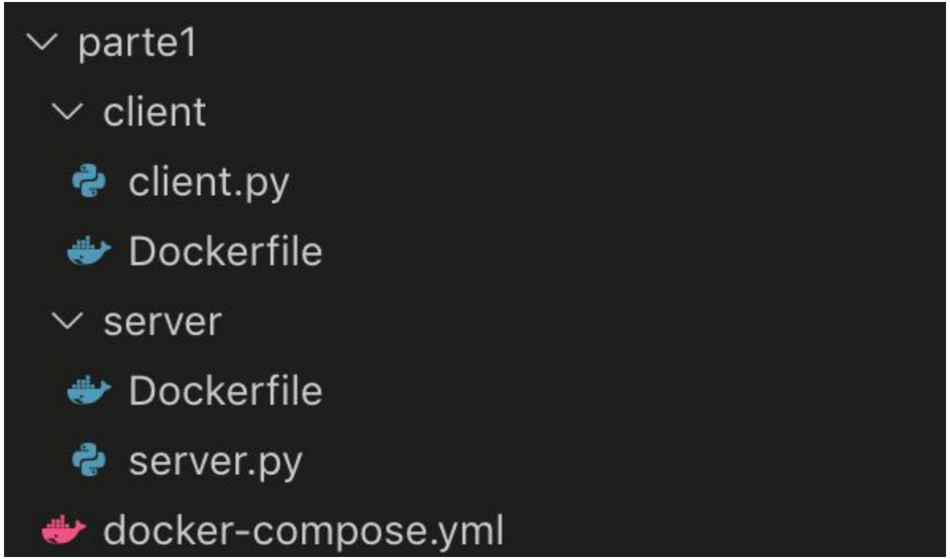
Departamento de Informática
Universidad Técnica Federico Santa María

Presentación Tarea 1 Sistemas Distribuidos

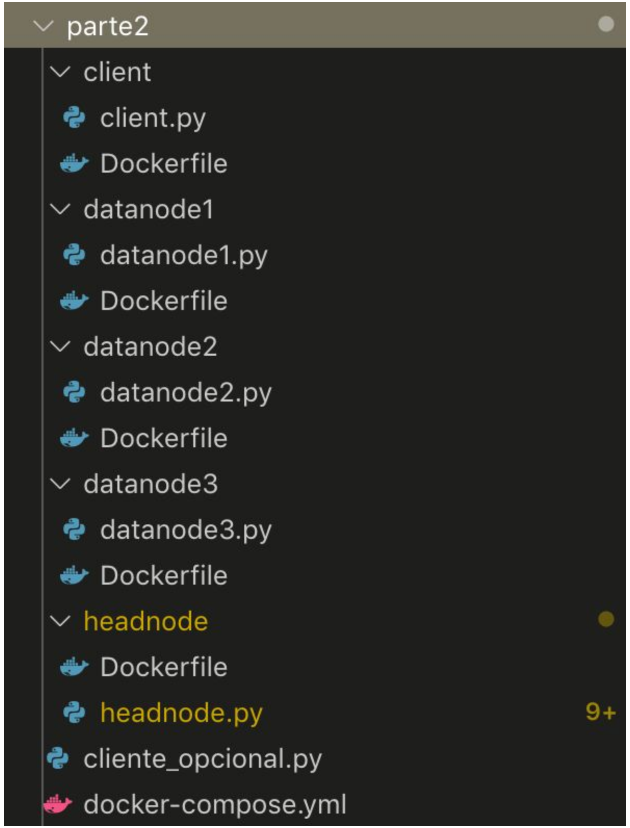
Cristian Navarrete
Benjamin Seider

2019-2

Estructura de Archivos



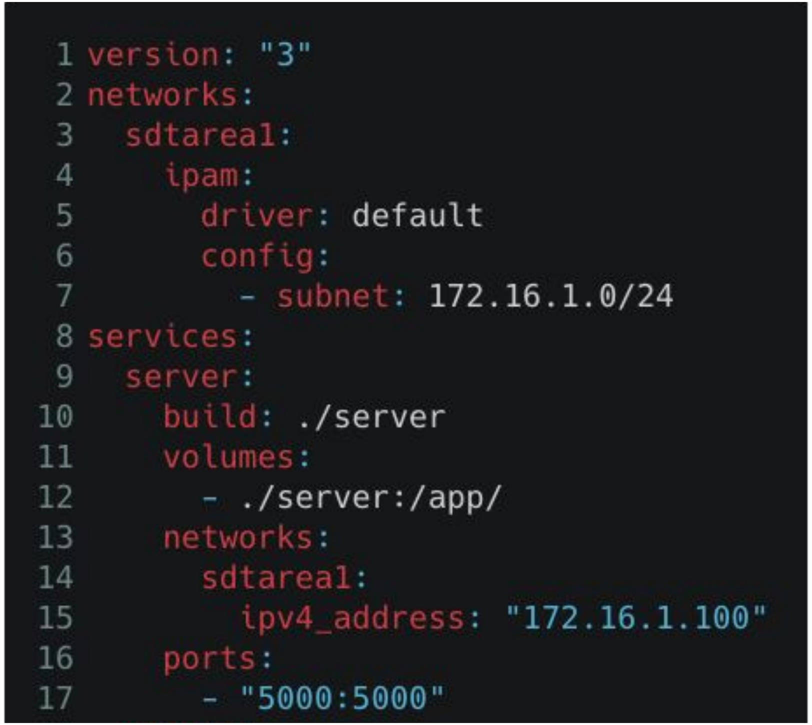
```
parte1
  client
    client.py
    Dockerfile
  server
    Dockerfile
    server.py
  docker-compose.yml
```



