Client/Server Communication using Docker

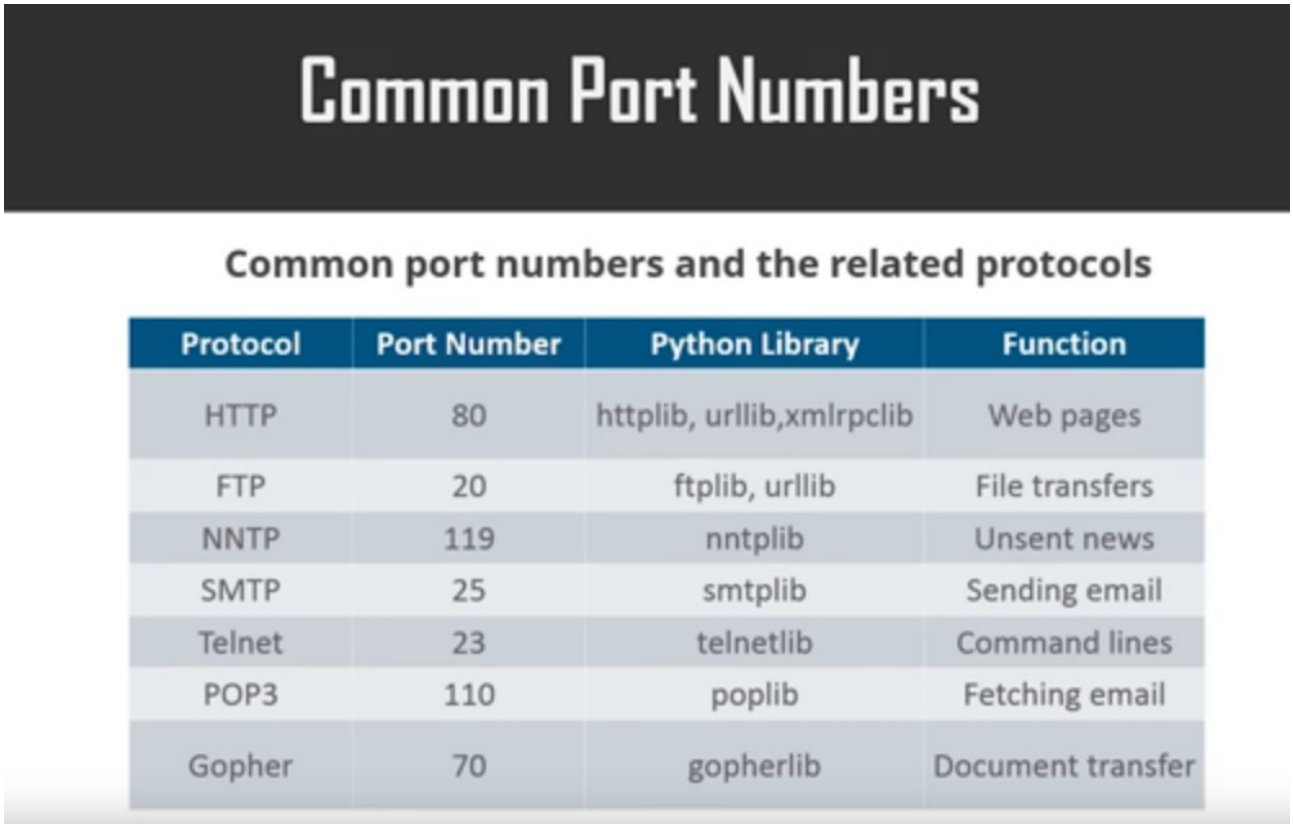
Sockets:

Sockets are the low-level endpoints used for processing information across a network. Some common pro-tocols like HTTP, FTP rely on sockets to make connections. Socket Programming is the route of connecting two points on a network to communicate with each other.

Sockets are interior end-point built for sending and receiving data

A single Network will have two sockets

Sockets are a combination of ip address and a port.



