Visual guide to SSH tunneling and port forwarding

CaffeineFueled

2023/01/01

To make it quick, I wish I had known about port forwarding and tunneling earlier. With this blog post, I try to understand it better myself and share some experiences and tips with you.

Topics: use cases, configuration, SSH jumphosts, local/remote/dynamic port forwarding, and limitations

Use cases

SSH tunneling and port forwarding can be used to forward TCP traffic over a secure SSH connection from the SSH client to the SSH server, or vice versa. TCP ports or UNIX sockets can be used, but in this post I'll focus on TCP ports only.

I won't go into details, but the following post should show enough examples and options to find use in your day-to-day work.

Security:

encrypt insecure connections (FTP, other legacy protocols)

access web admin panels via secure SSH tunnel (Pub Key Authentication)

having potentially less ports exposed (only 22, instead of additional 80/443)

Troubleshooting:

bypassing firewalls/content filters

choosing different routes

Connection:

reach server behind NAT

use jumphost to reach internal servers over the internet

exposing local ports to the internet

There are many more use cases, but this overview should give you a sense of possibilities.

Port forwarding

Before we start: the options of the following examples and be combined and configured to suit your setup. As a side note: if the `bind_address` isn't set, localhost will be the default

Configuration / Preparation

- The **local and remote users must have the necessary permissions** on the local and remote machines respectivly to open ports. **Ports between 0-1024 require root privileges** - if not configured differently - and the rest of the ports can be configured by standard users.
- **configure clients and network firewalls accordingly**

SSH port forwarding must be enabled on the server:

```
AllowTcpForwarding yes
```

It is enabled by default, if I recall it correctly

If you forward ports on interfaces other than 127.0.01, then you'll need to enable `GatewayPorts` on the SSH server:

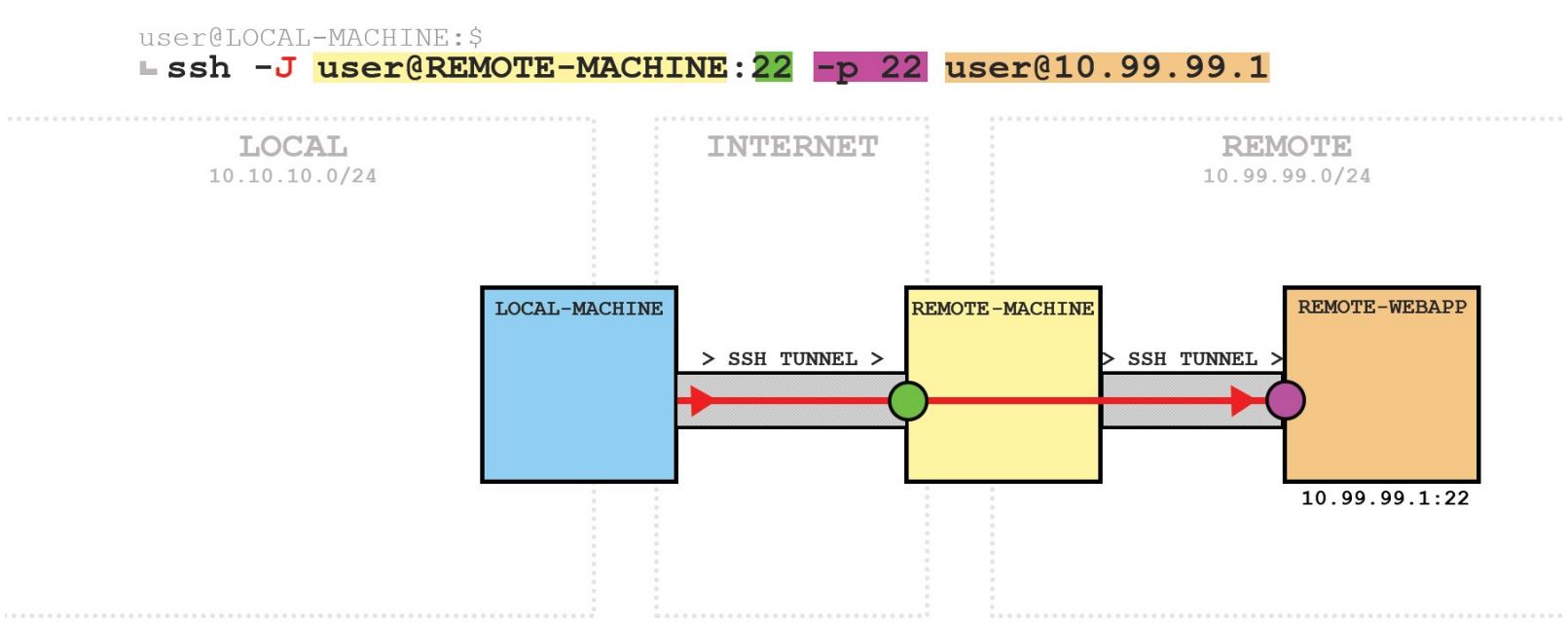
```
GatewayPorts yes
```

Remember to **restart the ssh server service**.

SSH jumphost / SSH tunnel

Transparently connecting to a remote host through one or more hosts.

```
ssh -J user@REMOTE-MACHINE:22 -p 22 user@10.99.99.1
```



Side note: The port addressing can be removed, if the default port 22 is used!

On REMOTE-MACHINE as jumphost:

```
[user@REMOTE-MACHINE]$ ss | grep -i ssh
tcp   ESTAB 0      0                                                  167.135.173.108:ssh    192.1
