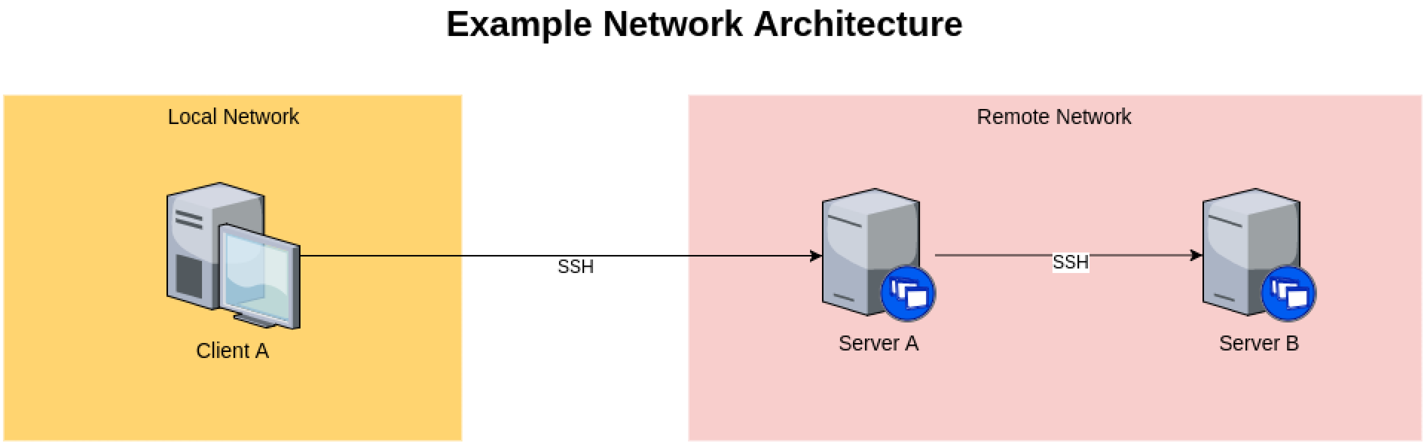
Proposed implementation solution for "Accessing a remote service" challenge

This document covers all of the steps used to implement a solution in order to permit someone to access a service that runs on a remote unreachable port.

Challenge

In this challenge, you will propose a way to access a service running on a remotely unreachable port. Consider the architecture proposed on the figure below:



The only allowed connection between Client A and Server A is via SSH. The only allowed connection between Server A and Server B is via SSH.
We need to access, from Client A and using HTTP, a service running on port 8000 of Server B.

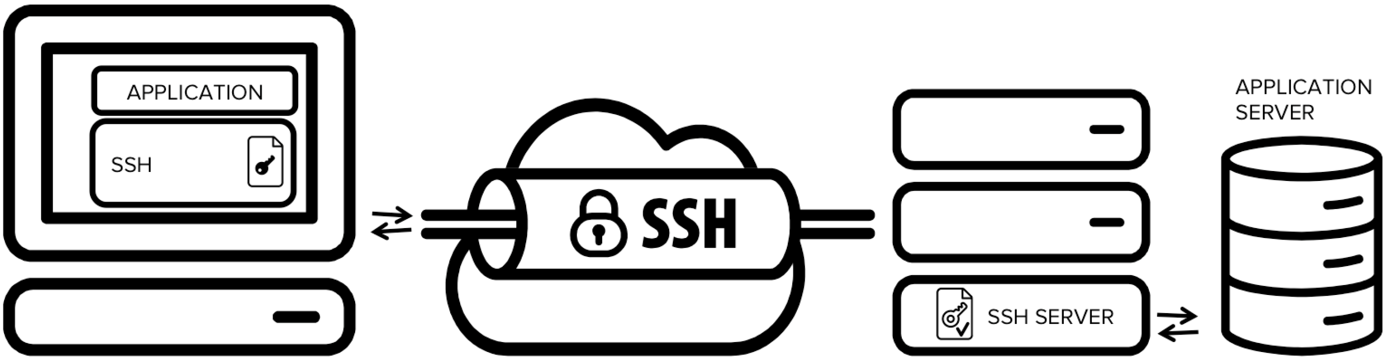
Notes and constraints:

- There is no direct route from Client A to Server B, and no practical way to build one. Imagine the following scenario as example: Server B belongs to a customer internal datacenter, and we were provided with a VPN that allows access to Server A SSH port only.
- There is another service running on port 8000 of Server A, we must not cause impacts on this one

Both client and servers run CentOS 7 without X.

