

KLE Society's
KLE Technological University



**Report
On
SSH Tunneling**

submitted in partial fulfillment of the requirement for the degree of

**Bachelor of Engineering in
Computer Science and Engineering**

Submitted By

Anusha Raikar	01FE18BCS043
Archana Badagi	01FE18BCS045
Atul Kumar	01FE18BCS056
Isha Bhandary	01FE18BCS063

**Under the guidance of
Dr. Vijayalakshmi M**

**SCHOOL OF COMPUTER SCIENCE & ENGINEERING
HUBLI – 580 031 (India).**

Academic year 2021-2022

INDEX

S.No	Content	Page Number
1.	SSH Tunneling	3
2.	Port forwarding	4
	1. Local port forwarding	4
	2. Remote port forwarding	5
	3. Dynamic port forwarding	5
3.	Architecture	7
4.	Software and Hardware Specifications	9
5.	Implementation Details	10
6.	Screenshots of Implementation	13
7.	Result Analysis	19
8.	Conclusion	20
9.	References	21
10.	Video Link	22

1. SSH Tunneling

SSH tunneling (also called SSH port forwarding) is a method of transporting arbitrary networking data over an encrypted SSH connection. It can be used to add encryption to legacy applications. It can also be used to implement VPNs (Virtual Private Networks) and access intranet services across firewalls. SSH is a standard for secure remote logins and file transfers over untrusted networks. It also provides a way to secure the data traffic of any given application using port forwarding, basically tunneling any TCP/IP port over SSH. This means that the application data traffic is directed to flow inside an encrypted SSH connection so that it cannot be eavesdropped or intercepted while it is in transit. SSH tunneling enables adding network security to legacy applications that do not natively support encryption.

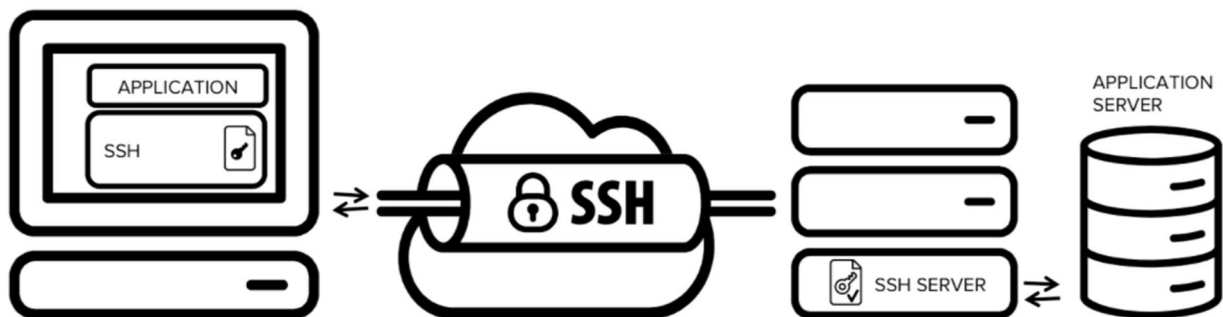


Figure 1: SSH Tunneling

The figure presents a simplified overview of SSH tunneling. The secure connection over the untrusted network is established between an SSH client and an SSH server. This SSH connection is encrypted, protects confidentiality and integrity, and authenticates communicating parties. The SSH connection is used by the application to connect to the application server. With tunneling enabled, the application contacts to a port on the local host that the SSH client listens on. The SSH client then forwards the application over its encrypted tunnel to the server. The server then connects to the actual application server - usually on the same machine or in the same data center as the SSH server. The application communication is thus secured, without having to modify the application or end user workflows. A SSH tunnel can be used to transfer unencrypted traffic over a network through an encrypted channel. For example we can use a SSH tunnel to securely transfer files between a FTP server and a client even though the FTP protocol itself is not encrypted. SSH tunnels also provide a means to bypass firewalls that prohibits or filter certain internet services. For example an organization will block certain sites using their proxy filter. But users may not wish to have their web traffic monitored or blocked by the organization proxy filter. If users can connect to an external SSH server, they can create a SSH tunnel to forward a given port on their local machine to port 80 on remote web-server via the external SSH.

2. Port Forwarding

Port forwarding or port mapping is an application of network address translation (NAT) that redirects a communication request from one address and port number combination to another while the packets are traversing a network gateway, such as a router or firewall. This technique is most commonly used to make services on a host residing on a protected or masqueraded (internal) network available to hosts on the opposite side of the gateway (external network), by remapping the destination IP address and port number of the communication to an internal host.

Port forwarding or port mapping is a name given to the combined technique of

1. translating the address and/or port number of a packet to a new destination
2. possibly accepting such packet(s) in a packet filter (firewall)
3. forwarding the packet according to the routing table.

SSH tunnels can be created in several ways using different kinds of port forwarding mechanisms. Ports can be forwarded in three ways.

1. Local port forwarding
2. Remote port forwarding
3. Dynamic port forwarding

2.1. Local port forwarding

Local port forwarding is the most common type of port forwarding. It is used to let a user connect from the local computer to another server, i.e. forward data securely from another client application running on the same computer as a Secure Shell (SSH) client. By using local port forwarding, firewalls that block certain web pages are able to be bypassed. Connections from an SSH client are forwarded, via an SSH server, to the intended destination server. The SSH server is configured to redirect data from a specified port (which is local to the host that runs the SSH client) through a secure tunnel to some specified destination host and port. The local port is on the same computer as the SSH client, and this port is the "forwarded port". On the same computer, any client that wants to connect to the same destination host and port can be configured to connect to the forwarded port (rather than directly to the destination host and port). After this connection is established, the SSH client listens on the forwarded port and directs all data sent by applications to that port, through a secure tunnel to the SSH server. The server decrypts the data, and then redirects it to the destination host and port. On the command line, "-L" specifies local port forwarding. The destination server, and two port numbers need to be included. Port numbers less than 1024 or greater than 49150 are reserved for the system. Some programs will only work with specific source ports, but for the most part any source port number can be used.

Some uses of local port forwarding:

- Using local port forwarding to Receive Mail
- Connect from a laptop to a website using an SSH tunnel.

2.2. Remote port forwarding

This form of port forwarding enables applications on the server side of a Secure Shell (SSH) connection to access services residing on the SSH's client side. In addition to SSH, there are proprietary tunneling schemes that utilize remote port forwarding for the same general purpose. In other words, remote port forwarding lets users connect from the server side of a tunnel, SSH or another, to a remote network service located at the tunnel's client side. To use remote port forwarding, the address of the destination server (on the tunnel's client side) and two port numbers must be known. The port numbers chosen depend on which application is to be used. Remote port forwarding allows other computers to access applications hosted on remote servers. Two examples:

- An employee of a company hosts an FTP server at their own home and wants to give access to the FTP service to employees using computers in the workplace. In order to do this, an employee can set up remote port forwarding through SSH on the company's internal computers by including their FTP server's address and using the correct port numbers for FTP (standard FTP port is TCP/21)
- Opening remote desktop sessions is a common use of remote port forwarding. Through SSH, this can be accomplished by opening the virtual network computing port (5900) and including the destination computer's address.