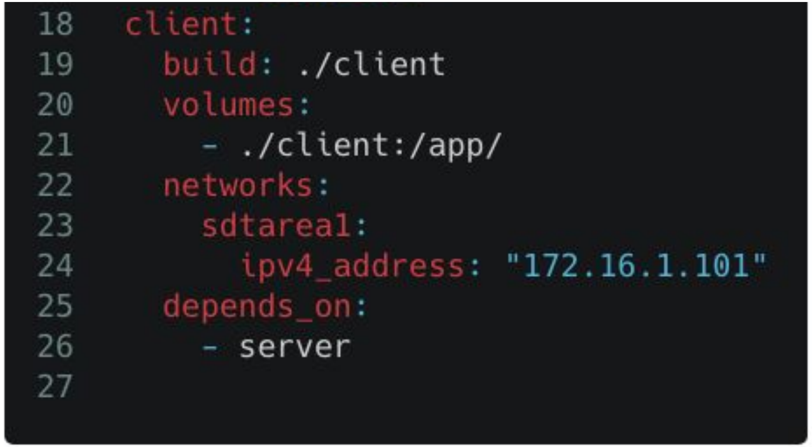
```
parte2
  client
    client.py
    Dockerfile
  datanode1
    datanode1.py
    Dockerfile
  datanode2
    datanode2.py
    Dockerfile
  datanode3
    datanode3.py
    Dockerfile
  headnode
    Dockerfile
    headnode.py                9+
  cliente_opcional.py
  docker-compose.yml
```

Parte 1

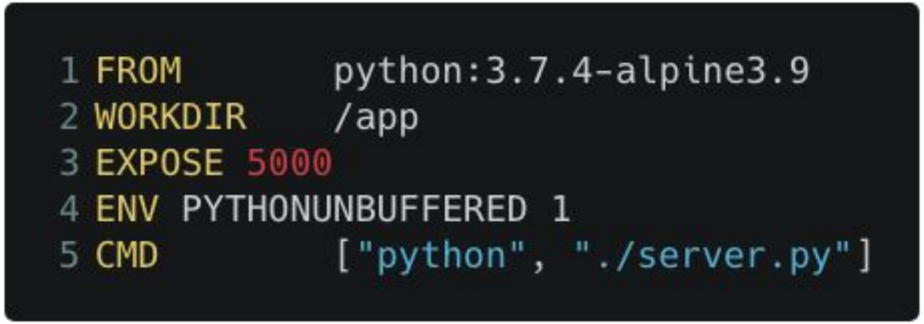
Docker Compose



```yaml
version: "3"
networks:
  sdtarea1:
    ipam:
      driver: default
      config:
        - subnet: 172.16.1.0/24
services:
  server:
    build: ./server
    volumes:
      - ./server:/app/
    networks:
      sdtarea1:
        ipv4_address: "172.16.1.100"
    ports:
      - "5000:5000"
```



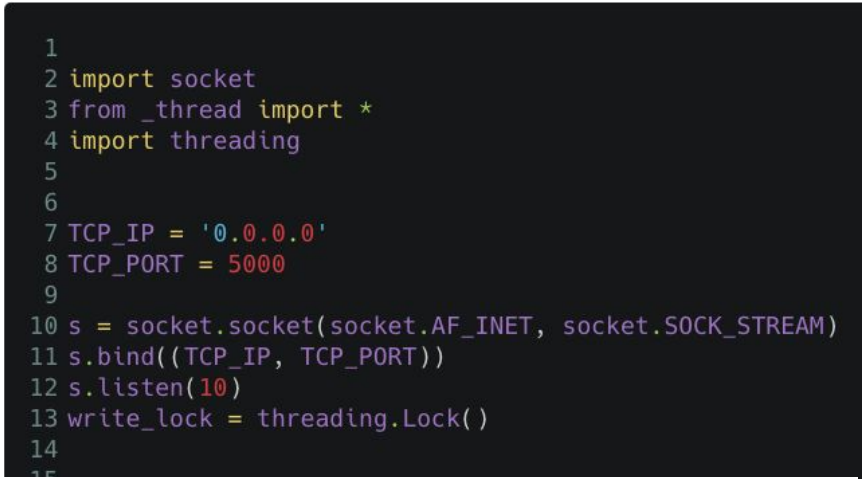
```yaml
  client:
    build: ./client
    volumes:
      - ./client:/app/
    networks:
      sdtarea1:
        ipv4_address: "172.16.1.101"
    depends_on:
      - server
```



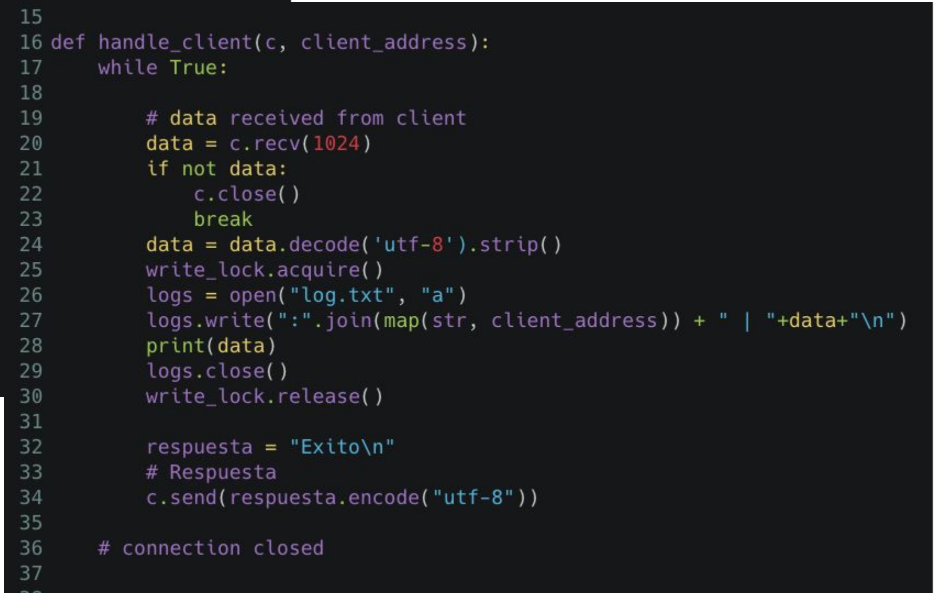
```
FROM       python:3.7.4-alpine3.9
WORKDIR    /app
EXPOSE 5000
ENV PYTHONUNBUFFERED 1
CMD        ["python", "./server.py"]
```