Proposed Solution

SSH tunnels



SSH is useful in a general sense for tunnelling pretty much any kind of TCP traffic, and doing so securely and with appropriate authentication. This can be used both for ad-hoc purposes such as talking to a process on a remote host that’s only listening locally or within a secured network, or for bypassing restrictive firewall rules, to more stable implementations such as setting up a persistent SSH tunnel between two machines to ensure sensitive traffic that might otherwise be sent in cleartext is not only encrypted but also authenticated.

For this challenge, I've used following configurations for the lab:

Client A:

- CentOS 7

CentOS-7-x86_64-DVD-2003

- Local Network IP: 192.168.135.132
- SSH, cURL

Server A:

- CentOS 7

CentOS-7-x86_64-DVD-2003

- nginx
- Remote Network IP: 192.168.99.102
- SSH, lsof

Server B:

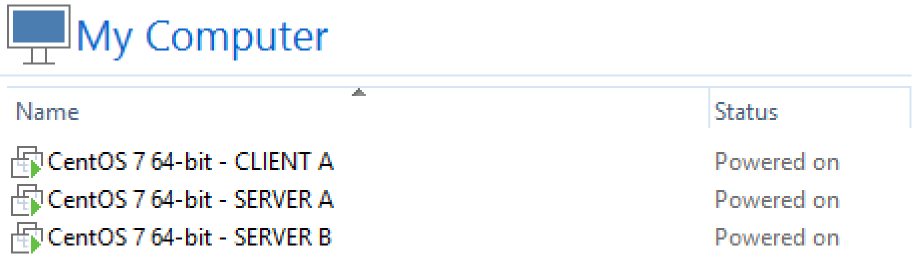
- CentOS 7

CentOS-7-x86_64-DVD-2003

- nginx
- Remote Network IP: 192.168.99.101
- SSH, lsof

Creating a Lab

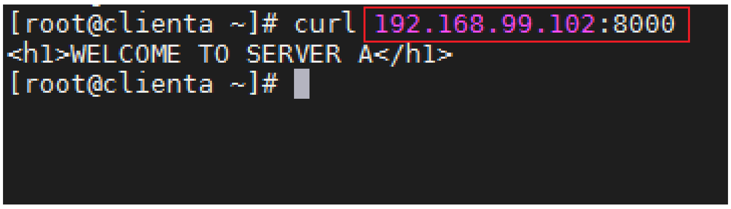
For this lab I've used **VMware Workstation** in order to create the client and servers machines:



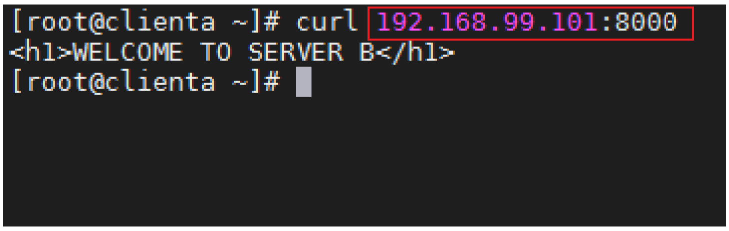
My Computer

Name	Status
CentOS 7 64-bit - CLIENT A	Powered on
CentOS 7 64-bit - SERVER A	Powered on
CentOS 7 64-bit - SERVER B	Powered on

To represent the service running on port 8000 on both Server A and Server B, I've used **nginx**. I've created a simple index.html to represent the homepage as below:



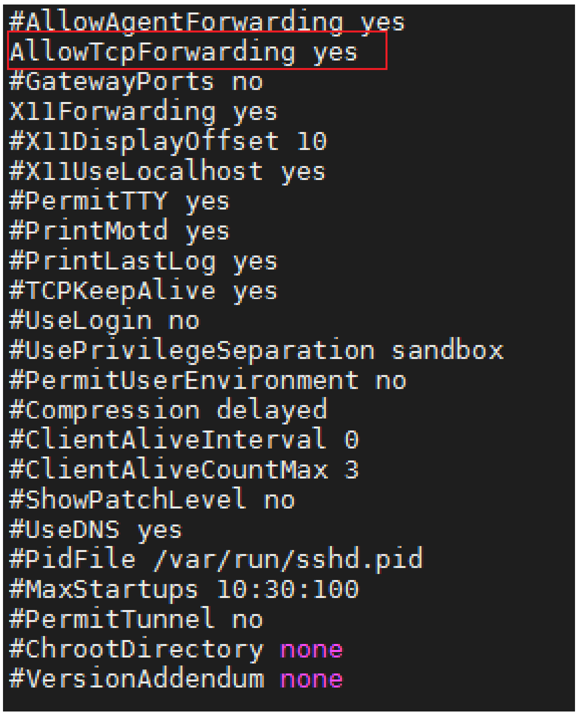
```
[root@clienta ~]# curl 192.168.99.102:8000
<h1>WELCOME TO SERVER A</h1>
[root@clienta ~]#
```



```
[root@clienta ~]# curl 192.168.99.101:8000
<h1>WELCOME TO SERVER B</h1>
[root@clienta ~]#
```