2.3. Dynamic port forwarding

Dynamic port forwarding (DPF) is an on-demand method of traversing a firewall or NAT through the use of firewall pinholes. The goal is to enable clients to connect securely to a trusted server that acts as an intermediary for the purpose of sending/receiving data to one or many destination servers. DPF can be implemented by setting up a local application, such as SSH, as a SOCKS proxy server, which can be used to process data transmissions through the network or over the Internet. Programs, such as web browsers, must be configured individually to direct traffic through the proxy, which acts as a secure tunnel to another server. Once the proxy is no longer needed, the programs must be reconfigured to their original settings. Because of the manual requirements of DPF, it is not often used. Once the connection is established, DPF can be used to provide additional security for a user connected to an untrusted network. Since data must pass through the secure tunnel to another server before being forwarded to its original destination, the user is protected from packet sniffing that may occur on the LAN. DPF is a powerful tool with many uses; for example, a user connected to the

Internet through a coffee shop, hotel, or otherwise minimally secure network may wish to use DPF as a way of protecting data. DPF can also be used to bypass firewalls that restrict access to outside websites, such as in corporate networks. Dynamic port forwarding allows you to create a socket on the local (SSH client) machine, which acts as a SOCKS proxy server. When a client connects to this port, the connection is forwarded to the remote (SSH server) machine, which is then forwarded to a dynamic port on the destination machine. This way, all the applications using the SOCKS proxy will connect to the SSH server, and the server will forward all the traffic to its actual destination. A typical example of a dynamic port forwarding is to tunnel the web browser traffic through an SSH server.

