# SSH 隧道技术

杨岢瑞

Jun 24, 2026

## 1　为什么还要谈 SSH 隧道?

当远程办公成为常态时，开发人员经常遇到这样的困境：公司内网的数据库仅允许办公网段访问，而在家或出差时需要直接连接生产环境的 MySQL 或 Redis；容器化环境里云主机往往只开放 22 端口，却需要把本地 IDE 连到集群内部的 3306 或 6379 端口。面对这些场景，常见的方案包括 VPN、跳板机或堡垒机，但它们要么需要部署额外的网关，要么会把全部流量都拉到一条加密通道里，运维成本和带宽开销都相当可观。SSH 隧道则不同，它利用已经存在的 SSH 连接，在传输层把本地端口映射到远端地址，或者反过来把远端端口映射到本地，无需额外组件即可完成跨网访问，因此在轻量级、零依赖的场景中被广泛采用。本文面向运维、开发与安全三类读者，系统梳理本地转发、远程转发、动态转发三种隧道模式，并结合多级跳板、持久化方案与安全加固给出生产实践建议。

## 2　核心原理：SSH 到底做了什么?

SSH 协议在建立连接时先完成密钥交换与会话加密，随后通过双向认证确认双方身份。隧道功能正是建立在这条加密通道之上的：当客户端执行端口转发参数时，会在本地或远端打开一个监听 socket；任何发往该 socket 的 TCP 流量都会被 SSH 客户端封装进加密通道，再由服务端解封装并转发到真正的目标地址。反之亦然，服务端收到的流量也能通过同一通道回送到客户端。这种映射关系可以理解为在传输层做了一次地址转换，只不过转换过程完全发生在加密通道内部，对应用层透明。需要注意的是，SSH 隧道本质上是 TCP 转发，UDP 与 ICMP 流量不会被处理，这一点与全流量 VPN 形成明显差异。

## 3　本地转发（-L）：把「远端服务」拉到「本地」

本地转发使用参数 `-L` 把远端某个地址映射到本机端口。以命令 `ssh -L 63306:db.internal:3306 user@jump.host` 为例，SSH 客户端首先在本地 63306 端口建立监听 socket；当应用通过该端口发起连接时，客户端把请求封装进已建立的 SSH 会话，跳板机收到后立即与内网数据库 3306 端口建立新的 TCP 连接，并把双向数据在两个 socket 之间透明转发。执行环境为 Linux 或 macOS 时，可在终端直接输入该命令；Windows 用户可在 PowerShell 或 WSL 中运行相同语法。需要注意的是，sshd 默认配置 `GatewayPorts no` 导致监听地址只能是 127.0.0.1，若需让其他主机也能访问，需在服务端配置 `GatewayPorts clientspecified` 并在防火墙中做相应限制。多级跳板可通过 `-J` 参数实现，例如 `ssh -L 63306:db.internal:3306 -J jump1,jump2 user@target`，此时 SSH 会依次登录 jump1、jump2，最终在目标主机建立隧道，整个过程对

用户透明。

## 4 远程转发（-R）：把「本地服务」暴露到「远端」

远程转发使用参数 `-R` 把本机服务映射到远端端口。以命令 `ssh -R 8080:localhost:3000 user@public.host` 为例，SSH 客户端在登录公网主机后，会在公网主机的 8080 端口建立监听 socket；当外部请求到达该端口时，公网主机会把流量封装进 SSH 通道，回送到客户端，再由客户端转发给本地的 3000 端口。这种模式常用于内网服务需要被公网回调的场景，例如 Webhook 触发或临时把笔记本的 22 端口暴露出去做反向 SSH。安全角度看，公网端口默认可被任意 IP 访问，因此必须配合 `GatewayPorts clientspecified` 并在防火墙中限制来源 IP；同时建议使用 `autossh` 或 systemd service 实现隧道后台常驻，避免 SSH 进程因网络波动而中断。

