Docker Bridge Networks and Host Service Accessibility: A Deep Dive into Container Networking Isolation

Abdallah Abouabdallah

January 2026

Abstract

This article documents a practical networking challenge encountered while deploying pgAdmin in a Docker container to manage a PostgreSQL database accessible only through an SSH reverse tunnel. We explore Docker's bridge network isolation mechanisms, the limitations of `host-gateway` resolution, and why containers cannot connect to services bound to their bridge gateway IP. The investigation reveals fundamental Docker networking design decisions and presents alternative architectural solutions.

1 Introduction

Modern infrastructure often involves multiple layers of network abstraction: Docker networks, SSH tunnels, reverse proxies, and firewall restrictions. When these layers interact unexpectedly, debugging requires understanding each component's behavior in isolation and in combination.

1.1 The Objective

Deploy pgAdmin (a PostgreSQL management interface) on a VPS to manage a PostgreSQL database running on a remote workstation. The workstation resides on a corporate network with strict egress policies, requiring all remote access to be established through secure, encrypted channels.

1.2 The Network Environment

The workstation's network enforces TLS-only outbound connections on standard ports. To maintain secure remote access while complying with network policies, we use **stunnel**—an SSL/TLS tunneling proxy that provides encrypted communication channels. This is a common enterprise pattern for secure remote administration.

2 Architecture Overview

3 The Problem

The SSH reverse tunnel successfully forwards PostgreSQL traffic and binds to `172.19.0.1:5432` on the VPS (the gateway IP of the Docker `traefik_default` network). From the VPS host, the connection works:

```
# From VPS host - SUCCESS
$ nc -zv 172.19.0.1 5432
Connection to 172.19.0.1 5432 port [tcp/postgresql] succeeded!

$ psql -h 172.19.0.1 -U abdallah -d postgres
# Connected successfully
```

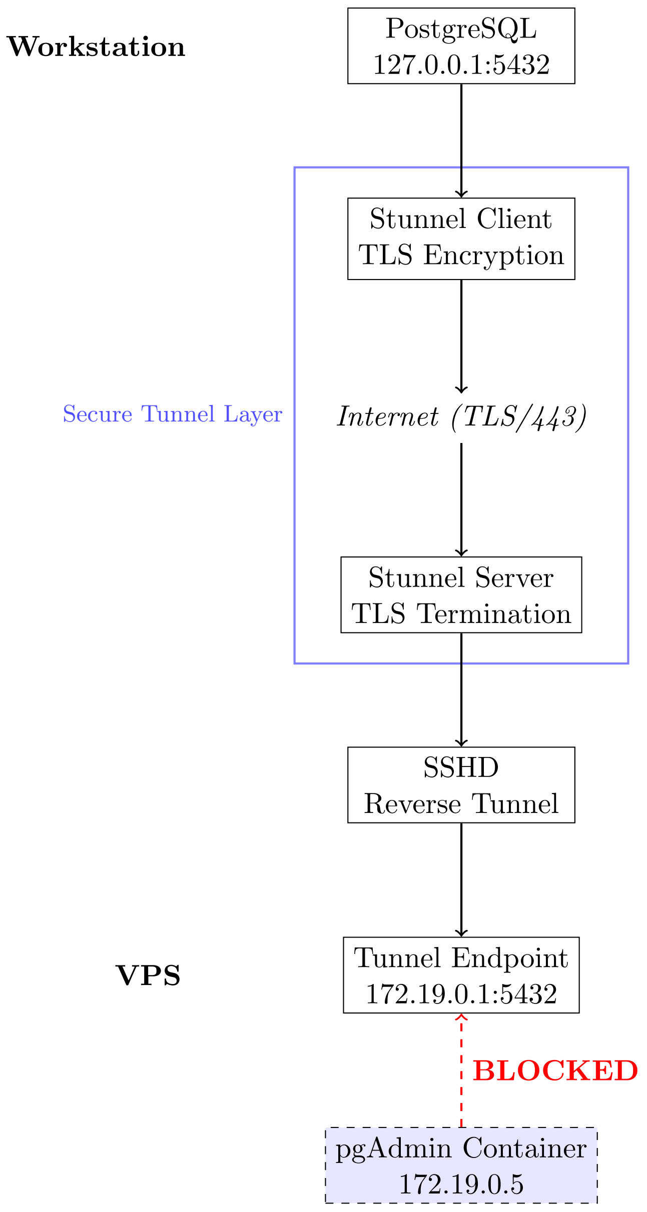

Workstation
PostgreSQL 127.0.0.1:5432
Secure Tunnel Layer
Stunnel Client TLS Encryption
Internet (TLS/443)
Stunnel Server TLS Termination
SSHD Reverse Tunnel
VPS
Tunnel Endpoint 172.19.0.1:5432
BLOCKED
pgAdmin Container 172.19.0.5

Figure 1: Network architecture showing the blocked connection. The secure tunnel layer is detailed in a separate article.