3. Architecture

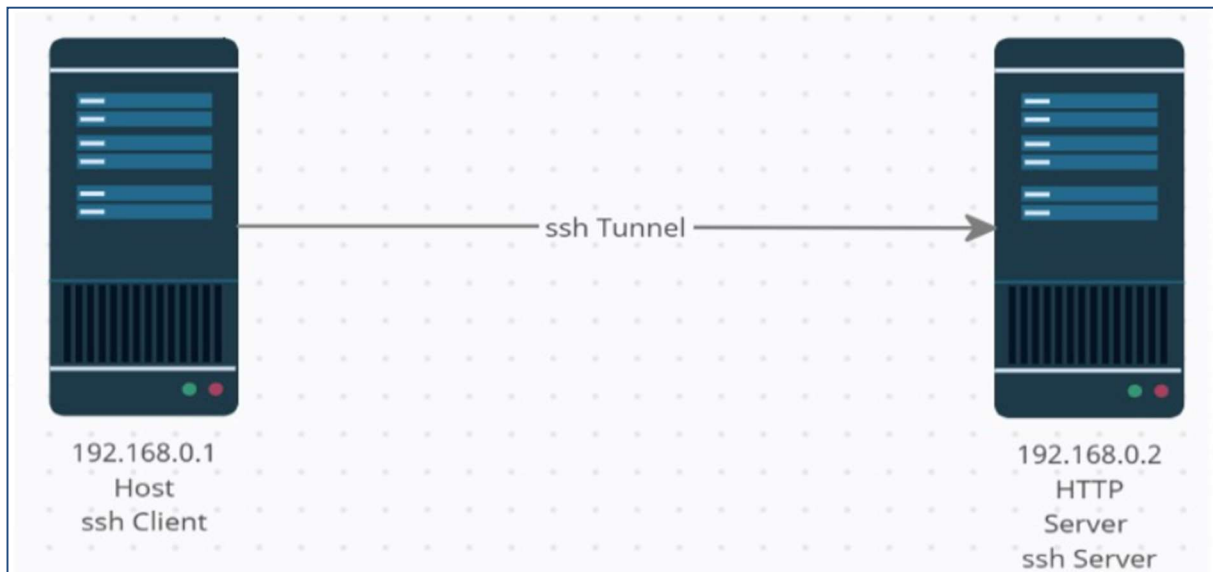


Figure 2: Local Port Forwarding

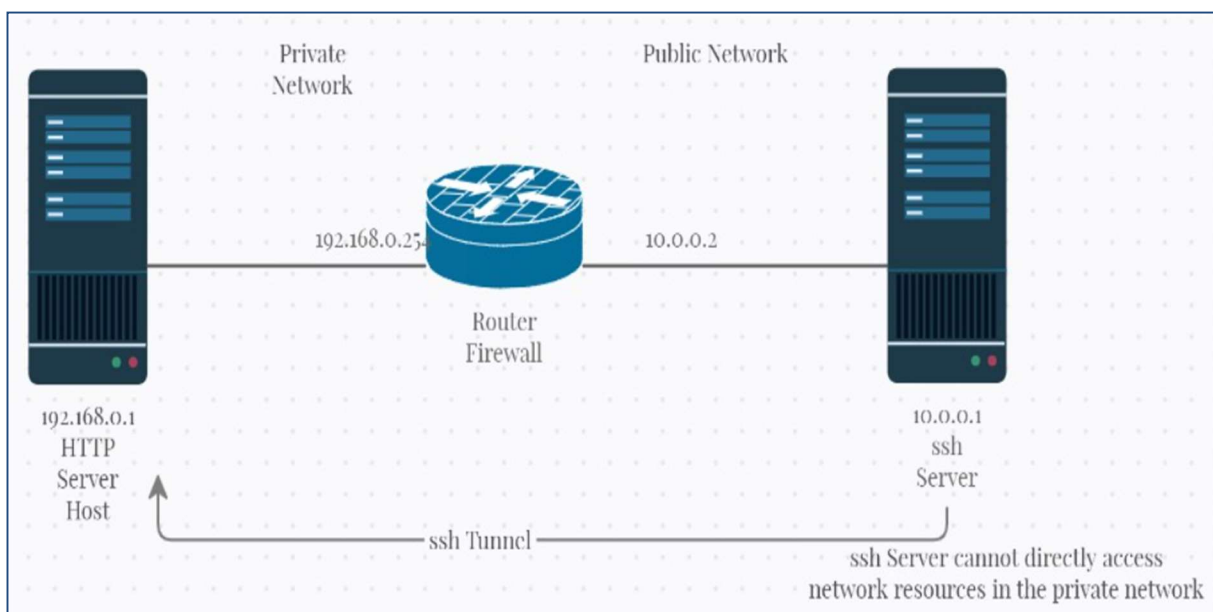


Figure 3: Remote Port Forwarding

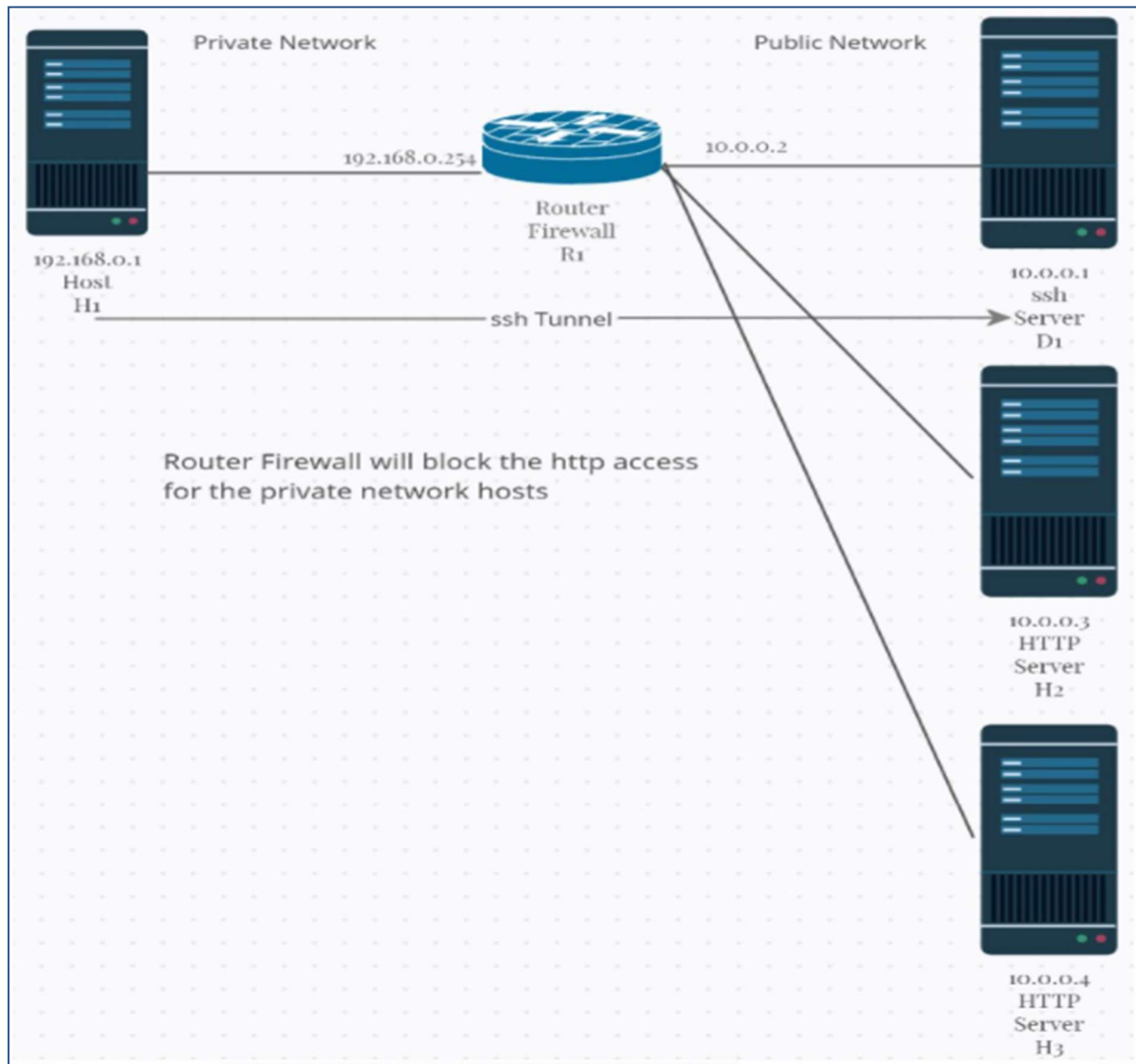


Figure 4: Dynamic Port Forwarding

4. Software & Hardware Specifications

Hardware Requirements:

1. 2GB RAM
2. Hard Disk : 20GB

Software Requirements:

1. Operating System : Ubuntu 20.04
2. Mininet
3. Containernet
4. Docker

5. Implementation Details

Command for local port forwarding:

Local: **-L** Specifies that the given port on the local (client) host is to be forwarded to the given host and port on the remote side.

\$ ssh -L sourcePort:forwardToHost:destPort connectToHost

Code for local port forwarding:

```
#!/usr/bin/python
#from mininet.net import Containernet
from mininet.node import Docker
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Link

def topology():

    "Create a network with some docker containers acting as hosts."
    net = Containernet()

    info('*** Adding hosts\n')
    h1 = net.addHost('h1', ip='192.168.0.1/24')
    d1 = net.addDocker('d1', ip='192.168.0.2/24', dimage="smallko/php-apache-dev:v10")

    info('*** Creating links\n')
    net.addLink(h1, d1)
    info('*** Starting network\n')
    net.start()
    d1.cmd("/etc/init.d/ssh start")
    info('*** Running CLI\n')
    CLI(net)

    info('*** Stopping network')
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    topology()
```

Command for remote port forwarding:

Local: **-R** Specifies that the given port on the remote (server) host is to be forwarded to the given host and port on the remote side.