## 5 动态转发（-D）与 SOCKS5 代理

动态转发使用参数 `-D` 把 SSH 客户端变成一个 SOCKS5 网关。以命令 `ssh -D 1080 user@jump.host` 为例，客户端在本地 1080 端口启动 SOCKS5 服务；当浏览器或终端把流量指向该端口时，SSH 会根据目标地址动态建立隧道，把请求转发到跳板机，再由跳板机访问真实目标。配置示例：在 SwitchyOmega 中新建 SOCKS5 代理，地址填 127.0.0.1，端口填 1080；终端则可使用 `proxychains` 包装命令，例如 `proxychains curl https://example.com`。与 VPN 相比，动态转发仅处理 TCP 流量，UDP 与 ICMP 不会被转发，因此适合需要快速代理特定服务的场景；若需级联跳板，可写成 `ssh -D 1080 -J bastion user@target`，此时 SOCKS5 网关本身就运行在多级跳板链的末端。

## 6 多级跳板与跳板链（-J / ProxyJump）

传统写法需要在每台跳板机上手动登录并保持会话，而新语法 `-J` 把多级登录封装成一次命令。配置文件 `~/.ssh/config` 中可写成：

```
Host bastion
   HostName 10.0.0.1
   User ops
Host target
   HostName 10.1.0.10
   User app
   ProxyJump bastion
```

执行 `ssh target` 时，客户端会先登录 bastion，再从 bastion 登录 target，整个过程自动完成。需要注意的是，每增加一级跳板都会引入往返延迟与加密开销；生产环境建议把跳板数量控制在两到三级以内，并配合 `ControlMaster` 与 `ControlPersist` 复用连接，降低握手成本。

# 7 生产级实践与运维技巧

为使隧道长期稳定运行，可使用 autossh 自动重连，或编写 systemd unit 文件。以 systemd 为例，创建 /etc/systemd/system/ssh-tunnel.service，内容如下：

```
[Unit]
Description=SSH Tunnel to internal DB
After=network.target

[Service]
Type=simple
ExecStart=/usr/bin/ssh -N -L 63306:db.internal:3306 user@jump.host
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

执行 systemctl enable --now ssh-tunnel 即可让隧道随系统启动。性能调优方面，~/.ssh/config 中加入 ControlMaster auto 与 ControlPersist 600 可复用同一连接，避免频繁握手；同时设置 ServerAliveInterval 30 可在网络空闲时发送心跳，防止中间设备超时断开。容器场景下，kubectl port-forward 与 SSH 隧道各有优劣：前者依赖 kubeconfig，后者只需一条 SSH 通道；在 sidecar 中运行 SSH 隧道可让 Pod 直接访问物联设备，而无需在集群内额外暴露端口。

# 8 安全加固 Checklist

访问控制上，sshd_config 中可写 AllowTcpForwarding yes 并配合 Match User tunneluser PermitOpen db.internal:3306 做细粒度限制；同时使用 PermitOpen 白名单可防止用户随意转发任意端口。密钥管理方面，禁止密码登录，强制使用密钥并设置 passphrase；更进一步可引入 SSH CA 签发短时效证书，降低密钥泄露风险。审计与告警上，可用 ForceCommand 把用户 shell 限制为 internal-sftp 或自定义脚本，仅允许隧道操作；同时在 OS 层面用 auditd 监控 sshd 的 TCP forward 系统调用，及时发现异常行为。当 SSH 隧道不再适用时，可考虑 Tailscale、WireGuard 或 API Gateway 作为替代方案，它们在零信任架构下提供了更细粒度的访问控制。

# 9 真实案例复盘

跨国团队需要零信任访问 MongoDB，可在办公网部署一台跳板机，开发人员使用本地转发加上 autossh 自动重连；当网络切换时隧道会在十秒内恢复，避免手动干预。IoT 设备远程调试场景下，可在设备上运行 systemd path 触发器，当检测到本地 3000 端口有流量时才启动远程转发，既节省资源又降低攻击面。家用树莓派可作为按需跳板，通过动态转发把家庭网络变成 SOCKS5 网关，再用 Cloudflare Tunnel 把 1080 端口安全地暴露

到公网，实现零成本远程访问。

SSH 隧道本质上是传输层手段，最终仍需结合零信任与最小权限原则来保障安全。延伸主题包括 mTLS 双向认证、SPIFFE/SPIRE 身份体系以及 API 网关的对比，它们在不同层面解决了跨网访问问题。参考资料可查阅 OpenSSH 官方手册与 RFC 4251 – 4254；勘误声明：如发现文中命令或配置与实际环境不符，欢迎在评论区反馈。