tcp   ESTAB 0      0                                                  10.99.99.2:49770       10.99.
```

Explanation:

`167.135.173.108` - public IP of REMOTE-MACHINE

`92.160.120.207` - public IP of LOCAL-MACHINE

`10.99.99.2` - internal IP of REMOTE-MACHINE

`10.99.99.1` - internal IP of REMOTE-WEBAPP

Using multiple jumphosts

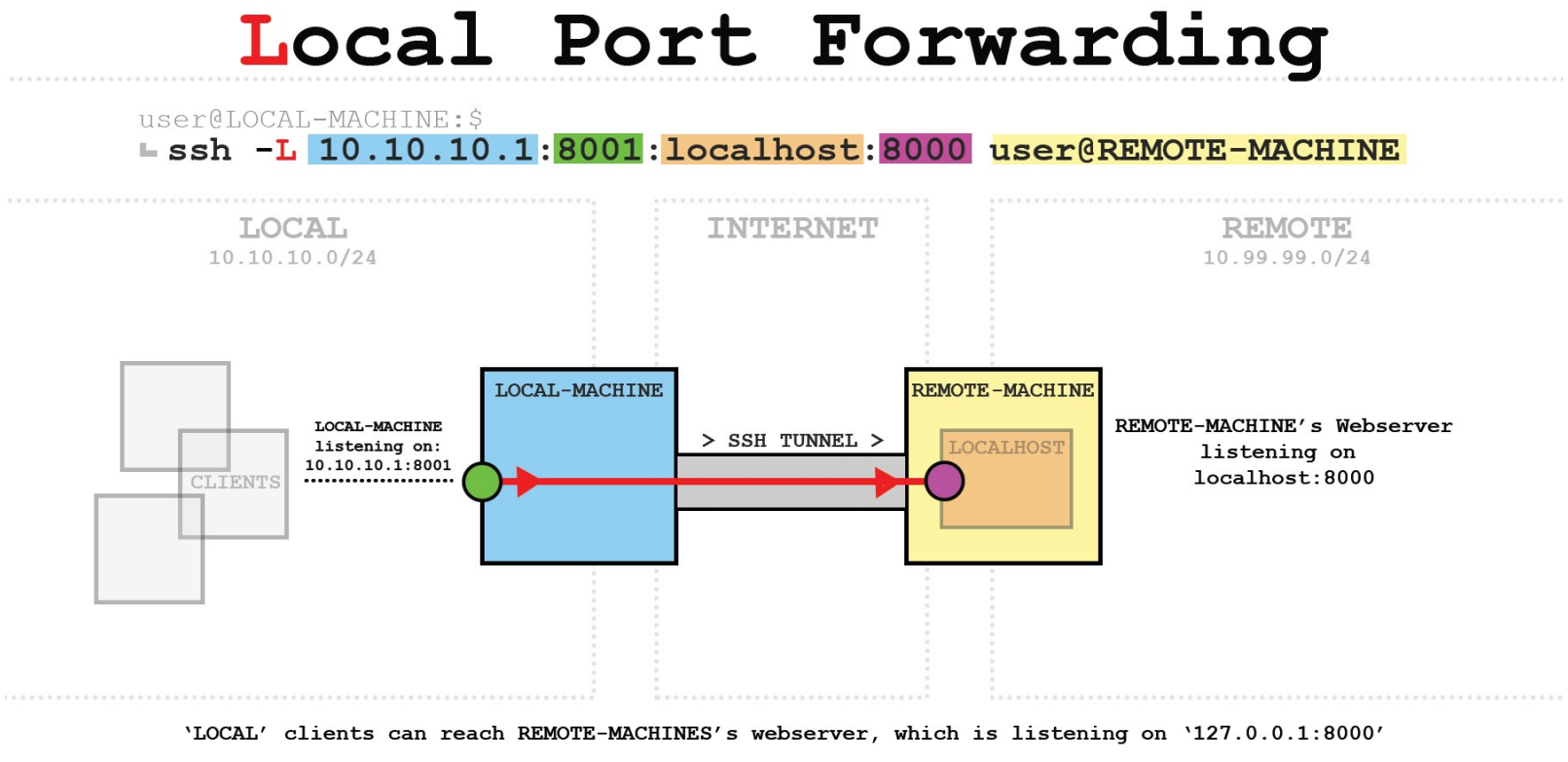
Jumphosts must be separated by commas:

```
ssh -J user@REMOTE-MACHINE:22,user@ANOTHER-REMOTE-MACHINE:22 -p 22 user@10.99.99.1
```

