

# Tarea 1 : Containers con Docker

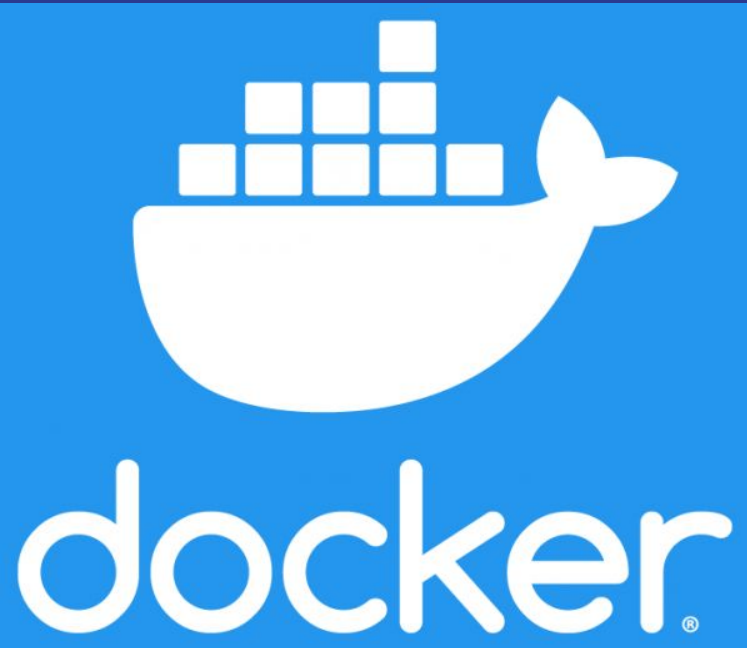
15/10/2019

Integrantes:

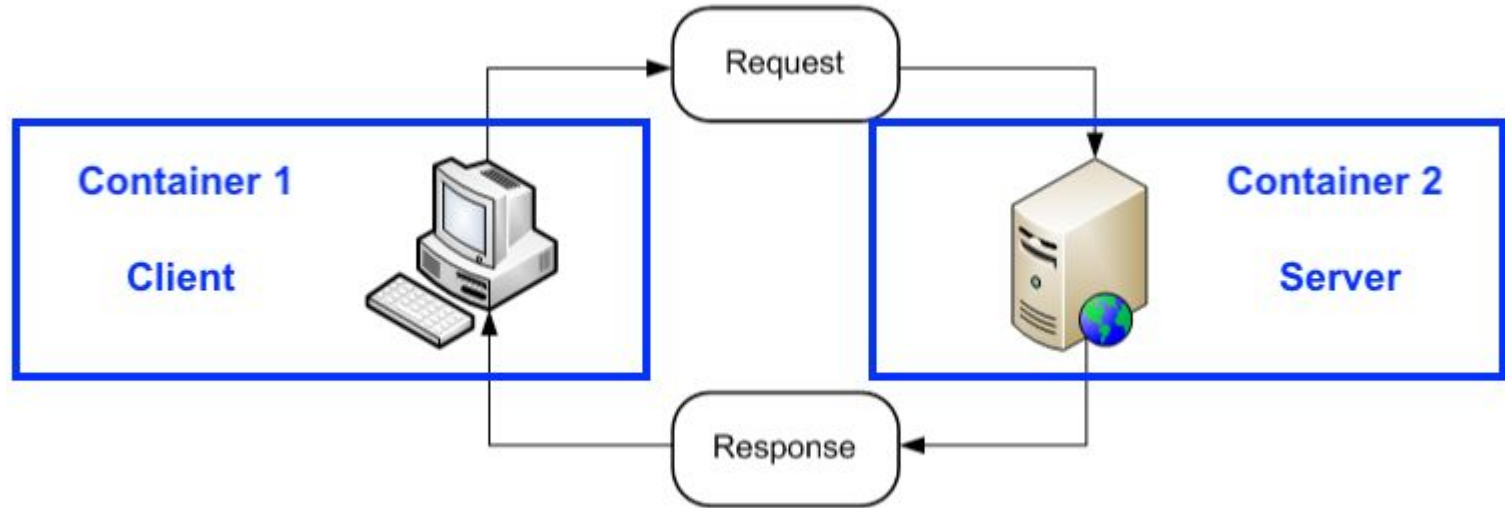
Francisco Vásquez

Gabriel Arjona

# Introducción



# Actividad 1



# Server

Abre puerto 5000, queda esperando

Acepta la conexión

Server espera hasta que llegue  
un mensaje del cliente

```
serv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serv.bind(('0.0.0.0', 5000))
serv.listen()

while True:

    conn, addr = serv.accept()
    from_client = ''

    while True:

        data = conn.recv(4096)
```

# Client

Se conecta al puerto 5000

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(('0.0.0.0', 5000))
```

Se envía un mensaje al server y se espera a que llegue una respuesta

```
client.send(bytes('¡Hola!', 'utf-8'))
from_server = client.recv(4096)
print(from_server.decode("utf-8") + "\n")
f.write(from_server.decode("utf-8") + "\n")
```



# docker-compose.yml

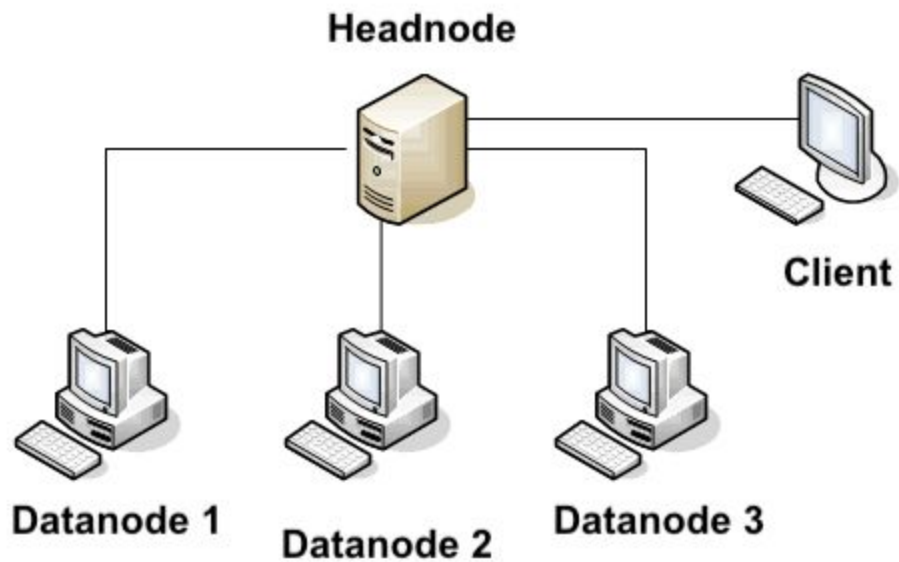
```
version: "3"
services:
  server:
    build: server/
    image: centos
    command: python ./server/server.py
    ports:
      - 5000:5000
    volumes:
      - ./:/server/
  client:
    build: client/
    image: ubuntu
    command: python ./client/client.py
    network_mode: host
    depends_on:
      - server
    volumes:
      - ./:/client/
    stdin_open: true
    tty: true
```

# Conclusiones de la actividad 1

- Se logró levantar la arquitectura cliente-servidor como containers, estableciéndose la comunicación a través de sockets.
- Mediante docker-compose se logra establecer el orden en que se levantan los servicios, tal que al levantar el servicio Client antes lo haga el servicio Server.
- Se dispone una consola interactiva para enviar mensajes al servidor, definiendo esta característica en el docker-compose.



# Actividad 2





# Headnode

Se conecta por cada  
puerto distinto al  
cliente y datanodes

```
15 serv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16 serv.bind(('0.0.0.0', 5000))
17 serv.listen()
18
19 datanode1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
20 datanode1.bind(('0.0.0.0', 6000))
21 datanode1.listen()
22
23 datanode2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
24 datanode2.bind(('0.0.0.0', 7000))
25 datanode2.listen()
26
27 datanode3 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
28 datanode3.bind(('0.0.0.0', 8000))
29 datanode3.listen()
```

# Headnode

