# Toewijzing - QR-bestand delen op netwerk

Bestandsoverdracht is het proces van het kopiëren of verplaatsen van een bestand van de ene computer naar de andere via een netwerk of internetverbinding. In deze zelfstudie gaan we stap voor stap in op hoe u client/server Python-scripts kunt schrijven die daarmee omgaan.

Het basisidee is om een ​​server te maken die op een bepaalde poort luistert; deze server is verantwoordelijk voor het ontvangen van bestanden (u kunt de server ook bestanden laten verzenden). Aan de andere kant zal de client proberen verbinding te maken met de server en een bestand van elk type te verzenden.

We zullen de module [socket](https://docs.python.org/3/library/socket.html) gebruiken, die bij Python is ingebouwd en ons socketbewerkingen biedt die veel worden gebruikt op het internet, omdat ze achter elke verbinding met elk netwerk zitten.

Houd er rekening mee dat er betrouwbaardere manieren zijn om bestanden over te zetten met tools zoals rsync of scp. Het doel van deze tutorial is echter om bestanden over te zetten met de programmeertaal Python en zonder enige tool van derden.

Eerst moeten we de ***tqdm***-bibliotheek installeren, waarmee we mooie voortgangsbalken kunnen afdrukken:

pip3 installeer tqdm

## Klantcode

Laten we beginnen met de opdrachtgever, de afzender:

import socket  
import tqdm  
import os  
  
SEPARATOR = "<SEPARATOR>"  
BUFFER\_SIZE = 4096 # send 4096 bytes each time step

We moeten het IP-adres specificeren, de poort van de server waarmee we verbinding willen maken en de naam van het bestand dat we willen verzenden.

# the ip address or hostname of the server, the receiver  
host = "192.168.1.101"  
# the port, let's use 5001  
port = 5001  
# the name of file we want to send, make sure it exists  
filename = "data.csv"  
# get the file size  
filesize = os.path.getsize(filename)

De bestandsnaam moet in de huidige map staan, of u kunt een absoluut pad naar dat bestand ergens op uw computer gebruiken. Dit is het bestand dat u wilt verzenden.

os.path.getsize(bestandsnaam) krijgt de grootte van dat bestand in bytes; dat is geweldig, want we hebben het nodig voor het afdrukken van voortgangsbalken in de client en de server.

Laten we de TCP-socket maken:

# create the client socket  
s = socket.socket()

Verbinding maken met de server:

print(f"[+] Connecting to {host}:{port}")  
s.connect((host, port))  
print("[+] Connected.")

connect() methode verwacht een adres van het paar (host, poort) om de socket te verbinden met dat externe adres. Zodra de verbinding tot stand is gebracht, moeten we de naam en grootte van het bestand verzenden:

# send the filename and filesize  
s.send(f"{filename}{SEPARATOR}{filesize}".encode())

Ik heb hier SEPARATOR gebruikt om de gegevensvelden te scheiden; het is gewoon een ongewenste boodschap, we kunnen send() gewoon twee keer gebruiken, maar misschien willen we dat toch niet doen. encode() functie codeert de string die we hebben doorgegeven aan ‘utf-8’-codering (dat is noodzakelijk).

Nu moeten we het bestand verzenden, en terwijl we het bestand verzenden, zullen we mooie voortgangsbalken afdrukken met behulp van de tqdm-bibliotheek:

# start sending the file  
progress = tqdm.tqdm(range(filesize), f"Sending {filename}", unit="B", unit\_scale=True, unit\_divisor=1024)  
with open(filename, "rb") as f:  
 while True:  
 # read the bytes from the file  
 bytes\_read = f.read(BUFFER\_SIZE)  
 if not bytes\_read:  
 # file transmitting is done  
 break  
 # we use sendall to assure transimission in   
 # busy networks  
 s.sendall(bytes\_read)  
 # update the progress bar  
 progress.update(len(bytes\_read))  
# close the socket  
s.close()

Kortom, wat we hier doen, is het bestand openen als binair gelezen, stukjes uit het bestand lezen (in dit geval 4096 bytes of 4 KB) en ze naar de socket sturen met de functie sendall(), en dan werken we de voortgang bij bar elke keer, als dat klaar is, sluiten we dat stopcontact.

## Servercode

Oké, dus we zijn klaar met de klant. Laten we in de server duiken, dus open een nieuw leeg Python-bestand en:

import socket  
import tqdm  
import os  
# device's IP address  
SERVER\_HOST = "0.0.0.0"  
SERVER\_PORT = 5001  
# receive 4096 bytes each time  
BUFFER\_SIZE = 4096  
SEPARATOR = "<SEPARATOR>"

Ik heb enkele parameters geïnitialiseerd die we gaan gebruiken. Merk op dat ik “0.0.0.0” heb gebruikt als het IP-adres van de server. Dit betekent alle IPv4-adressen die zich op de lokale computer bevinden. Je vraagt ​​je misschien af ​​waarom we niet gewoon ons lokale IP-adres of “localhost” of “127.0.0.1” gebruiken? Welnu, als de server twee IP-adressen heeft, laten we zeggen “192.168.1.101” op een netwerk en “10.0.1.1” op een ander, en de server luistert op “0.0.0.0”, is hij bereikbaar op beide IP’s.