Local Port Forwarding

Example 1

```
ssh -L 10.10.10.1:8001:localhost:8000 user@REMOTE-MACHINE
```



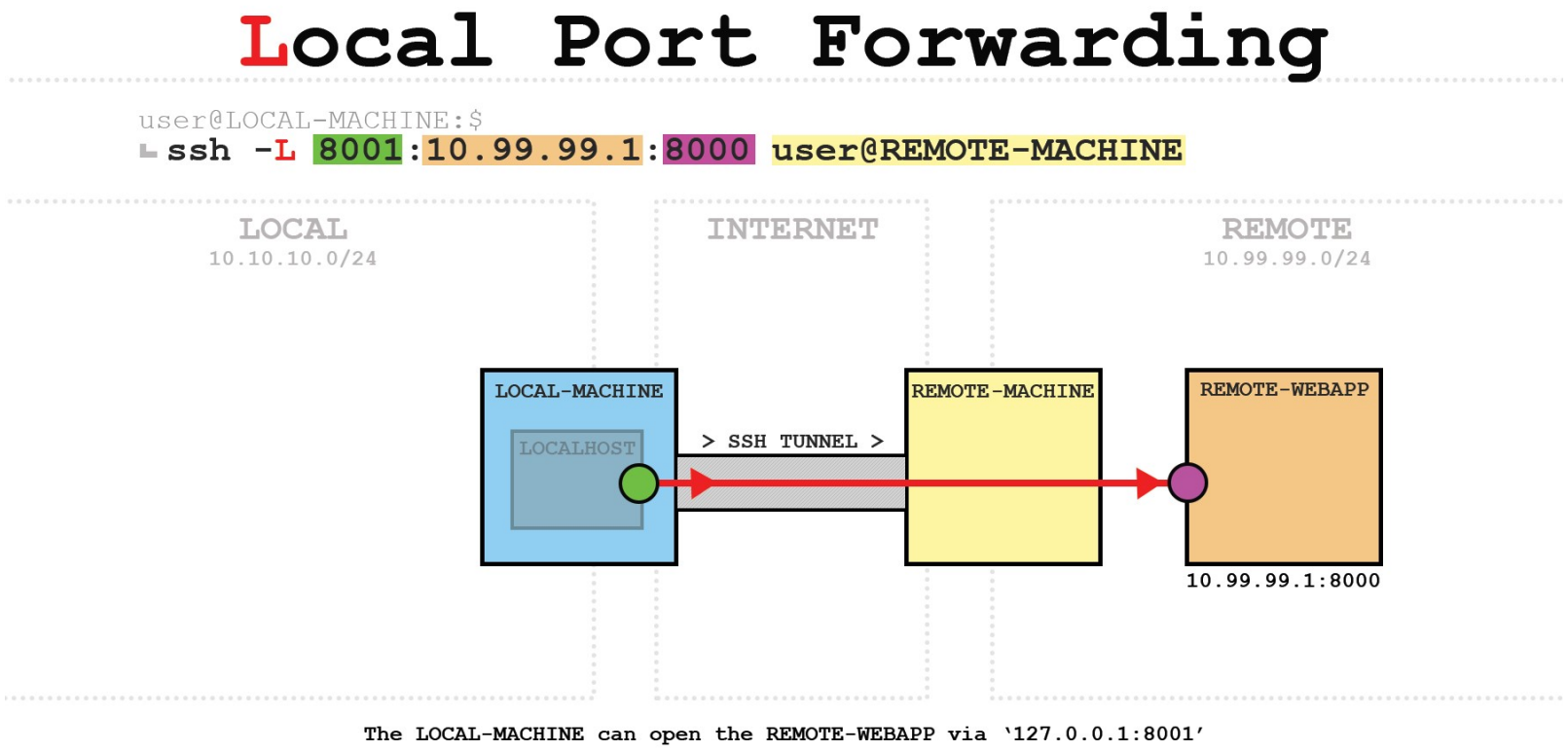
Access logs of the webserver on REMOTE-MACHINE that only listens on 127.0.0.1:

```
127.0.0.1 - - [30/Dec/2022 18:05:15] "GET / HTTP/1.1" 200
```