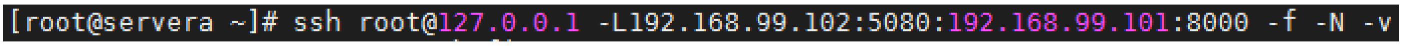
Implementation

For this implementation I've connected on Server A and uncommented the following line on /etc/ssh/sshd_config file, allowing SSH TCP Forwarding usage:

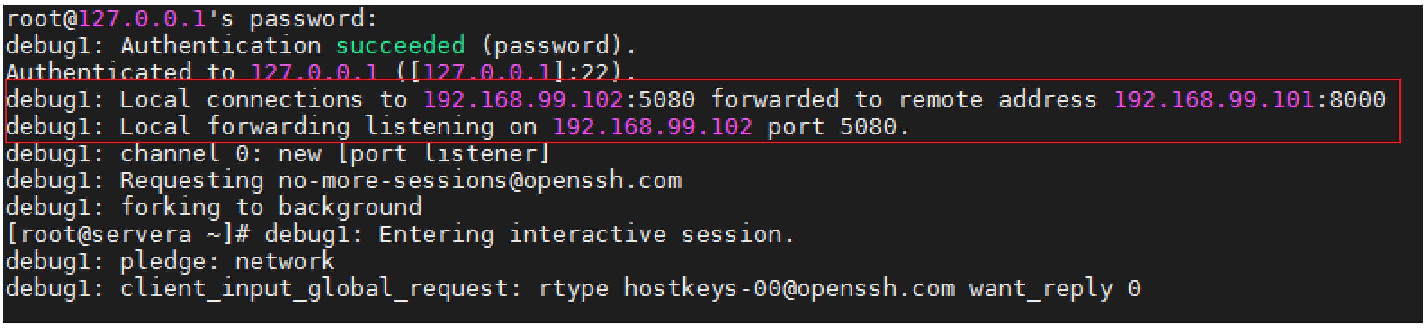


```
#AllowAgentForwarding yes
AllowTcpForwarding yes
#GatewayPorts no
X11Forwarding yes
#X11DisplayOffset 10
#X11UseLocalhost yes
#PermitTTY yes
#PrintMotd yes
#PrintLastLog yes
#TCPKeepAlive yes
#UseLogin no
#UsePrivilegeSeparation sandbox
#PermitUserEnvironment no
#Compression delayed
#ClientAliveInterval 0
#ClientAliveCountMax 3
#ShowPatchLevel no
#UseDNS yes
#PidFile /var/run/sshd.pid
#MaxStartups 10:30:100
#PermitTunnel no
#ChrootDirectory none
#VersionAddendum none
```

After that, I've created an SSH Tunnel between Server A and Server B using the following command:



```
[root@servera ~]# ssh root@127.0.0.1 -L192.168.99.102:5080:192.168.99.101:8000 -f -N -v
```