Als alternatief kunt u uw openbare of privé-IP-adres gebruiken, afhankelijk van uw klanten. Als de aangesloten clients zich in uw lokale netwerk bevinden, moet u uw privé-IP gebruiken (u kunt dit controleren met het commando ipconfig in Windows of het commando ifconfig in Mac OS/Linux), maar als u clients van internet verwacht , moet u zeker uw openbare adres gebruiken.

Zorg er ook voor dat u dezelfde poort in de server gebruikt als in de client.

Laten we onze TCP-socket maken:

# create the server socket  
# TCP socket  
s = socket.socket()

Dit is nu anders dan de client en we moeten de socket die we zojuist hebben gemaakt binden aan onze SERVER\_HOST en SERVER\_PORT:

# bind the socket to our local address  
s.bind((SERVER\_HOST, SERVER\_PORT))

Daarna gaan we luisteren naar verbanden:

# enabling our server to accept connections  
# 5 here is the number of unaccepted connections that  
# the system will allow before refusing new connections  
s.listen(5)  
print(f"[\*] Listening as {SERVER\_HOST}:{SERVER\_PORT}")

Zodra de client verbinding heeft gemaakt met onze server, moeten we die verbinding accepteren:

# accept connection if there is any  
client\_socket, address = s.accept()   
# if below code is executed, that means the sender is connected  
print(f"[+] {address} is connected.")

Onthoud dat wanneer de client is verbonden, deze de naam en grootte van het bestand zal verzenden. Laten we ze ontvangen:

# receive the file infos  
# receive using client socket, not server socket  
received = client\_socket.recv(BUFFER\_SIZE).decode()  
filename, filesize = received.split(SEPARATOR)  
# remove absolute path if there is  
filename = os.path.basename(filename)  
# convert to integer  
filesize = int(filesize)

Zoals eerder vermeld, worden de ontvangen gegevens gecombineerd met de bestandsnaam en de bestandsgrootte, en we kunnen ze gemakkelijk extraheren door ze te splitsen door de SEPARATOR-reeks.

Daarna moeten we het absolute pad van het bestand verwijderen omdat de afzender het bestand heeft verzonden met zijn eigen bestandspad, dat kan verschillen van het onze, de functie os.path.basename() retourneert het laatste onderdeel van een padnaam.

Nu moeten we het bestand ontvangen:

# start receiving the file from the socket  
# and writing to the file stream  
progress = tqdm.tqdm(range(filesize), f"Receiving {filename}", unit="B", unit\_scale=True, unit\_divisor=1024)  
with open(filename, "wb") as f:  
 while True:  
 # read 1024 bytes from the socket (receive)  
 bytes\_read = client\_socket.recv(BUFFER\_SIZE)  
 if not bytes\_read:   
 # nothing is received  
 # file transmitting is done  
 break  
 # write to the file the bytes we just received  
 f.write(bytes\_read)  
 # update the progress bar  
 progress.update(len(bytes\_read))  
  
# close the client socket  
client\_socket.close()  
# close the server socket  
s.close()

Niet geheel anders dan de klantcode. We openen het bestand hier echter als binair schrijven en gebruiken de methode recv(BUFFER\_SIZE) om BUFFER\_SIZE-bytes van de client-socket te ontvangen en naar het bestand te schrijven. Als dat klaar is, sluiten we zowel de client- als de server-sockets.

## De code testen

\_**Leer ook:** Hoe u alle bestanden en mappen op een FTP-server kunt weergeven met Python

Oké, laat me het proberen op mijn eigen privénetwerk:

C:\> python receiver.py  
  
[\*] Listening as 0.0.0.0:5001

Ik moet naar mijn Linux-box gaan en een voorbeeldbestand verzenden:

yilmaz@intec:~/tools# python3 sender.py  
[+] Connecting to 192.168.1.101:5001  
[+] Connected.  
Sending data.npy: 9%|███████▊ | 45.5M/487M [00:14<02:01, 3.80MB/s]

Laten we de server nu eens bekijken:

[+] ('192.168.1.101', 47618) is connected.  
Receiving data.npy: 33%|███████████████████▍ | 160M/487M [01:04<04:15, 1.34MB/s]

Geweldig, we zijn klaar!

U kunt deze code nu voor uw eigen behoeften uitbreiden. Hier zijn enkele voorbeelden die u kunt implementeren:

* De server in staat stellen om meerdere bestanden van meerdere clients tegelijkertijd te ontvangen met behulp van threads.
* De bestanden comprimeren voordat ze worden verzonden, wat de overdrachtsduur kan verlengen. Als de doelbestanden die u wilt verzenden afbeeldingen zijn, kunt u afbeeldingen optimaliseren door ze te comprimeren.
* Het bestand coderen voordat het wordt verzonden om ervoor te zorgen dat niemand dat bestand kan onderscheppen en lezen.
* Ervoor zorgen dat het bestand correct wordt verzonden door de checksums van beide bestanden te controleren (het originele bestand van de afzender en het verzonden bestand in de ontvanger). In dit geval hebt u veilige hash-algoritmen nodig om dit te doen.
* Een chatroom toevoegen zodat u zowel kunt chatten als bestanden kunt overbrengen.

Veel geluk!