Important Methods of Socket Module:

Socket.socket(): Used to create sockets (requires at both client as well as server end to create socket)

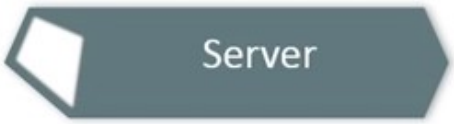
Socket.accept(): Used to accept a connection. It returns a pair of values(conn, address)

socket.bind(): used to bind to the address that is specified as parameter.

Socket.close(): used to mark the socket as close.

Socket.connect(): used to connect to remote address that is specified as parameter.

Socket.listen(): used to enable the server to accept connections.



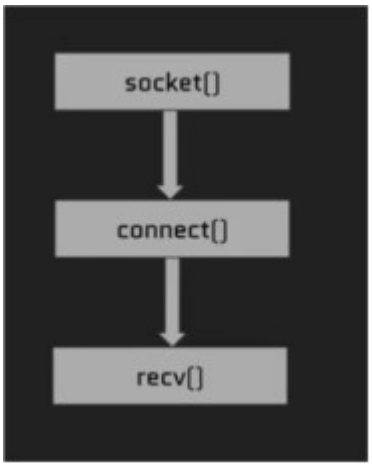
- Either a program, a computer, or a device
- Devoted to managing network resources
- Can be on the same device or computer **or** local or even remote

- Computer or software that receives information or services
- Clients requests for services from servers
- The best example is a web browser

Client Socket Workflow

The client socket is created with a socket() call, and then connected to a remote address with the connect()

call, and then finally can retrieve data with the recv() call

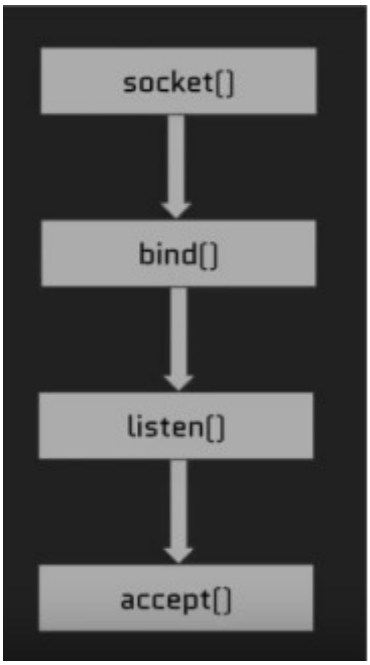


Client:

```
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("localhost", 1024))
msg=s.recv(1024)
print(msg.decode("utf-8"))
```

Server Socket workflow

On the "Server" end of the socket, we need to also create a socket with a socket() call, but then, we need to bind() that socket to an IP and port where it can then listen() for connections, and then finally accept() a connection and then send() or recv() data to the other sockets it has connected to



Server:

```
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("localhost",1024))
s.listen(5)
```

```
while True:
    clt, adr=s.accept()
    stradr=str(adr)
    print("connection to"+stradr+" established")
    string = "Socket programming in python."
    arr = string.encode("utf-8")
    print(arr)
    clt.send(arr)
```

Docker

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application.

Docker containers that run on Docker Engine:

- **Standard:** Docker created the industry standard for containers, so they could be portable anywhere
- **Lightweight:** Containers share the machine’s OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs
- **Secure:** Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry

Commands:

sudo apt update (to update the repositories)

sudo apt install docker.io (to install docker)

sudo apt systemctl start docker (to start docker service)

sudo apt systemctl enable docker (to enable docker to keep on running in background)

sudo docker images (to see images available in docker)

sudo docker pull ubuntu (to download ubuntu image)

sudo docker –name containername -it ubuntu (to create container)

sudo docker ps -a (to see how many containers are created and their state)

watch ‘sudo docker ps -a’ (to continuously watch containers state)

exit (to exit from container)

sudo docker exec -it containername bash (to execute container again from that point where you exited)

vi (to check whether it started from that point or not)