\$ ssh -R sourcePort:forwardToHost:destPort connectToHost

Code for remote port forwarding:

```
#!/usr/bin/python
from mininet.net import Containernet
from mininet.node import Docker
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Link

"Create a network with some docker containers acting as hosts."
def topology():
    net = Containernet()
    info('*** Adding hosts\n')
    h1 = net.addHost('h1', ip='192.168.0.1/24')
    r1 = net.addHost('r1', ip='192.168.0.254/24')
    d1 = net.addDocker('d1', ip='10.0.0.1/24', dimage="smallko/php-apache-dev:v10")
    info('*** Creating links\n')
    net.addLink(h1, r1)
    net.addLink(r1, d1)
    info('*** Starting network\n')
    net.start()
    d1.cmd("/etc/init.d/ssh start")
    r1.cmd("ifconfig r1-eth1 0")
    r1.cmd("ip addr add 10.0.0.2/24 brd + dev r1-eth1")
    r1.cmd("echo 1 > /proc/sys/net/ipv4/ip_forward")
    r1.cmd("iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o r1-eth1 -j MASQUERADE")
    h1.cmd("ip route add default via 192.168.0.254")
    info('*** Running CLI\n')
    CLI(net)
    info('*** Stopping network')
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    topology()
```

Command for dynamic port forwarding:

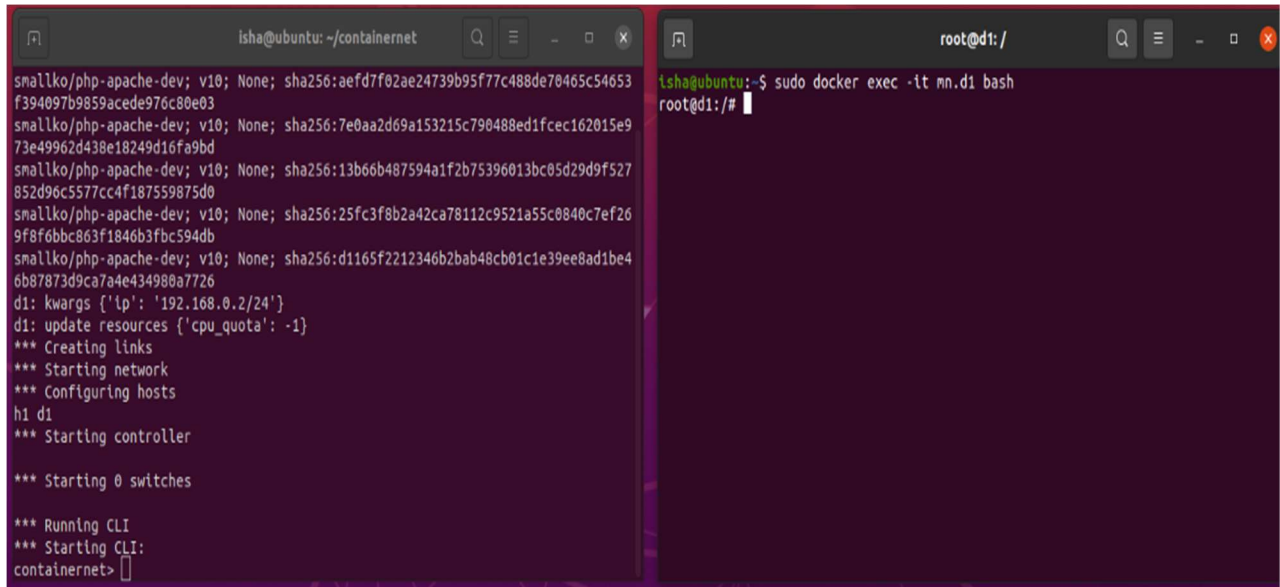
\$ ssh -D 9000 -f -C -q -N connectToHost

- D tells SSH to create a SOCKS tunnel on the the port 9000.
- f forks the process to the background.
- C compresses the data before sending it.
- q enables quiet mode.
- N tells SSH that no command will be sent once the tunnel is up.