the request originates from LOCAL-MACHINE

Example 2

```
ssh -L 8001:10.99.99.1:8000 user@REMOTE-MACHINE
```



Access logs of the webserver on REMOTE-WEBAPP:

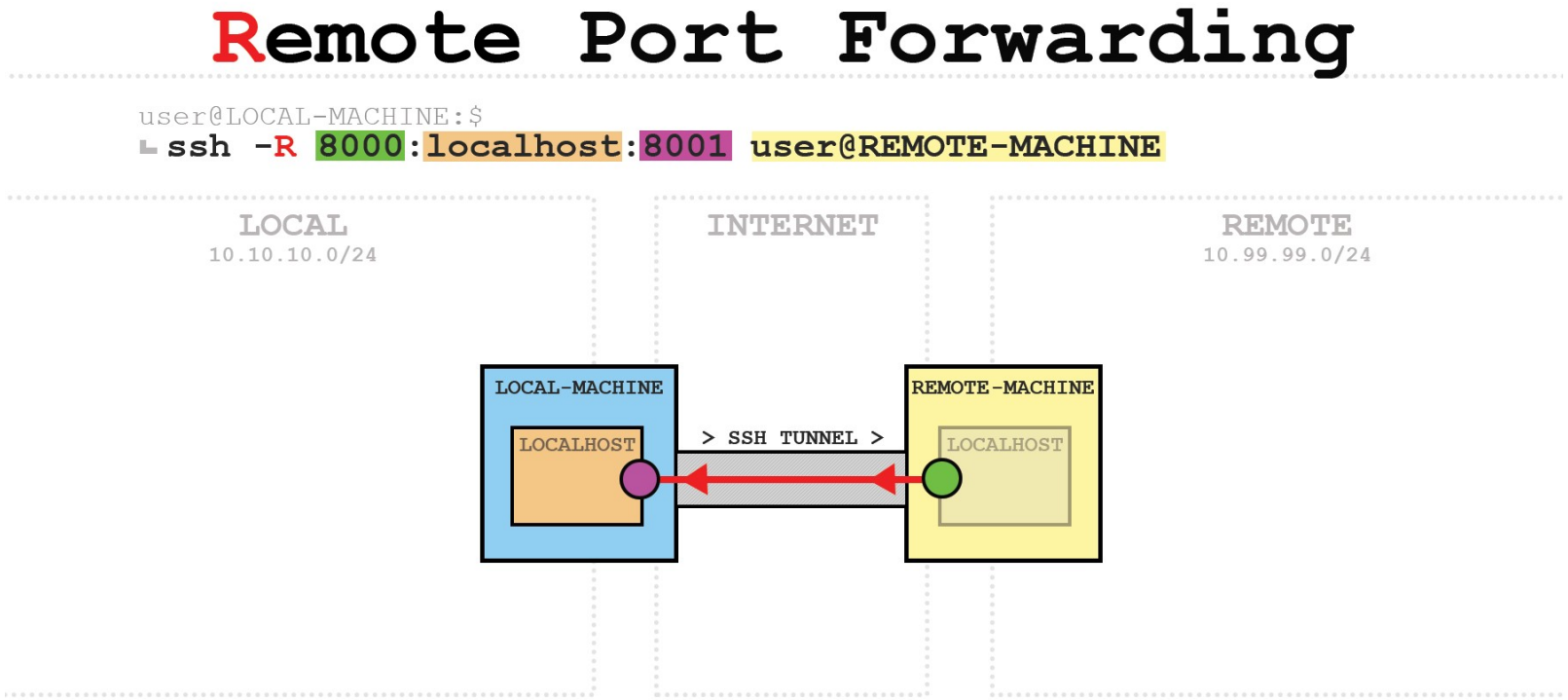
```
10.99.99.2 - - [30/Dec/2022 21:28:42] "GET / HTTP/1.1" 200
```

