Python: creare un server TCP usando le classi

Di seguito vedremo come creare un server TCP di base in Python usando una classe.

Il nostro server dovrà avere una porta ed un indirizzo IP su cui restare in ascolto. Ciascuna connessione da parte di un client verrà gestita in un thread separato.

Cominciamo a definire la nostra classe creando il costruttore.

```
import socket
import threading


class TCPServer:
    def __init__(self, ip='0.0.0.0', port=9999):
        self.ip = ip
        self.port = port
        self.server = None
        self.run()
```

L'attributo `server` verrà inizializzato all'interno del metodo `run()` in cui creeremo un'istanza della classe `socket`.

```
import socket
import threading


class TCPServer:
    def __init__(self, ip='0.0.0.0', port=9999):
        self.ip = ip
        self.port = port
        self.server = None
        self.run()

    def run(self):
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.bind((self.ip, self.port))
        self.server.listen()
        print(f'[*] Listening on {self.ip}:{self.port}')

        while True:
            client, address = self.server.accept()
            print(f'[*] Accepted connection from {address[0]}:{address[1]}')
            client_handler = threading.Thread(target=self.handle_client,
```

```
args=(client,))
            client_handler.start()
```

Il server viene inizializzato e posto in ascolto sull'IP e la porta specificati nel costruttore. Quindi per mantenere il server in esecuzione, creiamo un loop `while` infinito al cui interno accettiamo i dati inviati dal client. In questo loop creiamo un nuovo thread per ogni nuovo client che si connette. Il metodo `handle_client()` sarà un metodo statico definito come segue:

```
import socket
import threading


class TCPServer:
    def __init__(self, ip='0.0.0.0', port=9999):
        self.ip = ip
        self.port = port
        self.server = None
        self.run()

    def run(self):
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.bind((self.ip, self.port))
        self.server.listen()
        print(f'[*] Listening on {self.ip}:{self.port}')

        while True:
            client, address = self.server.accept()
            print(f'[*] Accepted connection from {address[0]}:{address[1]}')
            client_handler = threading.Thread(target=self.handle_client,
args=(client,))
            client_handler.start()

    @staticmethod
    def handle_client(client_socket):
        with client_socket as sock:
            request = sock.recv(1024)
            print(f'[*] Received: {request.decode("utf-8")}')
            sock.send(b'ACK')
```

È da notare che i dati in transito dal client al server e viceversa sono sotto forma di buffer di byte, quindi per visualizzarli nel terminale è necessario decodificarli in formato stringa.

Possiamo testare la nostra classe creando un file Python specifico:

```
from TCPServer import TCPServer


def main():
```

```
    tcp_server = TCPServer()


if __name__ == '__main__':
    main()
```

Eseguendo questo file otterremo nel Terminale:

```
[*] Listening on 0.0.0.0:9999
```

A livello client possiamo aprire una nuova finestra del Terminale e digitare il seguente comando:

```
nc 0.0.0.0 9999
```

Nella shell del server potremmo visualizzare ad esempio:

```
[*] Accepted connection from 127.0.0.1:49663
```

Ora digitiamo una parola (ad esempio `test`) nella shell del client e premiamo Invio. Nella shell del server avremo ad esempio:

```
[*] Received: test
```

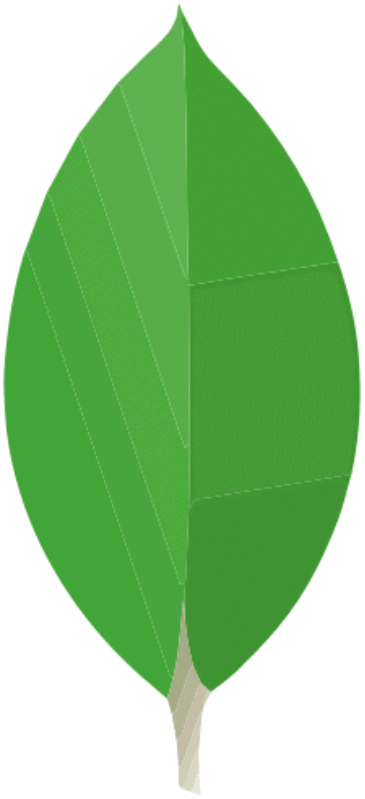
Come si può notare, si tratta di un'interazione elementare che tuttavia ci permette di comprendere le dinamiche del rapporto tra client e server TCP.

