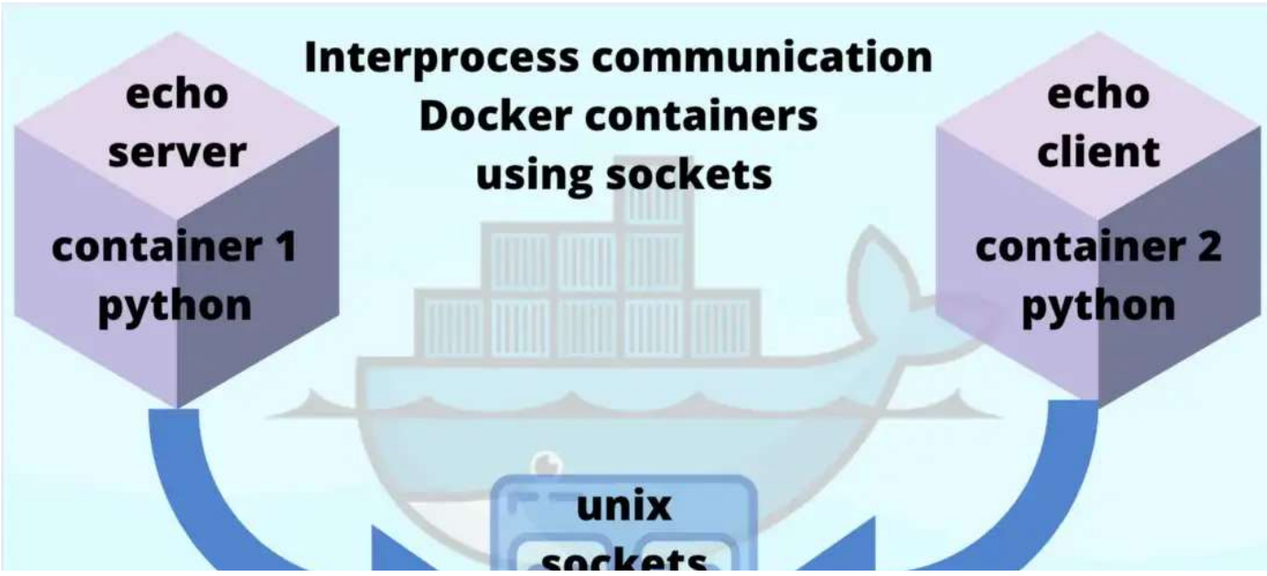
Published in Techanic

Aniket Pingley, Ph. D Follow

May 26, 2020 · 2 min read · Listen



Docker Containers: IPC using Sockets — Part 2

Implementing an echo server and client using Python and deploying them in Docker containers.

In the first part of the this blog, I have demonstrated IPC using sockets in Python between a Docker container that is running the server code and a client script running on the host. The server echoes the message back to the client.

> *The short of it: Enable networking in docker — application inside the container is agnostic to the changes — Container should be named (using — name flag), which become the DNS name for that container.*

As the first step, create a directory titled ‘server’. Inside that directory create the following two files - ‘ipc_server.py’ and ‘Dockerfile’. In the file ‘ipc_server.py’, replace line #6 with the following:

```
HOST = socket.gethostbyname('ipc_server_dns_name')
```

```python
#!/usr/bin/env python3
# ipc_server.py

import socket

HOST = '127.0.0.1'  # Standard loopback interface address (localhost)
PORT = 9898        # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()

    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)
```

ipc_server.py hosted with ❤ by GitHub view raw

Build the docker container for ipc_server using following command from inside the ‘server’ directory.

```
>> docker build -t my_ipc_server .
```

Now create a directory titled ‘client’. Inside that directory create the following two files — ‘ipc_server.py’ and ‘Dockerfile’. In the file ‘ipc_client.py’, replace line #6 with the following. Note that the hostname ‘ipc_server_dns_name’ must be consistent with the ipc_server.py and the ‘docker run’ command, which we will see later in the post:

```
HOST = socket.gethostbyname('ipc_server_dns_name')
```

Build the docker container for ipc_client using following command from inside the ‘client’ directory.

```
>> docker build -t my_ipc_client .
```

If you are running on Linux, running both containers can communicate with each other using following commands (server container first):

```
>> docker run -rm --network=host --name ipc_server_dns_name
my_ipc_server
>> docker run -rm --network=host my_ipc_client
```

On MacOS and Windows, Docker runs inside a virtual machine. Thus, ‘network=host’ will not bind the IP address for DNS name ‘ipc_server_dns_name’ to host. We must define bridged network in Docker for containers to be able to communicate via sockets. Use following command to create a network:

```
>> docker network create my_socket_ipc_network
```

Now, running both containers using following commands will enable the server and client containers to communicate with each other(server container first):

```
>> docker run -rm --network=my_socket_ipc_network --name
ipc_server_dns_name my_ipc_server

>> docker run -rm --network=my_socket_ipc_network my_ipc_client
```

You should see the message ‘Hello, world. IPC success!’ printed on the terminal of client. The message is simply echoed by the server.

Docker Python Sockets Containers