```
37 class Datanodes(threading.Thread):
38     def run(self):
39         while True:
40             time.sleep(5) #Se ejecuta cada 5 segundos
41
42             message = 'ping'
43             f = open("heartbeat_server.txt","a")
44
45             conn1.send(bytes(message, 'utf-8'))
46             f.write(time.strftime("%X") + ": se envio ping a datanode1\n")
47             print("se envio ping a datanode1")
48
49             from_server = conn1.recv(4096)
50             f.write(time.strftime("%X") + ": se recibio respuesta de datanode1\n")
51             print("se recibio respuesta de datanode1")
52
53             #Y se repite por cada datanode
```

# Headnode

```
77 while True:
78
79     thread = Datanodes()
80     thread.start()
```

#Se inicia el thread Datanodes

```
88 while True:
```

```
96     datanode_elegido = random.randint(1,3)
97     if datanode_elegido == 1:
98         print("se envia los datos a datanode 1")
99         conn1.send(bytes(from_client, 'utf-8'))
100        from_server = conn1.recv(4096)
101        ip_elegido = IP_datanode1
102    elif datanode_elegido == 2:
```

#Se envía la data  
al datanode  
elegido

# Client

```
4 client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5 client.connect(('0.0.0.0', 5000))
```

#Se inicia la conexión al Headnode

```
16 while(True):
```

```
31 client.send(bytes(msg, 'utf-8'))
32 from_server = client.recv(4096)
33 f = open("client/registro_cliente.txt", "a")
34 f.write(str(contador) + "\t\t" + from_server.decode("utf-8") + "\n")
35 f.close()
```

#Se envía y se recibe  
confirmación del Headnode



# Datanodes

```
8 client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9 client.connect(('0.0.0.0', 6000))
```

#Se inicia la conexión al Headnode

```
15 while(True):
16
17     from_server = client.recv(4096)
18
19     if (from_server.decode("utf-8")== "ping"):
20         msg = "pong"
21         client.send(bytes(msg, 'utf-8'))
22         print("se ha enviado respuesta a headnode")
23     else:
24         f = open("datanode1/data.txt", "a")
25         f.write(from_server.decode("utf-8")+"\n")
26         f.close()
27         print("se ha guardado en data")
28         msg = "ok"
29         client.send(bytes(msg, 'utf-8'))
```

#Si se envía un ping responderá con un pong al headnode

#En cualquier otro caso se escribirá el data en el archivo data.txt

# Conclusiones de la actividad 2

- Se logró mediante docker la conexión entre containers con la arquitectura deseada.
- Se logró que el cliente al mandar un data, el Headnode decida aleatoriamente a qué datanode enviarlo, guardarlo y que quede registro de aquello.
- Queda a investigación cómo mejorar la conexión mediante docker cuando la conexión no es fiable y presenta una latencia mayor.

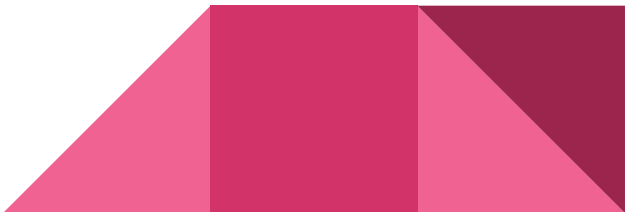


# Preguntas

# Dockerfiles Actividad 1

```
FROM python:latest  
ADD server.py /server/  
WORKDIR /server/
```

```
FROM python:latest  
ADD client.py /client/  
WORKDIR /client/
```





# log.txt y respuestas.txt Actividad 1

10/02/19 19:47:26

IP	Mensaje
172.22.0.1	¡Hola!

10/02/19 19:47:27

Conectado a 127.0.0.1

Respuestas

Se ha recibido su petición '¡Hola!'

# stdin\_open y tty actividad 1