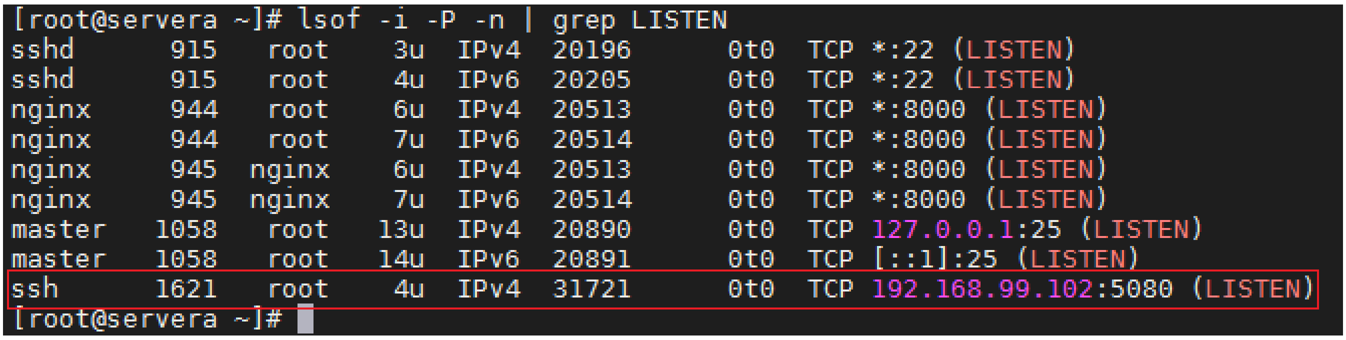
```
root@127.0.0.1's password:
debug1: Authentication succeeded (password).
Authenticated to 127.0.0.1 ([127.0.0.1]:22).
debug1: Local connections to 192.168.99.102:5080 forwarded to remote address 192.168.99.101:8000
debug1: Local forwarding listening on 192.168.99.102 port 5080.
debug1: channel 0: new [port listener]
debug1: Requesting no-more-sessions@openssh.com
debug1: forking to background
[root@servera ~]# debug1: Entering interactive session.
debug1: pledge: network
debug1: client_input_global_request: rtype hostkeys-00@openssh.com want_reply 0
```

Which means:

Local connections to server A on port 5080 are going to be forward to remote server B on port 8000.

Testing

Using the **lsof** command, I could retrieve information about opened ports on Server A in order to check if the port 5080 is LISTEN:

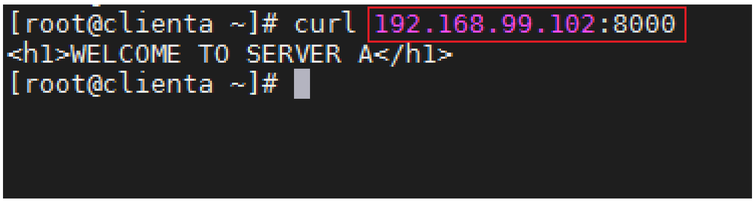


```
[root@servera ~]# lsof -i -P -n | grep LISTEN
sshd      915   root    3u  IPv4  20196      0t0  TCP *:22 (LISTEN)
sshd      915   root    4u  IPv6  20205      0t0  TCP *:22 (LISTEN)
nginx     944   root    6u  IPv4  20513      0t0  TCP *:8000 (LISTEN)
nginx     944   root    7u  IPv6  20514      0t0  TCP *:8000 (LISTEN)
nginx     945  nginx    6u  IPv4  20513      0t0  TCP *:8000 (LISTEN)
nginx     945  nginx    7u  IPv6  20514      0t0  TCP *:8000 (LISTEN)
master   1058   root   13u  IPv4  20890      0t0  TCP 127.0.0.1:25 (LISTEN)
master   1058   root   14u  IPv6  20891      0t0  TCP [::1]:25 (LISTEN)
ssh      1621   root    4u  IPv4  31721      0t0  TCP 192.168.99.102:5080 (LISTEN)
[root@servera ~]#
```

Once that it is, let's check if SSH Tunneling is working properly using cURL:

Server A:

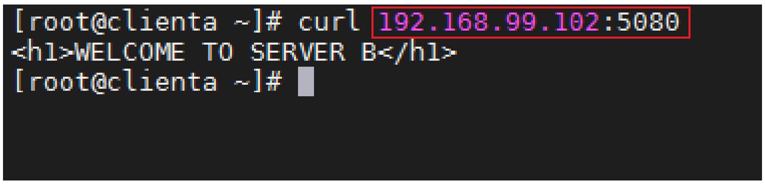
- Service from Server A on port 8000 is running OK, in other words, there were no impact on it.



```
[root@clienta ~]# curl 192.168.99.102:8000
<h1>WELCOME TO SERVER A</h1>
[root@clienta ~]#
```

Server B:

- Service from Server B on port 8000 is running OK using the tunnel from Server A on port **5080**
- Note that we are using the **same IP address** here, but using **different** ports.



```
[root@clienta ~]# curl 192.168.99.102:5080
<h1>WELCOME TO SERVER B</h1>
[root@clienta ~]#
```

Conclusion

Using SSH Tunneling is a good way to solve this challenge, once it's easy to manage and ensures good functionality on both services that are running on Server A and Server B causing no impact.