Dockerfile Servidor

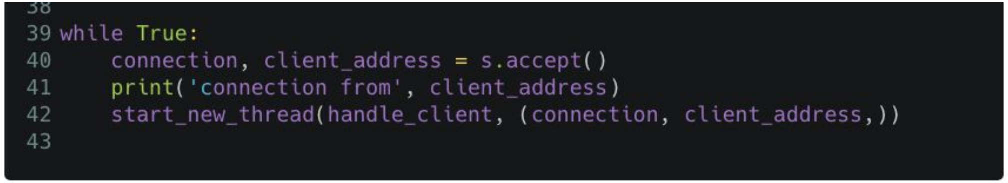
Servidor



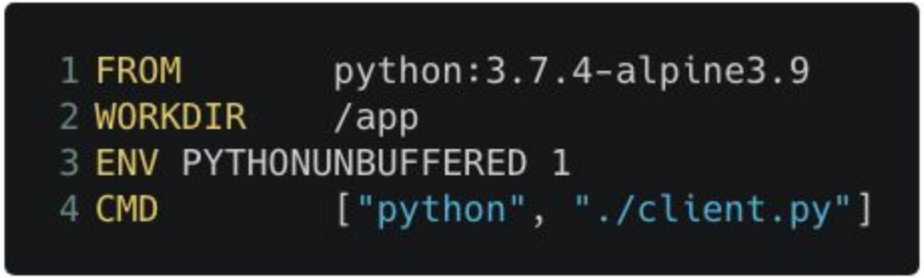
```python

import socket
from _thread import *
import threading


TCP_IP = '0.0.0.0'
TCP_PORT = 5000

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(10)
write_lock = threading.Lock()

```



```python

def handle_client(c, client_address):
    while True:

        # data received from client
        data = c.recv(1024)
        if not data:
            c.close()
            break
        data = data.decode('utf-8').strip()
        write_lock.acquire()
        logs = open("log.txt", "a")
        logs.write(":".join(map(str, client_address)) + " | "+data+"\n")
        print(data)
        logs.close()
        write_lock.release()

        respuesta = "Exito\n"
        # Respuesta
        c.send(respuesta.encode("utf-8"))

    # connection closed

```

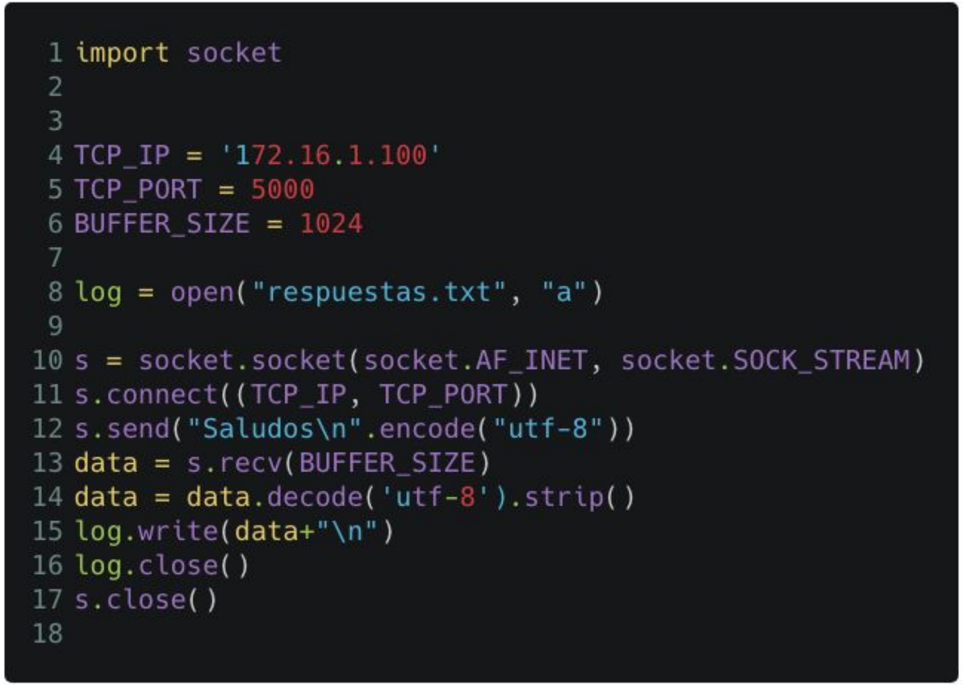


```python

while True:
    connection, client_address = s.accept()
    print('connection from', client_address)
    start_new_thread(handle_client, (connection, client_address,))

```



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

Dockerfile Cliente

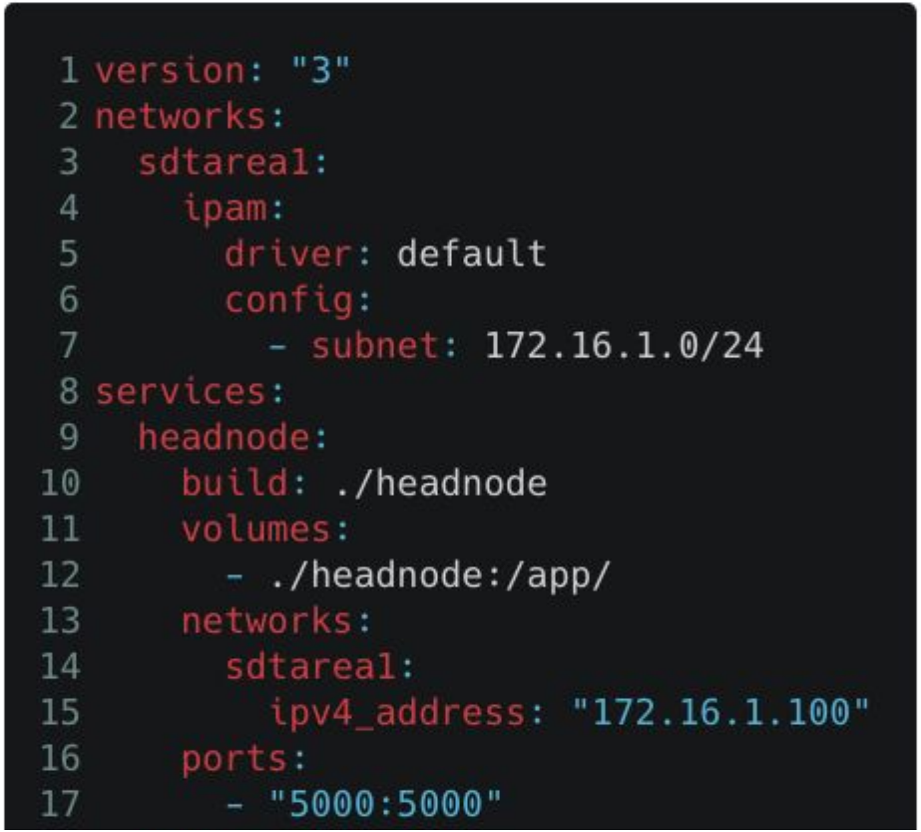


```python
import socket


TCP_IP = '172.16.1.100'
TCP_PORT = 5000
BUFFER_SIZE = 1024

log = open("respuestas.txt", "a")

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send("Saludos\n".encode("utf-8"))
data = s.recv(BUFFER_SIZE)
data = data.decode('utf-8').strip()
log.write(data+"\n")
log.close()
s.close()