Code for dynamic port forwarding:

```
#!/usr/bin/python
from mininet.net import Containernet
from mininet.node import Docker
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Link
def topology(): #Create a network with some docker containers acting as hosts.
    net = Containernet()
    info('*** Adding hosts\n')
    h1 = net.addHost('h1', ip='192.168.0.1/24')
    r1 = net.addHost('r1', ip='192.168.0.254/24')
    d1 = net.addDocker('d1', ip='10.0.0.1/24', dimage="smallko/php-apache-dev:v10")
    h2 = net.addHost('h2', ip='10.0.0.2/24')
    info('*** Creating links\n')
    net.addLink(h1, r1)
    net.addLink(r1, d1)
    net.addLink(r1, h2)
    info('*** Starting network\n')
    net.start()
    d1.cmd("/etc/init.d/ssh start")
    r1.cmd("ifconfig r1-eth1 0")
    r1.cmd("ip addr add 10.0.0.2/24 brd + dev r1-eth1")
    r1.cmd("echo 1 > /proc/sys/net/ipv4/ip_forward")
    r1.cmd("iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o r1-eth1 -j MASQUERADE")
    r1.cmd("iptables -A FORWARD -s 192.168.0.1 -p tcp --dport 80 -j REJECT")
    h1.cmd("ip route add default via 192.168.0.254")
    info('*** Running CLI\n')
    CLI(net)
    info('*** Stopping network')
    net.stop()
if __name__ == '__main__':
    setLogLevel('info')
    topology()
```

6. Screenshots of Implementation



The image shows two terminal windows side-by-side. The left window, titled 'isha@ubuntu: ~/containernet', displays the output of the 'docker exec' command, showing the initialization of a Docker container named 'mn.d1'. The output includes details about the container's configuration, such as its IP address (192.168.0.2/24) and the creation of links and network. The right window, titled 'root@d1: /', shows the prompt for the Docker container, indicating that the container is now running.


```
isha@ubuntu: ~/containernet
smallko/php-apache-dev; v10; None; sha256:aefd7f02ae24739b95f77c488de70465c54653
f394097b9859acade976c80e03
smallko/php-apache-dev; v10; None; sha256:7e0aa2d69a153215c790488ed1fcec162015e9
73e49962d438e18249d16fa9bd
smallko/php-apache-dev; v10; None; sha256:13b66b487594a1f2b75396013bc05d29d9f527
852d96c5577cc4f187559875d0
smallko/php-apache-dev; v10; None; sha256:25fc3f8b2a42ca78112c9521a55c0840c7ef26
9f8f6bb863f1846b3fbc594db
smallko/php-apache-dev; v10; None; sha256:d1165f2212346b2bab48cb01c1e39ee8ad1be4
6b87873d9ca7a4e434980a7726
d1: kwargs {'ip': '192.168.0.2/24'}
d1: update resources {'cpu_quota': -1}
*** Creating links
*** Starting network
*** Configuring hosts
h1 d1
*** Starting controller

*** Starting 0 switches

*** Running CLI
*** Starting CLI:
containernet>
```

```
isha@ubuntu:~$ sudo docker exec -it mn.d1 bash
root@d1:/#
```

Figure 5: Starting the topology and the Docker container for local port forwarding



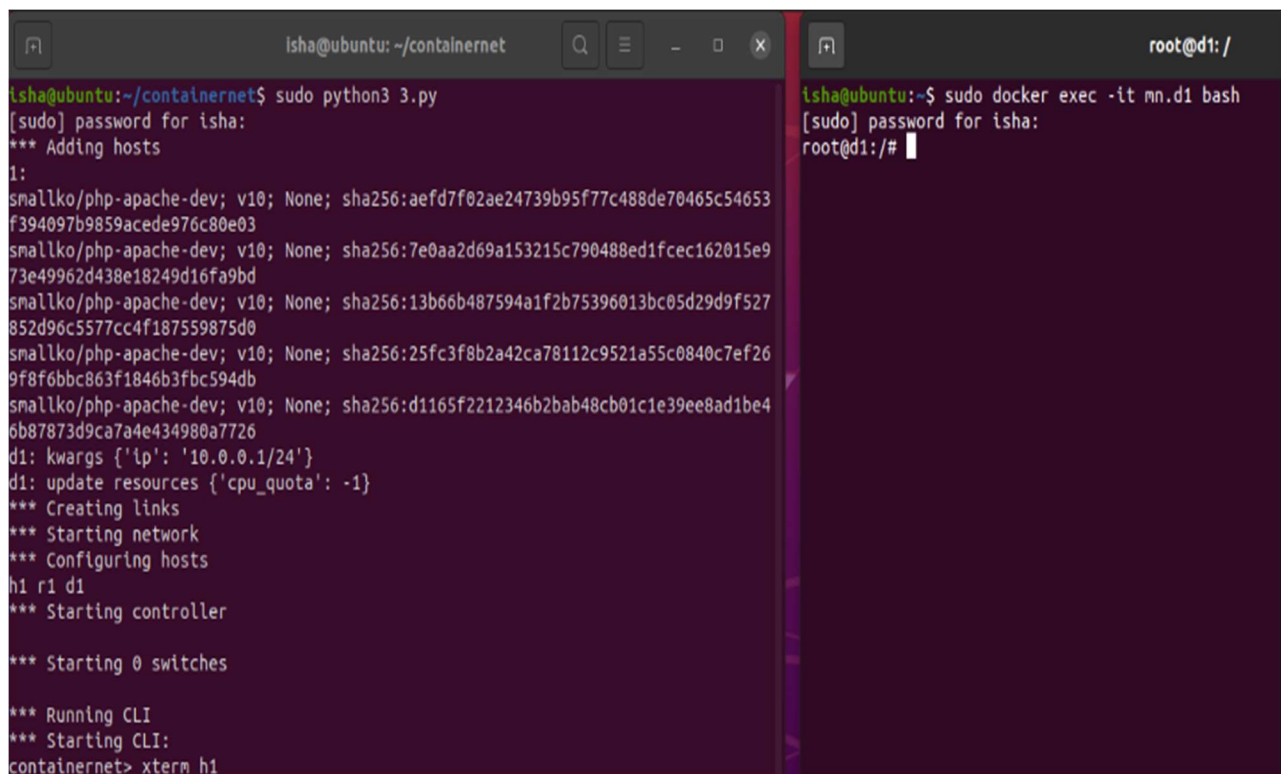
The image shows a terminal window titled '"Node: h1"'. The terminal displays the command 'ssh -Nf -L 5555:192.168.0.2:80 user@192.168.0.2' being executed. The prompt 'user@192.168.0.2's password:' is shown, indicating that the user is prompted to enter their password. The terminal then shows the prompt 'root@ubuntu:/home/isha/containernet#', indicating that the user is back at the host machine.

```
"Node: h1"
root@ubuntu:/home/isha/containernet# ssh -Nf -L 5555:192.168.0.2:80 user@192.168.0
.2
user@192.168.0.2's password:
root@ubuntu:/home/isha/containernet#
```

Figure 6: Starting the local ssh tunnel

```
root@d1: /  
root@d1:/# python -m SimpleHTTPServer 80  
Serving HTTP on 0.0.0.0 port 80 ...  
192.168.0.2 - - [05/May/2021 18:12:00] "GET / HTTP/1.1" 200 -  
^C  
Traceback (most recent call last):  
  File "/usr/lib/python2.7/runpy.py", line 174, in _run_module_as_main  
    "__main__", fname, loader, pkg_name)  
  File "/usr/lib/python2.7/runpy.py", line 72, in _run_code  
    exec code in run_globals  
  File "/usr/lib/python2.7/SimpleHTTPServer.py", line 235, in <module>  
    test()  
  File "/usr/lib/python2.7/SimpleHTTPServer.py", line 231, in test  
    BaseHTTPServer.test(HandlerClass, ServerClass)  
  File "/usr/lib/python2.7/BaseHTTPServer.py", line 610, in test  
    httpd.serve_forever()  
  File "/usr/lib/python2.7/SocketServer.py", line 231, in serve_forever  
    poll_interval)  
  File "/usr/lib/python2.7/SocketServer.py", line 150, in _eintr_retry  
    return func(*args)  
KeyboardInterrupt  
root@d1:/# echo 'hi isha'>isha.htm  
root@d1:/# python -m SimpleHTTPServer 80  
Serving HTTP on 0.0.0.0 port 80 ...  
192.168.0.2 - - [05/May/2021 18:12:30] "GET /isha.htm HTTP/1.1" 200 -  
[  
  
"Node: h1"  
root@ubuntu:/home/isha/containernet# curl 127.0.0.1:5555  
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>  
<title>Directory listing for /</title>  
<body>  
<h2>Directory listing for /</h2>  
<hr>  
<ul>  
<li><a href=".dockerenv">dockerenv</a>  
<li><a href="app/">app/</a>  
<li><a href="bin/">bin/</a>  
<li><a href="boot/">boot/</a>  
<li><a href="dev/">dev/</a>  
<li><a href="docker.stderr">docker.stderr</a>  
<li><a href="docker.stdout">docker.stdout</a>  
<li><a href="entrypoint">entrypoint</a>  
<li><a href="entrypoint.cmd">entrypoint.cmd</a>  
<li><a href="entrypoint.d/">entrypoint.d/</a>  
<li><a href="etc/">etc/</a>  
<li><a href="home/">home/</a>  
<li><a href="isha.htm">isha.htm</a>  
<li><a href="lib/">lib/</a>  
<li><a href="lib64/">lib64/</a>  
<li><a href="media/">media/</a>  
<li><a href="mnt/">mnt/</a>  
<li><a href="opt/">opt/</a>  
<li><a href="proc/">proc/</a>  
<li><a href="root/">root/</a>  
<li><a href="run/">run/</a>  
<li><a href="sbin/">sbin/</a>  
<li><a href="srv/">srv/</a>  
<li><a href="sys/">sys/</a>  
<li><a href="tmp/">tmp/</a>  
<li><a href="usr/">usr/</a>  
<li><a href="var/">var/</a>  
</ul>  
<hr>  
</body>  
</html>  
root@ubuntu:/home/isha/containernet# curl 127.0.0.1:5555/isha.htm  
hi isha  
root@ubuntu:/home/isha/containernet#
```