However, from inside the pgAdmin container:

```
# From pgAdmin container - FAILURE
$ nc -zv 172.19.0.1 5432 -w 3
nc: 172.19.0.1 (172.19.0.1:5432): Operation timed out
```

4 Investigation

4.1 Understanding Docker Bridge Networks

Docker bridge networks create isolated network segments. Each bridge network has:

- A **subnet** (e.g., `172.19.0.0/16`)
- A **gateway IP** (e.g., `172.19.0.1`) — this is the host's interface to the bridge
- **Container IPs** assigned from the subnet (e.g., `172.19.0.5`)

4.2 The Gateway Misconception

The gateway IP (`172.19.0.1`) is **not** a general-purpose entry point to host services. It exists primarily for:

1. Outbound NAT (containers accessing the internet)
2. Docker's internal DNS resolution
3. Inter-container routing

While Docker doesn't explicitly block container→gateway connections, the interaction between several layers creates this behavior:

- **SSH tunnel binding semantics**: How `sshd` binds reverse tunnels to specific interfaces
- **Network namespace routing**: Packets from containers traverse different routing paths than host-originated packets
- **Host firewall rules**: UFW, nftables, or VPS-level filtering may affect traffic patterns

The result is that services bound to the gateway IP from the host's perspective may not be reachable from containers, even though they appear to be on the same subnet.

4.3 The host-gateway Special Value

Docker provides `host-gateway` as a special DNS value:

```
extra_hosts:
  - "host.docker.internal:host-gateway"
```

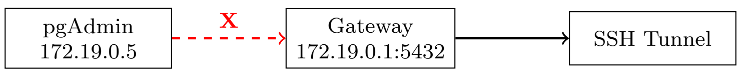
This is a design simplification: `host-gateway` always resolves to `172.17.0.1` (the **default** bridge gateway), regardless of which network the container is attached to. For containers on custom networks like `traefik_default`, this creates a mismatch:

```
# Inside container on traefik_default (172.19.0.0/16)
$ cat /etc/hosts | grep host.docker
172.17.0.1      host.docker.internal   # Wrong network!
```

5 Attempted Solutions

5.1 Approach 1: Direct Gateway Connection

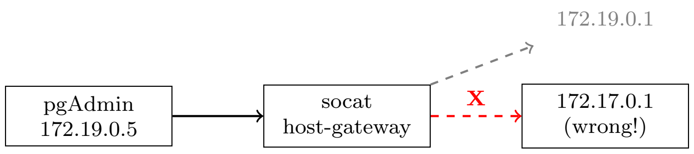
Bind the SSH tunnel to the Docker gateway IP and have containers connect directly.



Result: Failed. The SSH tunnel bound to the gateway IP is not reachable from within the container's network namespace.

5.2 Approach 2: Socat Bridge with host-gateway

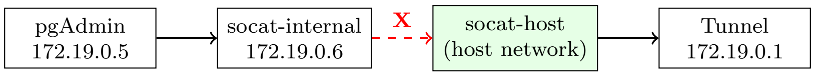
Use a socat container with `host-gateway` to bridge traffic to the host.



Result: Failed. `host-gateway` resolves to default bridge (172.17.0.1), not our network's gateway (172.19.0.1).

5.3 Approach 3: Dual Socat Bridge

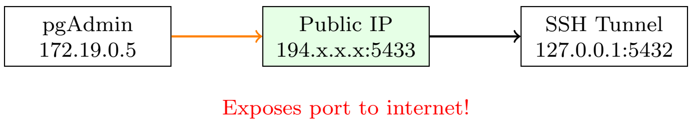
Chain two socat instances: one in host network mode, one in the Docker network.



Result: Failed. socat-internal still can't reach socat-host via gateway IP—same isolation issue.

5.4 Approach 4: Public IP Access

Have containers connect via the VPS's public IP where socat listens.



Result: Would work technically, but exposes PostgreSQL proxy to the public internet—unacceptable security risk.

6 Working Solution

Run pgAdmin directly on the host (without Docker), connecting to `127.0.0.1:5432`. Use Traefik's file provider or a reverse proxy configuration to route HTTPS traffic to the local pgAdmin instance.

```
# pgAdmin connects directly to localhost
pgAdmin -> 127.0.0.1:5432 -> SSH Tunnel -> Albus PostgreSQL
```

This eliminates the Docker networking layer entirely for this specific service.

7 Key Takeaways

1. **Gateway accessibility is not guaranteed**: Services bound to Docker bridge gateway IPs may not be reachable from containers due to network namespace routing and SSH tunnel binding behavior.

2. **host-gateway is a simplification**: It always resolves to the default bridge gateway (`172.17.0.1`), not the container's actual network gateway—a design choice, not a security feature.

3. **SSH tunnel bind addresses matter**: `-R 127.0.0.1:5432` vs `-R 0.0.0.0:5432` have very different accessibility implications.

4. **GatewayPorts in sshd_config**: Required for SSH tunnels to bind to non-localhost addresses.

5. **Sometimes Docker isn't the answer**: For services requiring complex host networking, native installation may be simpler.

8 Conclusion

This investigation revealed that the interaction between multiple networking layers—SSH reverse tunnels, network namespaces, and Docker bridge networks—can create unexpected connectivity challenges. The behavior emerged not from any single component's design, but from how these layers interact. The combination of SSH tunneling, TLS encryption, and Docker networking created a multi-layer debugging challenge where each component worked correctly in isolation, but their interaction produced the observed failure.

Understanding these boundaries helps inform better architectural decisions: not every service benefits from containerization, especially when complex host networking is required.