```

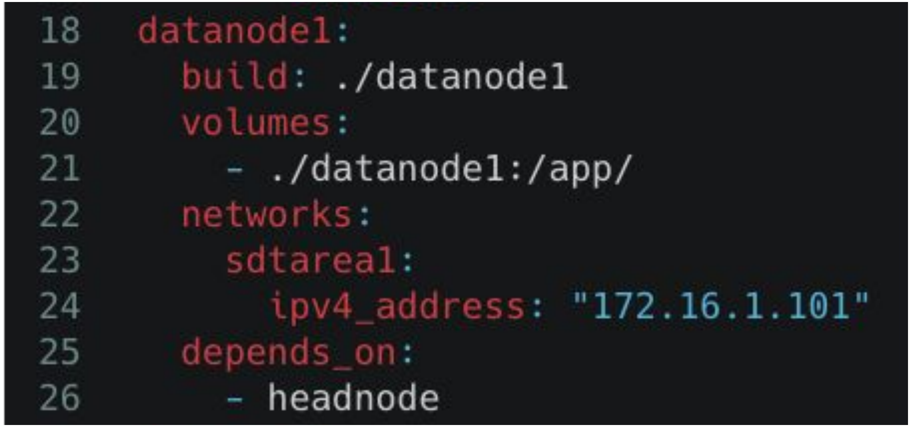
Cliente

Parte 2

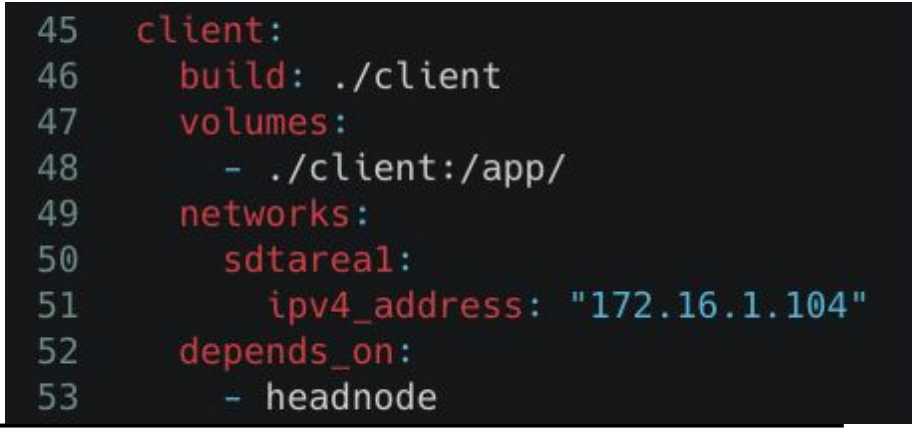
Docker Compose



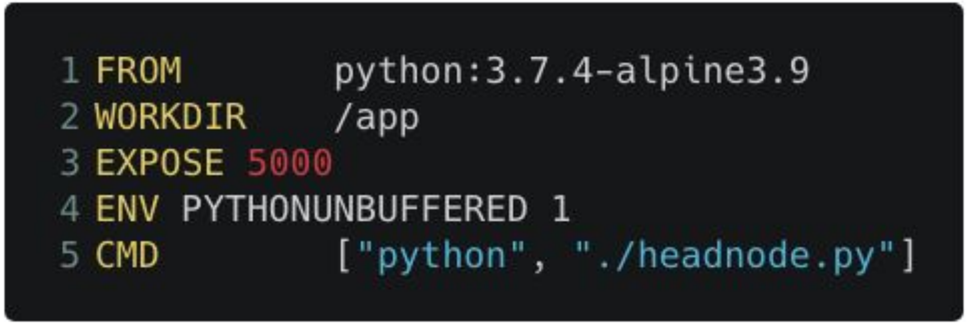
```yaml
version: "3"
networks:
  sdtarea1:
    ipam:
      driver: default
      config:
        - subnet: 172.16.1.0/24
services:
  headnode:
    build: ./headnode
    volumes:
      - ./headnode:/app/
    networks:
      sdtarea1:
        ipv4_address: "172.16.1.100"
    ports:
      - "5000:5000"
```



```yaml
  datanode1:
    build: ./datanode1
    volumes:
      - ./datanode1:/app/
    networks:
      sdtarea1:
        ipv4_address: "172.16.1.101"
    depends_on:
      - headnode
```



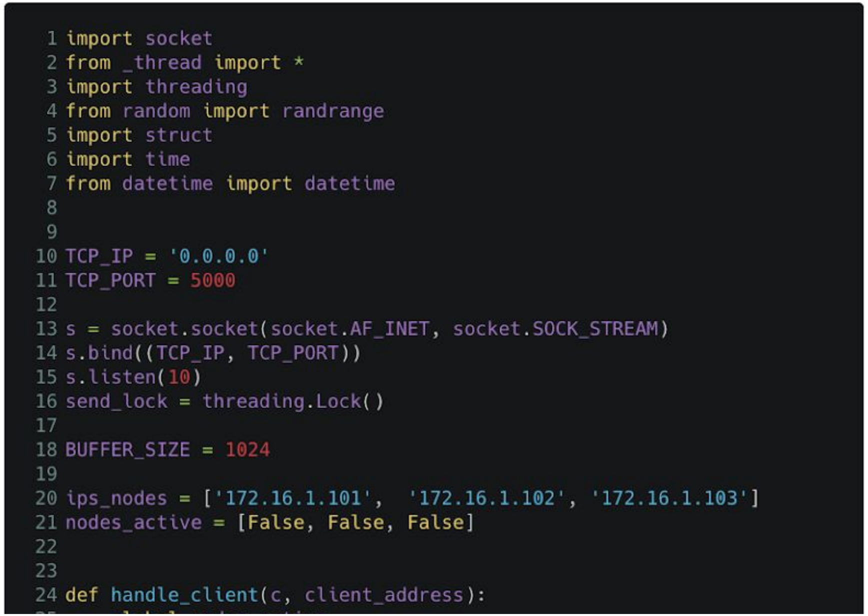
```yaml
  client:
    build: ./client
    volumes:
      - ./client:/app/
    networks:
      sdtarea1:
        ipv4_address: "172.16.1.104"
    depends_on:
      - headnode
```