Figure 7: Starting http server in Docker container and sending http requests on port 5555 from h1



The image shows two terminal windows side-by-side. The left window, titled 'Isha@ubuntu: ~/containernet', shows the execution of 'python3 3.py'. The output lists several 'smallko/php-apache-dev' containers with their IDs and SHA256 hashes, followed by network configuration steps like 'd1: kwargs', 'd1: update resources', 'Creating links', 'Starting network', 'Configuring hosts', 'Starting controller', and 'Starting 0 switches'. It ends with 'Running CLI' and 'Starting CLI: containernet> xterm h1'. The right window, titled 'root@d1: /', shows the command 'sudo docker exec -it mn.d1 bash' being executed, resulting in a shell prompt 'root@d1:/#'.

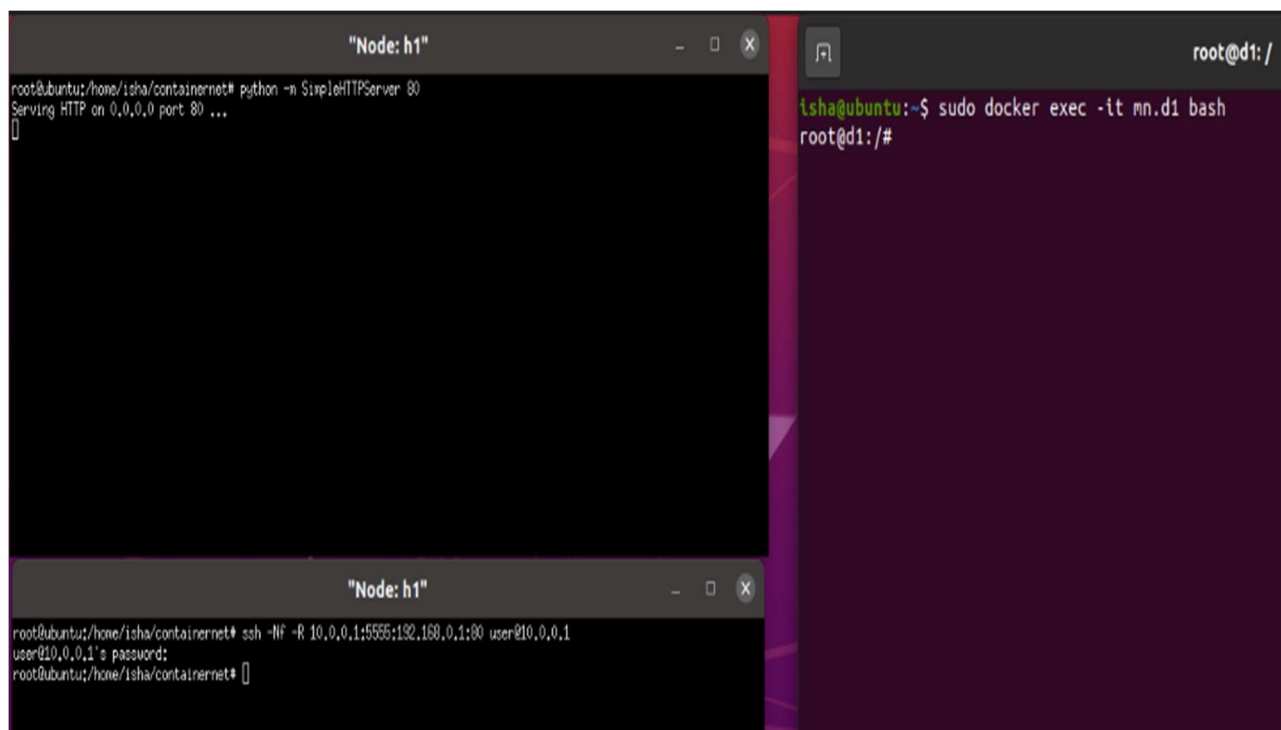
```
Isha@ubuntu: ~/containernet
Isha@ubuntu:~/containernet$ sudo python3 3.py
[sudo] password for isha:
*** Adding hosts
1:
smallko/php-apache-dev; v10; None; sha256:aefd7f02ae24739b95f77c488de70465c54653
f394097b9859acede976c80e03
smallko/php-apache-dev; v10; None; sha256:7e0aa2d69a153215c790488ed1fcec162015e9
73e49962d438e18249d16fa9bd
smallko/php-apache-dev; v10; None; sha256:13b66b487594a1f2b75396013bc05d29d9f527
852d96c5577cc4f187559875d0
smallko/php-apache-dev; v10; None; sha256:25fc3f8b2a42ca78112c9521a55c0840c7ef26
9f8f6bbc863f1846b3fbc594db
smallko/php-apache-dev; v10; None; sha256:d1165f2212346b2bab48cb01c1e39ee8ad1be4
6b87873d9ca7a4e434980a7726
d1: kwargs {'ip': '10.0.0.1/24'}
d1: update resources {'cpu_quota': -1}
*** Creating links
*** Starting network
*** Configuring hosts
h1 r1 d1
*** Starting controller

*** Starting 0 switches

*** Running CLI
*** Starting CLI:
containernet> xterm h1

root@d1: /
Isha@ubuntu:~$ sudo docker exec -it mn.d1 bash
[sudo] password for isha:
root@d1:/#
```

Figure 8:Starting the topology and the Docker container for remote port forwarding



The image shows two terminal windows. The top-left window, titled '"Node: h1"', shows the command 'python -m SimpleHTTPServer 80' being executed, with output 'Serving HTTP on 0.0.0.0 port 80 ...'. The bottom-left window, also titled '"Node: h1"', shows the command 'ssh -Nf -R 10.0.0.1:5555:192.168.0.1:80 user@10.0.0.1' being executed, with output 'user@10.0.0.1's password:'. The right window, titled 'root@d1: /', shows the command 'sudo docker exec -it mn.d1 bash' being executed, resulting in a shell prompt 'root@d1:/#'.

```
"Node: h1"
root@ubuntu:/home/isha/containernet# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...

"Node: h1"
root@ubuntu:/home/isha/containernet# ssh -Nf -R 10.0.0.1:5555:192.168.0.1:80 user@10.0.0.1
user@10.0.0.1's password:
root@ubuntu:/home/isha/containernet#

root@d1: /
Isha@ubuntu:~$ sudo docker exec -it mn.d1 bash
root@d1:/#
```