Applicazioni Correlate

-

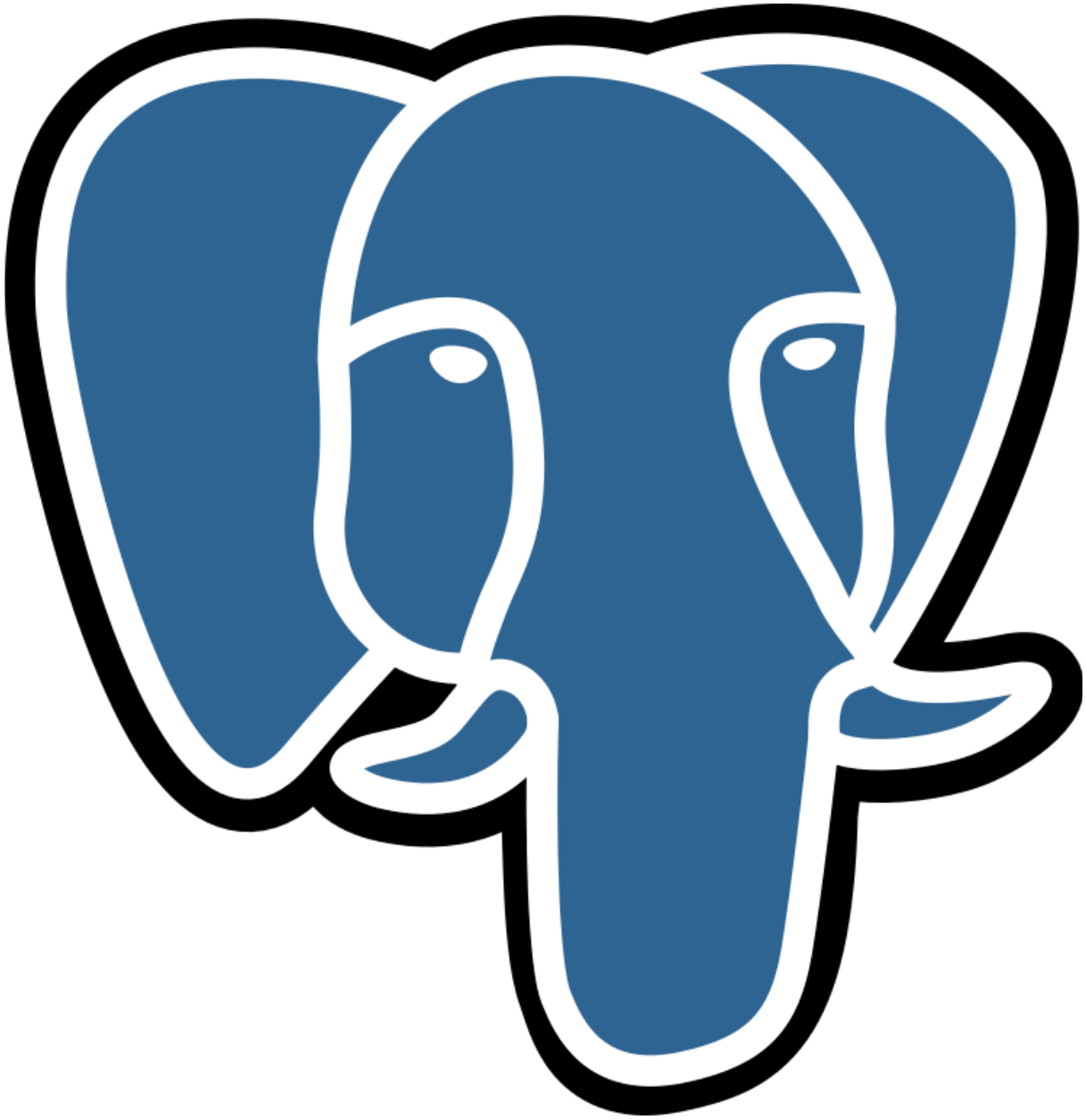
Python Placeholder Image

Applicazione sviluppata in Python con Flask per la creazione di immagini segnaposto.
DockerDocker ComposePythonFlaskJavaScript



Python MongoDB App

Applicazione basata su MongoDB e sviluppata in Python e Flask.
DockerDocker ComposePythonFlaskMongoDB



Python PostgreSQL App

Applicazione basata su PostgreSQL con Python e Flask.
DockerDocker ComposePythonFlask