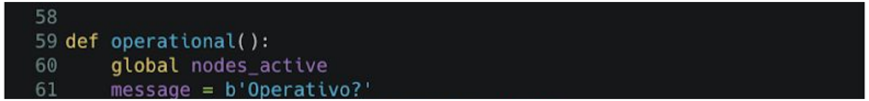
```
FROM       python:3.7.4-alpine3.9
WORKDIR    /app
EXPOSE 5000
ENV PYTHONUNBUFFERED 1
CMD        ["python", "./headnode.py"]
```

Dockerfile HEADNODE

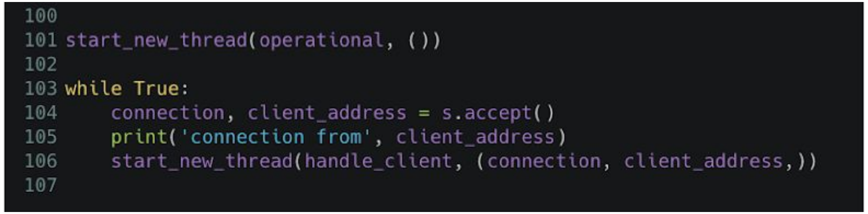
En General



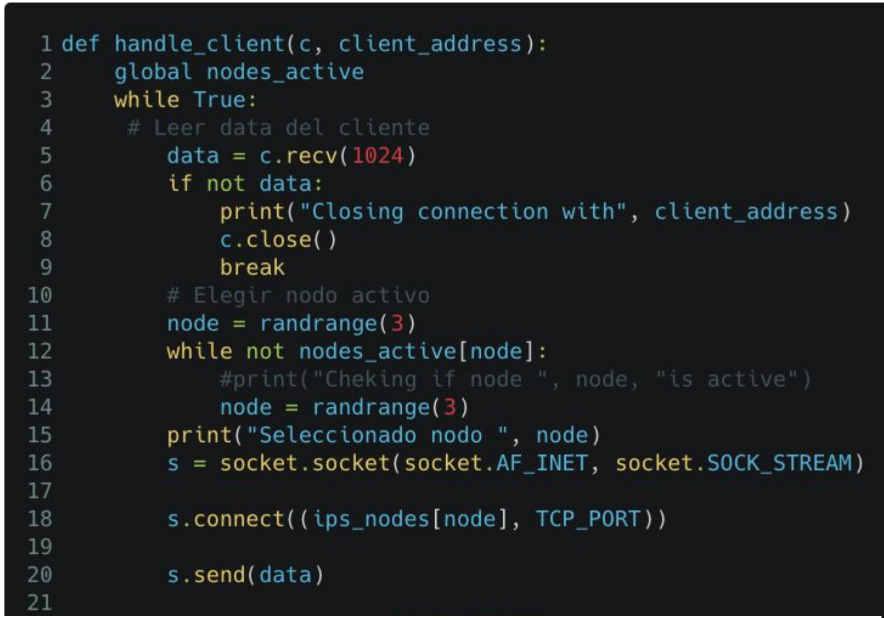
```python
import socket
from _thread import *
import threading
from random import randrange
import struct
import time
from datetime import datetime


TCP_IP = '0.0.0.0'
TCP_PORT = 5000

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(10)
send_lock = threading.Lock()

BUFFER_SIZE = 1024

ips_nodes = ['172.16.1.101',  '172.16.1.102', '172.16.1.103']
nodes_active = [False, False, False]


def handle_client(c, client_address):
```



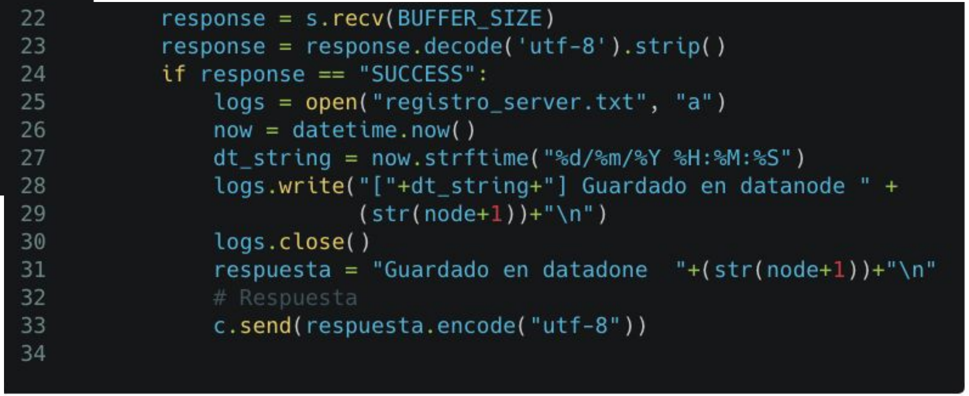
```python

def operational():
    global nodes_active
    message = b'Operativo?'
```



```python

start_new_thread(operational, ())

while True:
    connection, client_address = s.accept()
    print('connection from', client_address)
    start_new_thread(handle_client, (connection, client_address,))