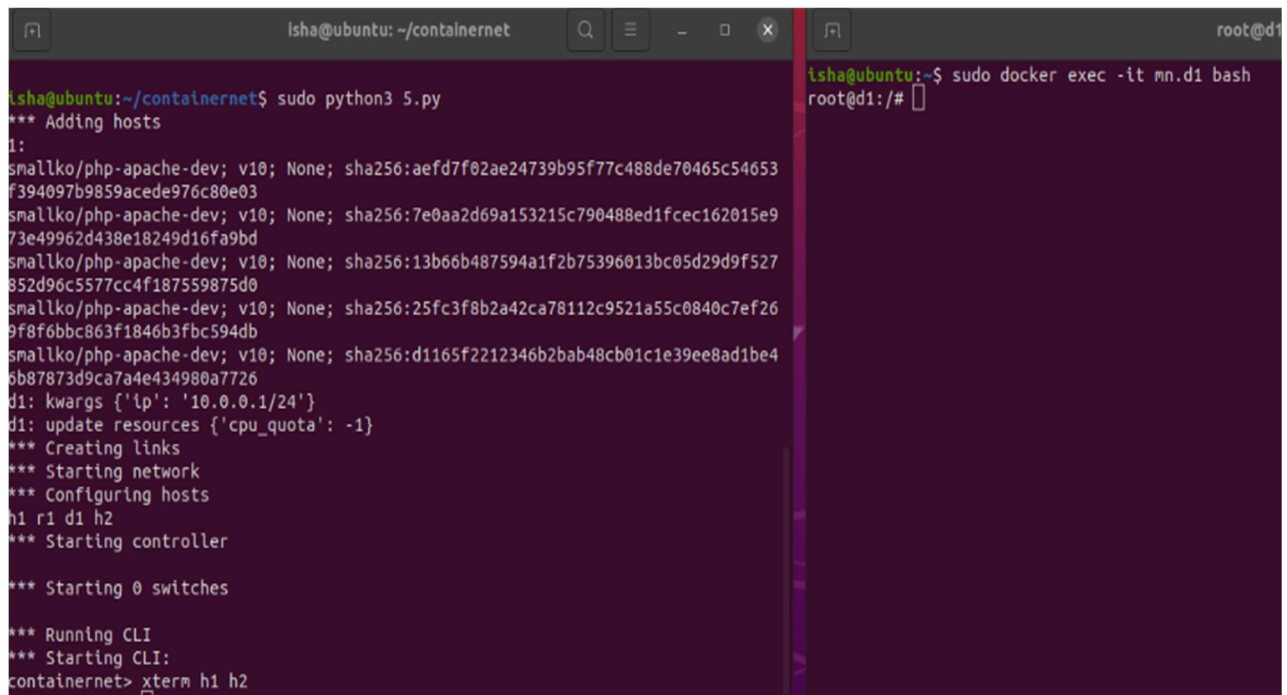
Figure 9:Starting the remote ssh tunnel for the ssh server and http server on h1


```
"Node: h1"

root@ubuntu:/home/isha/containernet# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
192.168.0.1 - - [05/May/2021 11:32:42] "GET / HTTP/1.1" 200 -
192.168.0.1 - - [05/May/2021 11:38:14] "GET /cn.htm HTTP/1.1" 200 -
[]

root@d1: /
<li><a href="1.py">1.py</a>
<li><a href="2.py">2.py</a>
<li><a href="3.py">3.py</a>
<li><a href="5.py">5.py</a>
<li><a href="ansible/">ansible</a>
<li><a href="bin/">bin</a>
<li><a href="build/">build</a>
<li><a href="cn.htm">cn.htm</a>
<li><a href="cn.py">cn.py</a>
<li><a href="CONTRIBUTORS">CONTRIBUTORS</a>
<li><a href="dist/">dist</a>
<li><a href="doc/">doc</a>
<li><a href="Dockerfile">Dockerfile</a>
<li><a href="Dockerfile.centos">Dockerfile.centos</a>
<li><a href="examples/">examples</a>
<li><a href="INSTALL">INSTALL</a>
<li><a href="LICENSE">LICENSE</a>
<li><a href="Makefile">Makefile</a>
<li><a href="mininet/">mininet</a>
<li><a href="mininet.egg-info/">mininet.egg-info</a>
<li><a href="mn.1">mn.1</a>
<li><a href="mnexec">mnexec</a>
<li><a href="mnexec.1">mnexec.1</a>
<li><a href="mnexec.c">mnexec.c</a>
<li><a href="README.md">README.md</a>
<li><a href="setup.py">setup.py</a>
<li><a href="StandaloneVagrantfile">StandaloneVagrantfile</a>
<li><a href="util/">util</a>
<li><a href="Vagrantfile">Vagrantfile</a>
</ul>
<hr>
</body>
</html>
root@d1:/# curl 127.0.0.1:5555/cn.htm
<h1 align="center" style="color:pink;">Welcome to CN2</h1>
root@d1:/#
```

Figure 10: Requesting the cn.htm file in containernet folder from ssh server through remote ssh tunnel on port 5555



The image shows two terminal windows. The left window, titled 'isha@ubuntu: ~/containernet', displays the output of a Python script '5.py'. The script performs several actions: it adds hosts with specific IP addresses and SHA256 hashes, creates links, starts a network, configures hosts, starts a controller, and starts switches. The right window, titled 'root@d1: /', shows the execution of 'sudo docker exec -it mn.d1 bash', which opens a shell inside a Docker container named 'mn.d1'.

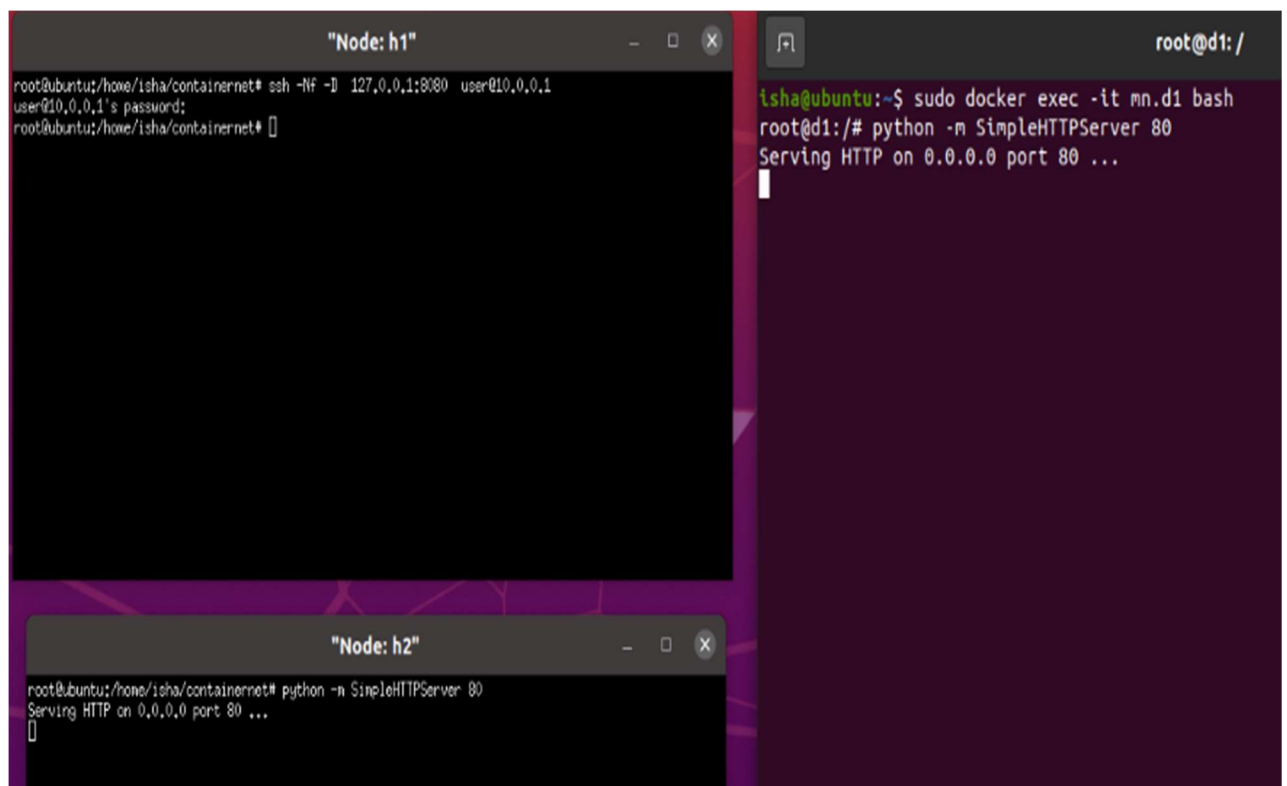
```
isha@ubuntu:~/containernet$ sudo python3 5.py
*** Adding hosts
1:
smallko/php-apache-dev; v10; None; sha256:aefd7f02ae24739b95f77c488de70465c54653
f394097b9859acde976c80e03
smallko/php-apache-dev; v10; None; sha256:7e0aa2d69a153215c790488ed1fcec162015e9
73e49962d438e18249d16fa9bd
smallko/php-apache-dev; v10; None; sha256:13b66b487594a1f2b75396013bc05d29d9f527
852d96c5577cc4f187559875d0
smallko/php-apache-dev; v10; None; sha256:25fc3f8b2a42ca78112c9521a55c0840c7ef26
9f8f6bbc863f1846b3fbc594db
smallko/php-apache-dev; v10; None; sha256:d1165f2212346b2bab48cb01c1e39ee8ad1be4
bb87873d9ca7a4e434980a7726
d1: kwargs {'ip': '10.0.0.1/24'}
d1: update resources {'cpu_quota': -1}
*** Creating links
*** Starting network
*** Configuring hosts
h1 r1 d1 h2
*** Starting controller

*** Starting 0 switches

*** Running CLI
*** Starting CLI:
containernet> xterm h1 h2
```

```
isha@ubuntu:~$ sudo docker exec -it mn.d1 bash
root@d1:/#
```