```
Creating network "actividad1_default" with the default driver
Creating actividad1_server_1 ... done
Creating actividad1_client_1 ... done
Attaching to actividad1_server_1, actividad1_client_1
client_1 |
client_1 | Conectado a 127.0.0.1
client_1 |
client_1 | Se ha recibido su peticion '!Hola!'
client_1 |
client_1 | Ingrese su solicitud:
```

# Dockerfiles actividad 2

```
1 FROM python:latest
2 ADD client.py /client/
3 WORKDIR /client/
4
```

```
1 FROM python:latest
2 ADD headnode.py /headnode/
3 WORKDIR /headnode/
```



## registro\_cliente y registro\_server Actividad 2

1	id_data	datanode guardado
2	1	datanode3
3	2	datanode2
4	3	datanode2
5	4	datanode1
6	5	datanode1
7	6	datanode3
8	7	datanode2
9	8	datanode3
10	9	datanode3
11	10	datanode2
12	11	datanode3
13	12	datanode2
14	13	datanode3
15	14	datanode2
16	15	datanode2
17	16	datanode1
18	17	datanode1
19	18	datanode1
20	19	datanode2
21	20	datanode1

1	id	ip_cliente	datanode
2	0	192.168.240.1	3
3	1	192.168.240.1	2
4	2	192.168.240.1	3
5	3	192.168.240.1	3
6	4	192.168.240.1	2
7	5	192.168.240.1	2
8	6	192.168.240.1	1
9	7	192.168.240.1	2
10	8	192.168.240.1	3
11	9	192.168.240.1	1
12	10	192.168.240.1	3
13	11	192.168.240.1	2
14	12	192.168.240.1	1
15	13	192.168.240.1	1
16	14	192.168.240.1	2
17	15	192.168.240.1	3
18	16	192.168.240.1	1
19	17	192.168.240.1	1
20	18	192.168.240.1	2
21	19	192.168.240.1	2

# heartbeat\_server.txt

```
1 10/02/19      01:54:37
2
3 IP           Mensaje
4 01:54:43: se envio ping a datanode1
5 01:54:43: se recibio respuesta de datanode1
6 01:54:43: se envio ping a datanode2
7 01:54:43: se recibio respuesta de datanode2
8 01:54:43: se envio ping a datanode3
9 01:54:43: se recibio respuesta de datanode3
10 01:54:48: se envio ping a datanode1
11 01:54:48: se recibio respuesta de datanode1
12 01:54:48: se envio ping a datanode2
13 01:54:48: se recibio respuesta de datanode2
14 01:54:48: se envio ping a datanode3
15 01:54:48: se recibio respuesta de datanode3
16
```

# Docker-compose actividad 2

```
1  version: "3"
2  services:
3    headnode:
4      build: headnode/
5      image: centos
6      command: python ./headnode/headnode.py
7      environment:
8        - UDPPORT=10000
9      ports:
10         - 5000:5000
11         - 6000:6000
12         - 7000:7000
13         - 8000:8000
14         - 10000:10000/udp
15      volumes:
16        - ./:/headnode/
17      stdin_open: true
18      tty: true
19    datanode1:
20      build: datanode1/
21      command: python ./datanode1/datanode1.py
22      network_mode: host
23      image: ubuntu
24      depends_on:
25        - headnode
26      volumes:
27        - ./:/datanode1/
28      stdin_open: true
29      tty: true
30    datanode2:
```

```
31      build: datanode2/
32      command: python ./datanode2/datanode2.py
33      network_mode: host
34      image: debian
35      depends_on:
36        - headnode
37      volumes:
38        - ./:/datanode2/
39      stdin_open: true
40      tty: true
41    datanode3:
42      build: datanode3/
43      command: python ./datanode3/datanode3.py
44      network_mode: host
45      depends_on:
46        - headnode
47      volumes:
48        - ./:/datanode3/
49      stdin_open: true
50      tty: true
51    client:
52      build: client/
53      command: python ./client/client.py
54      network_mode: host
55      depends_on:
56        - headnode
57      volumes:
58        - ./:/client/
59      stdin_open: true
60      tty: true
```