```

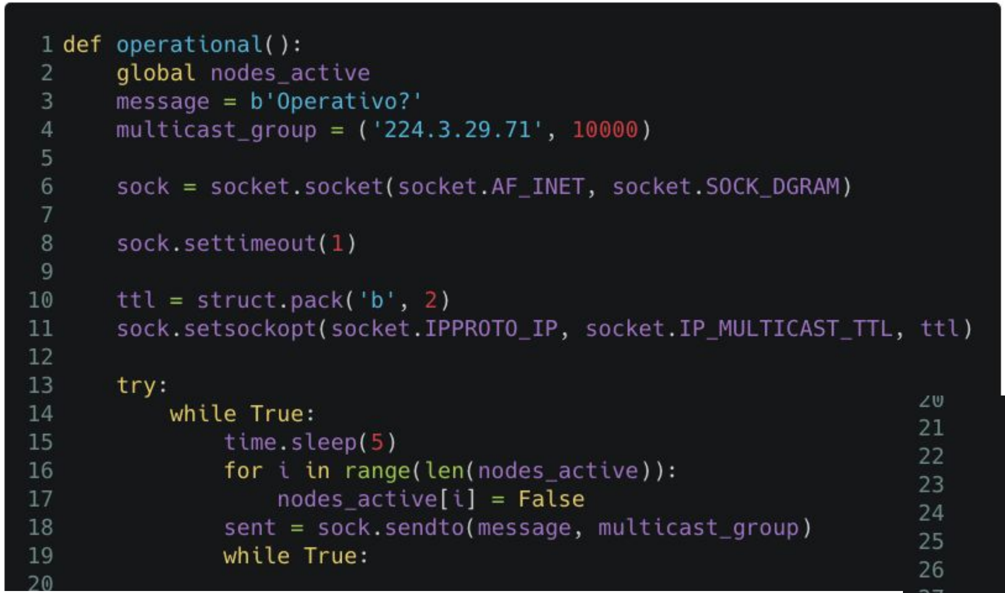


```python
def handle_client(c, client_address):
    global nodes_active
    while True:
     # Leer data del cliente
        data = c.recv(1024)
        if not data:
            print("Closing connection with", client_address)
            c.close()
            break
        # Elegir nodo activo
        node = randrange(3)
        while not nodes_active[node]:
            #print("Cheking if node ", node, "is active")
            node = randrange(3)
        print("Seleccionado nodo ", node)
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        s.connect((ips_nodes[node], TCP_PORT))

        s.send(data)

```

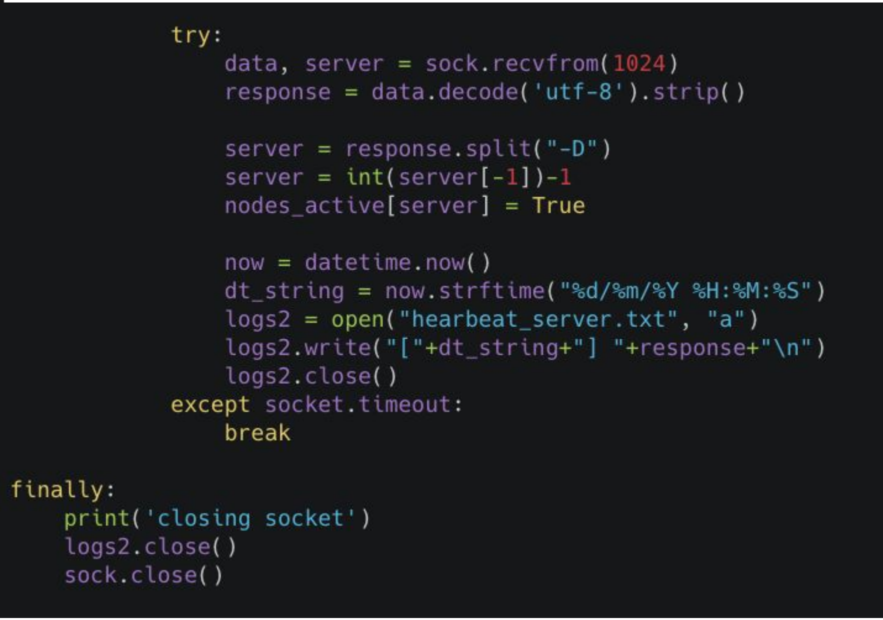


```python
        response = s.recv(BUFFER_SIZE)
        response = response.decode('utf-8').strip()
        if response == "SUCCESS":
            logs = open("registro_server.txt", "a")
            now = datetime.now()
            dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
            logs.write("["+dt_string+"]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

```

función handle_client (Gestión del mensaje)

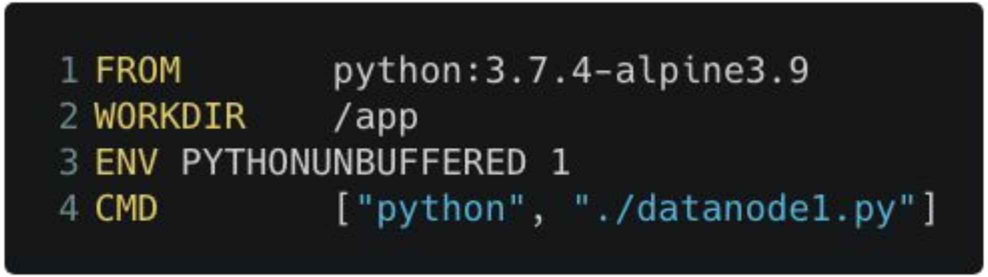


```python
def operational():
    global nodes_active
    message = b'Operativo?'
    multicast_group = ('224.3.29.71', 10000)

    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    sock.settimeout(1)

    ttl = struct.pack('b', 2)
    sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)

    try:
        while True:
            time.sleep(5)
            for i in range(len(nodes_active)):
                nodes_active[i] = False
            sent = sock.sendto(message, multicast_group)
            while True:
```



```python
                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("hearbeat_server.txt", "a")
                    logs2.write("["+dt_string+"] "+response+"\n")
                    logs2.close()
                except socket.timeout:
                    break

    finally:
        print('closing socket')
        logs2.close()
        sock.close()
```