Figure 11: Starting the topology and the Docker container for dynamic port forwarding



The image shows three terminal windows. The top-left window, titled '"Node: h1"', shows an SSH connection from 'root@ubuntu:/home/isha/containernet' to 'user@10.0.0.1' with the command 'ssh -Nf -D 127.0.0.1:8080 user@10.0.0.1'. The bottom-left window, titled '"Node: h2"', shows the command 'python -m SimpleHTTPServer 80' being executed, resulting in 'Serving HTTP on 0.0.0.0 port 80 ...'. The right window, titled 'root@d1: /', shows the execution of 'sudo docker exec -it mn.d1 bash' followed by 'python -m SimpleHTTPServer 80', which also results in 'Serving HTTP on 0.0.0.0 port 80 ...'.

```
"Node: h1"
root@ubuntu:/home/isha/containernet# ssh -Nf -D 127.0.0.1:8080 user@10.0.0.1
user@10.0.0.1's password:
root@ubuntu:/home/isha/containernet#
```

```
"Node: h2"
root@ubuntu:/home/isha/containernet# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

```
root@d1: /
isha@ubuntu:~$ sudo docker exec -it mn.d1 bash
root@d1:/# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Figure 12: Starting the dynamic ssh tunnel to the ssh server and http server on h2

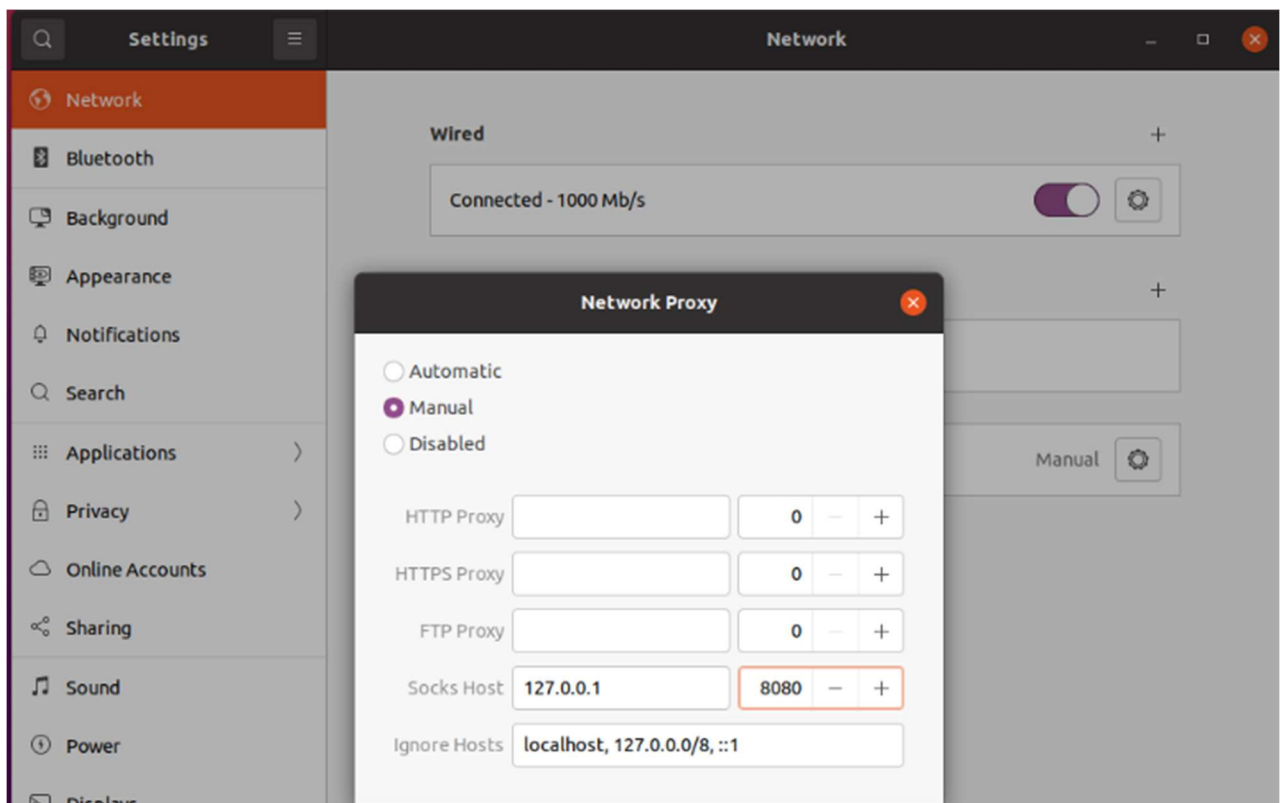


Figure 13: Configuring the Socks Host on port 8080 in proxy settings

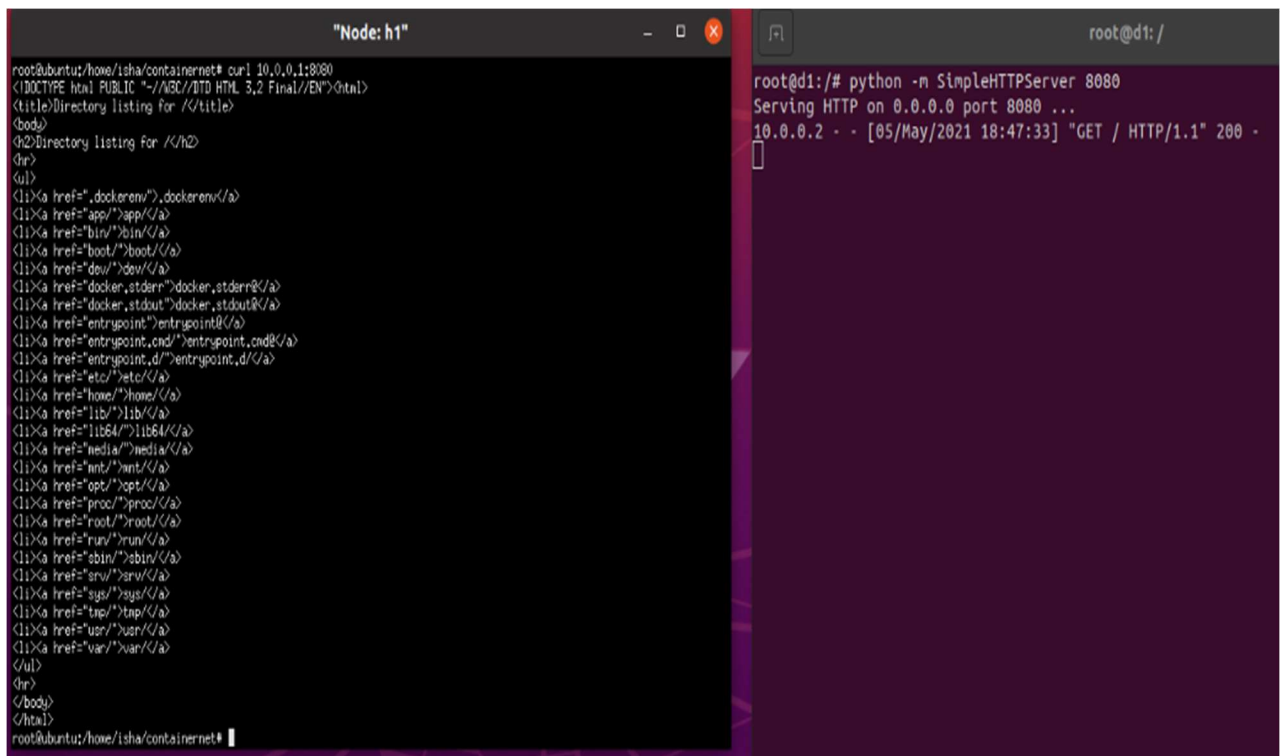


Figure 14: Sending the dynamic http request from h1

7. Result Analysis

1. In Local port forwarding the string sent from http server through echo command was fetched by h1 through SSH tunnel on specified local port 5555.
2. The remote port forwarding is reverse of local port forwarding. SSH server in public network could directly access network resources in the private network through remote SSH tunnel on port 5555.
3. Dynamic port forwarding allows to configure one local port for tunneling data to all remote destinations. However, to utilize this the client application connecting to local port should send their traffic using the SOCKS protocol. At the client side of the tunnel a SOCKS proxy was created and the application (browser) used the SOCKS protocol to specify where the traffic should be sent when it leaves the other end of the SSH tunnel.

8. Conclusion

The created tunnel can be used to transfer all kinds of data not limited to web browsing sessions. We can also tunnel SSH sessions from this as well. Let's assume there is another computer ('banned') to which we need to SSH from within University but the SSH access is being blocked. It is possible to tunnel a SSH session to this host using a local port forward. The setup would look like this.

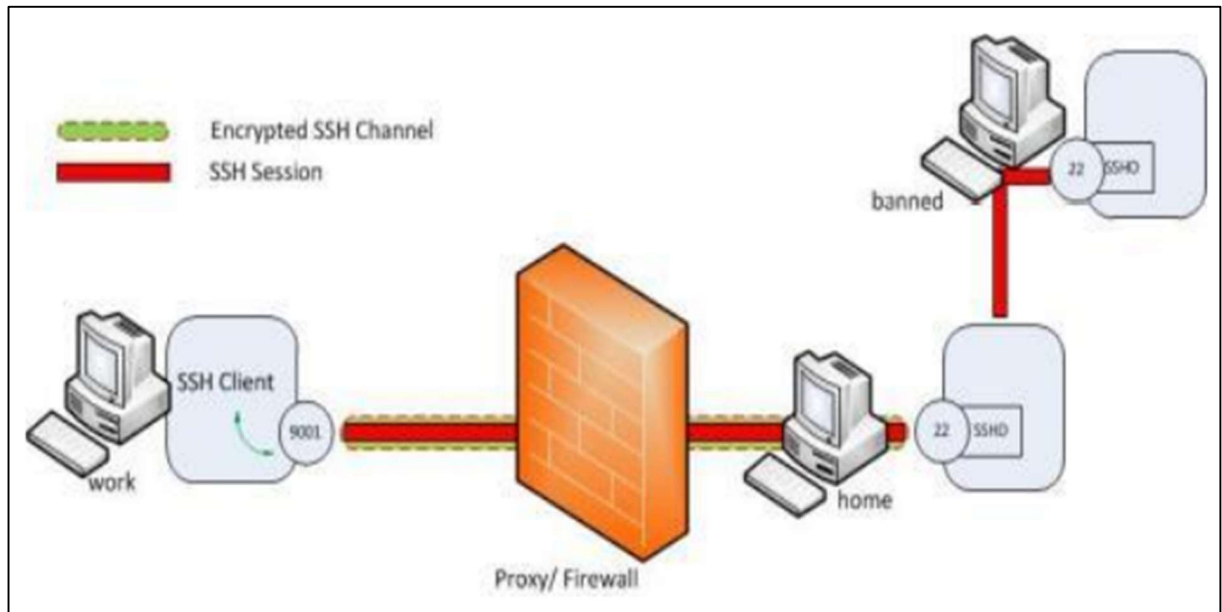


Figure 15: SSH tunneling between banned and work systems.

As can be seen now the transferred data between 'work' and 'banned' are encrypted end to end.

For this we need to create a local port forward as follows.

```
ssh -L 9001:banned:22 home
```

Now we need to create a SSH session to local port 9001 from where the session will get tunneled to 'banned' via 'home' computer.

```
ssh -p 9001 localhost
```

9. References

- [1]. Burande, Aniket, et al. "Wireless Network Security by SSH Tunneling." *Int. J. Sci. Res. Publ* 4.1 (2014): 2250-3153.
- [2]. Orr, Giles, and Jacob Wyatt. "SSH Port Forwarding." *Annual Linux Showcase & Conference*. 2000.
- [3]. Eggendorfer, Tobias, and Daniel Weber. "Running a port forwarding firewall system on a bridge." *Proceedings of the Fourth IASTED International Conference on Communication, Network and Information Security*. 2007.
- [4]. N. (2018, June 27). SSH Tunneling - Local, Remote & Dynamic. DEV Community. https://dev.to/__namc/ssh-tunneling---local-remote--dynamic-34fa
- [5]. SSH Tunnel - Local and Remote Port Forwarding Explained With Examples - Trackets Blog. (2014, May 17). SSH Tunnel. https://blog.trackets.com/2014/05/17/ssh-tunnel-local-and-remote-port-forwarding-explained-with-examples.html?utm_source=cronweekly.com
- [6]. SSH Tunneling Explained. (2012, November 11). Source Open. <https://chamibuddhika.wordpress.com/2012/03/21/ssh-tunnelling-explained/>
- [7]. Wikipedia contributors. (2021, March 10). Tunneling protocol. Wikipedia. https://en.wikipedia.org/wiki/Tunneling_protocol
- [8]. How to Set up SSH Tunneling (Port Forwarding). (2021, March 10). Linuxize. <https://linuxize.com/post/how-to-setup-ssh-tunneling/>
- [9]. Hoffman, C. (2017, July 12). How to Use SSH Tunneling to Access Restricted Servers and Browse Securely. How-To Geek. <https://www.howtogeek.com/168145/how-to-use-ssh-tunneling/>

10. Video Link

https://drive.google.com/file/d/1h-P3p9EvZ50_pmq49dn2PiEgUYefi2ri/view?usp=sharing