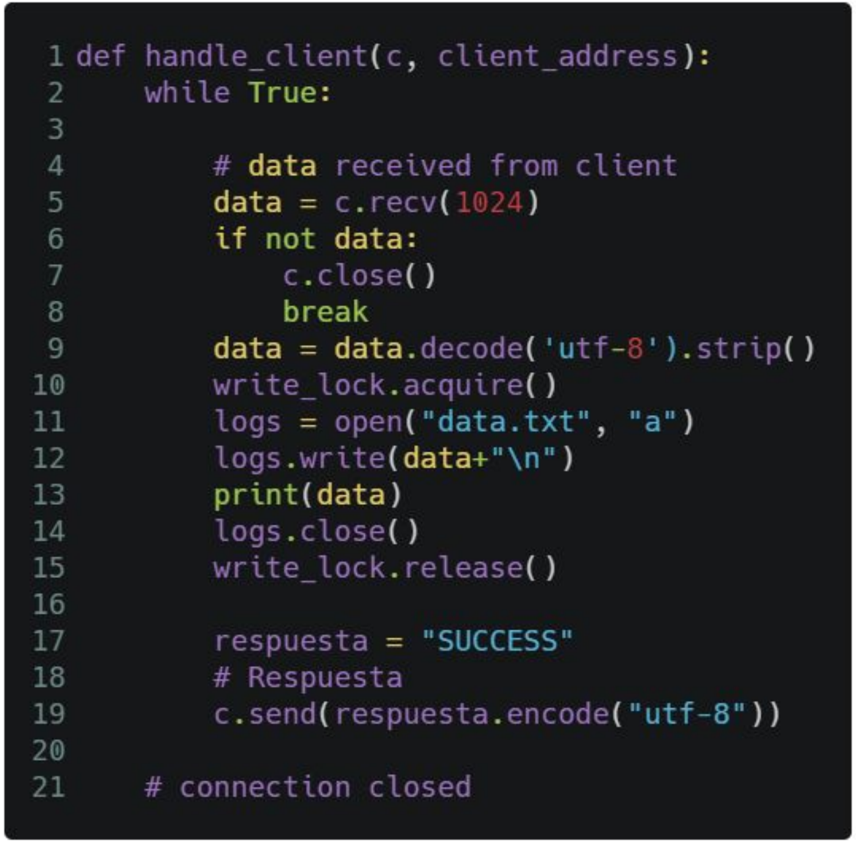
Heartbeat Headnode



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

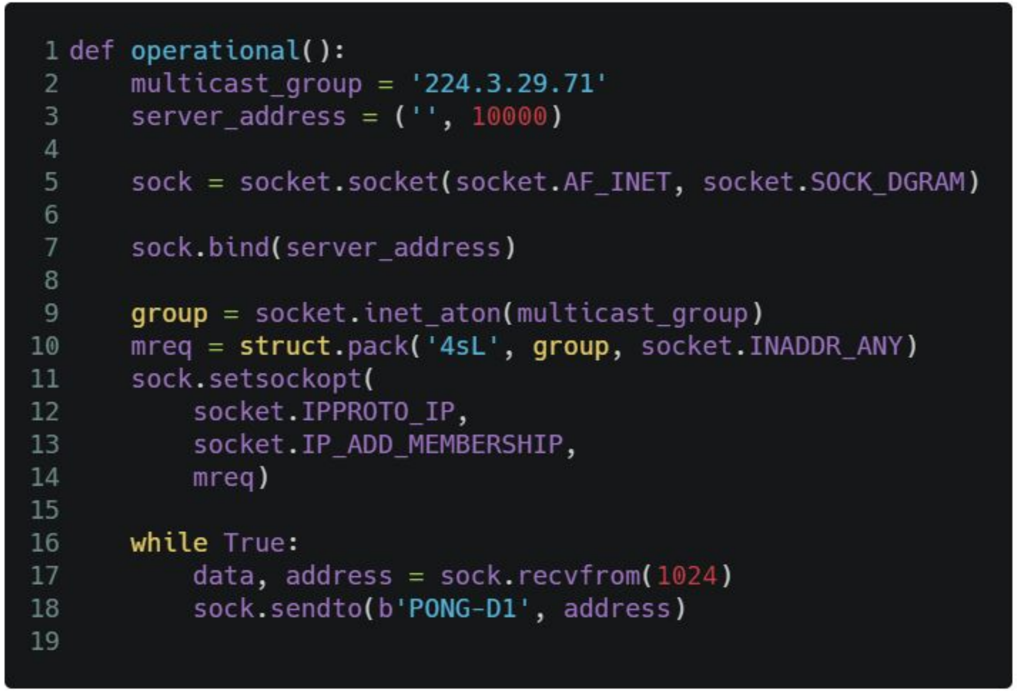
Dockerfile DATANODES

Procesamiento “Archivo” datanode

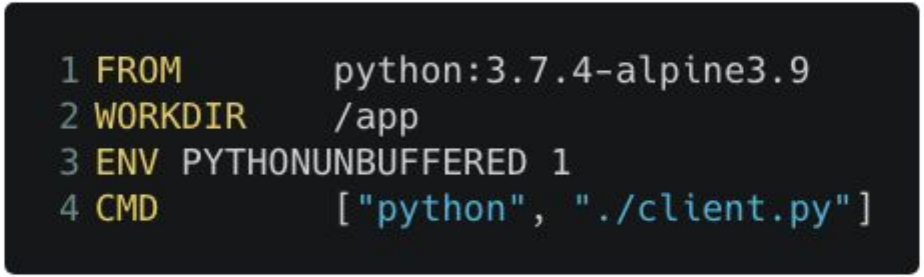


```python
def handle_client(c, client_address):
    while True:

        # data received from client
        data = c.recv(1024)
        if not data:
            c.close()
            break
        data = data.decode('utf-8').strip()
        write_lock.acquire()
        logs = open("data.txt", "a")
        logs.write(data+"\n")
        print(data)
        logs.close()
        write_lock.release()

        respuesta = "SUCCESS"
        # Respuesta
        c.send(respuesta.encode("utf-8"))

    # connection closed
```

Response a heartbet datanode

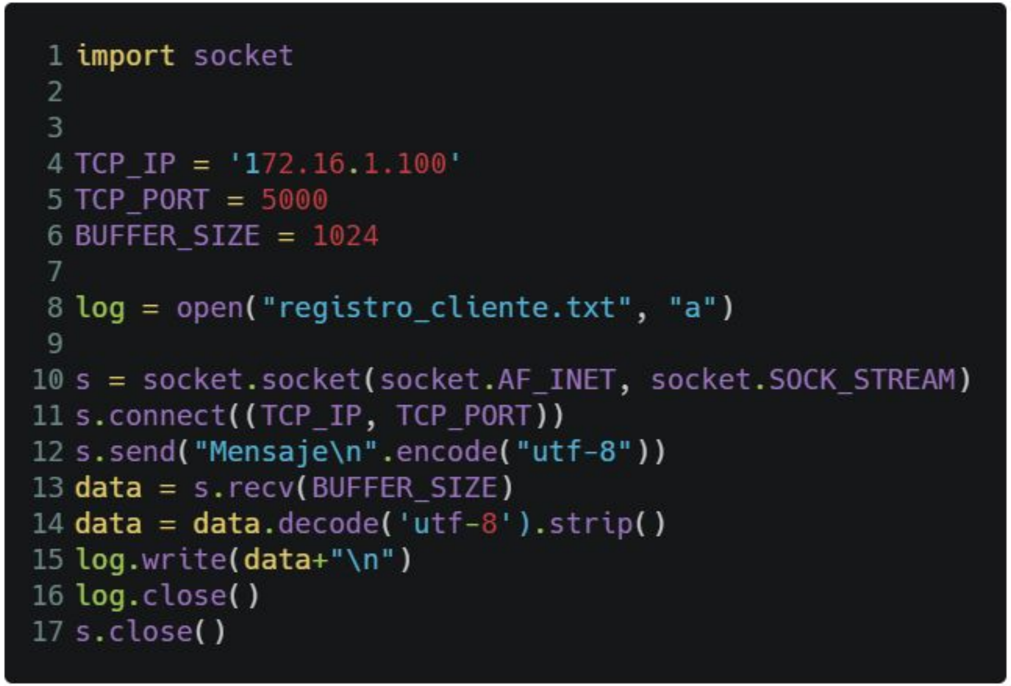


```python
def operational():
    multicast_group = '224.3.29.71'
    server_address = ('', 10000)

    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    sock.bind(server_address)

    group = socket.inet_aton(multicast_group)
    mreq = struct.pack('4sL', group, socket.INADDR_ANY)
    sock.setsockopt(
        socket.IPPROTO_IP,
        socket.IP_ADD_MEMBERSHIP,
        mreq)

    while True:
        data, address = sock.recvfrom(1024)
        sock.sendto(b'PONG-D1', address)

```



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

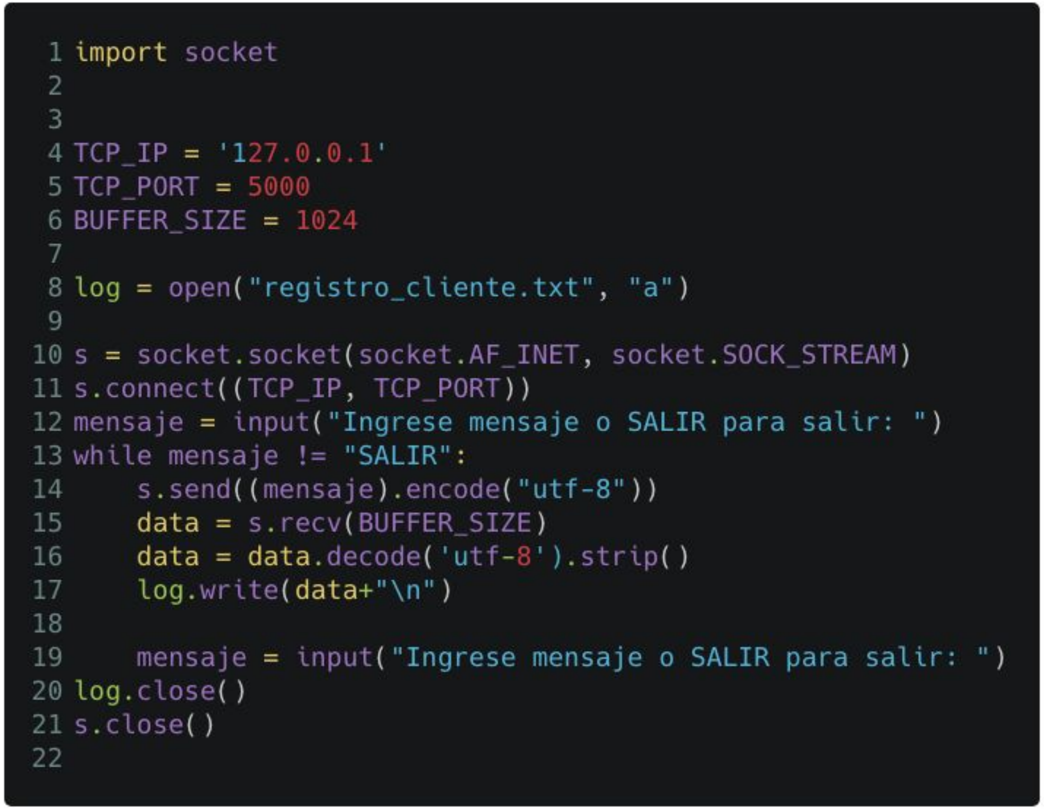
Dockerfile Cliente



```python
import socket


TCP_IP = '172.16.1.100'
TCP_PORT = 5000
BUFFER_SIZE = 1024

log = open("registro_cliente.txt", "a")

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send("Mensaje\n".encode("utf-8"))
data = s.recv(BUFFER_SIZE)
data = data.decode('utf-8').strip()
log.write(data+"\n")
log.close()
s.close()
```

Código Cliente

Extra: cliente_opcional.py



```python
import socket


TCP_IP = '127.0.0.1'
TCP_PORT = 5000
BUFFER_SIZE = 1024

log = open("registro_cliente.txt", "a")

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
mensaje = input("Ingrese mensaje o SALIR para salir: ")
while mensaje != "SALIR":
    s.send((mensaje).encode("utf-8"))
    data = s.recv(BUFFER_SIZE)
    data = data.decode('utf-8').strip()
    log.write(data+"\n")

    mensaje = input("Ingrese mensaje o SALIR para salir: ")
log.close()
s.close()

```

cliente_opcional.py

Gracias, ¿Consultas?