the request originates from the intern IP of LOCAL-MACHINE (10.99.99.2)

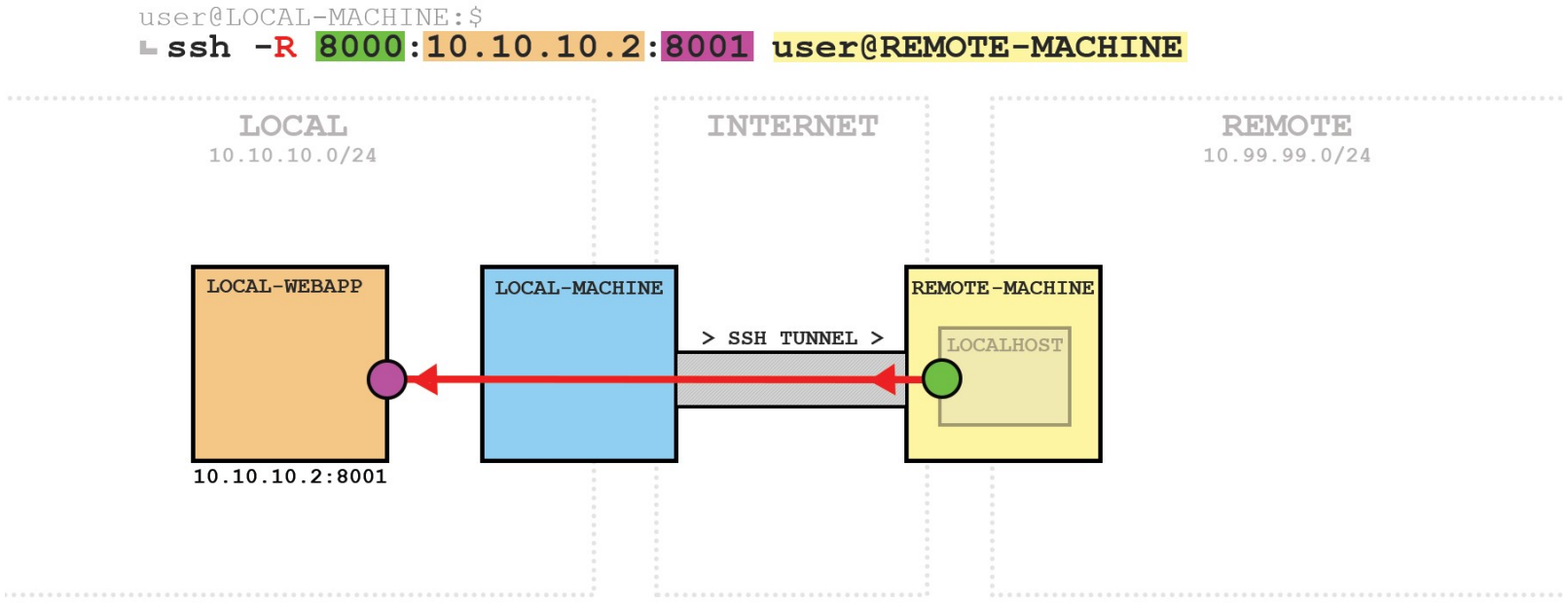
Remote Port Forwarding

Example 1+2

```
ssh -R 8000:localhost:8001 user@REMOTE-MACHINE
```

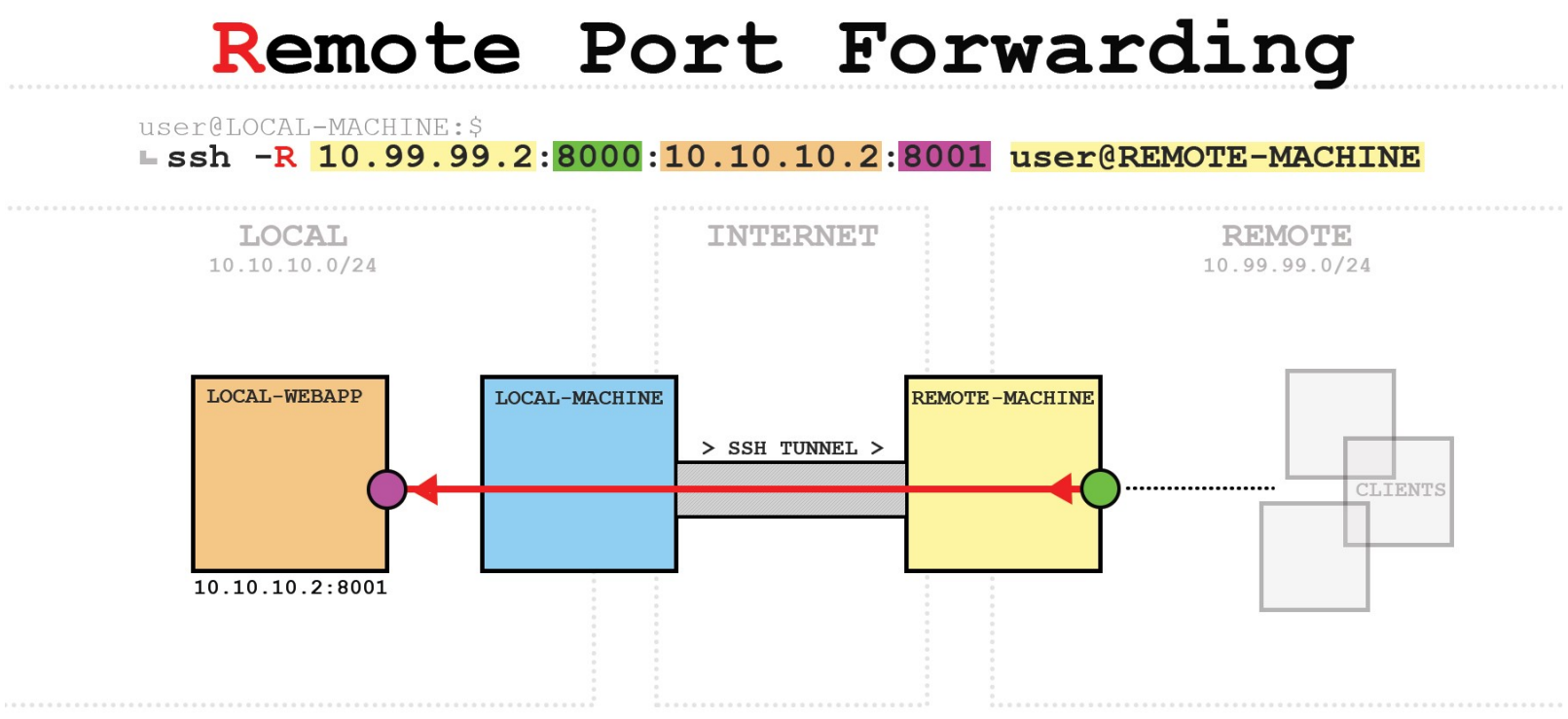



```
ssh -R 8000:10.10.10.2:8001 user@REMOTE-MACHINE
```



Example 3

```
ssh -R 10.99.99.2:8000:10.10.10.2:8001 user@REMOTE-MACHINE
```



Important: `GatewayPorts yes` must be enabled on the SSH server to listen on another interface than the loopback interface.

Dynamic port forwarding

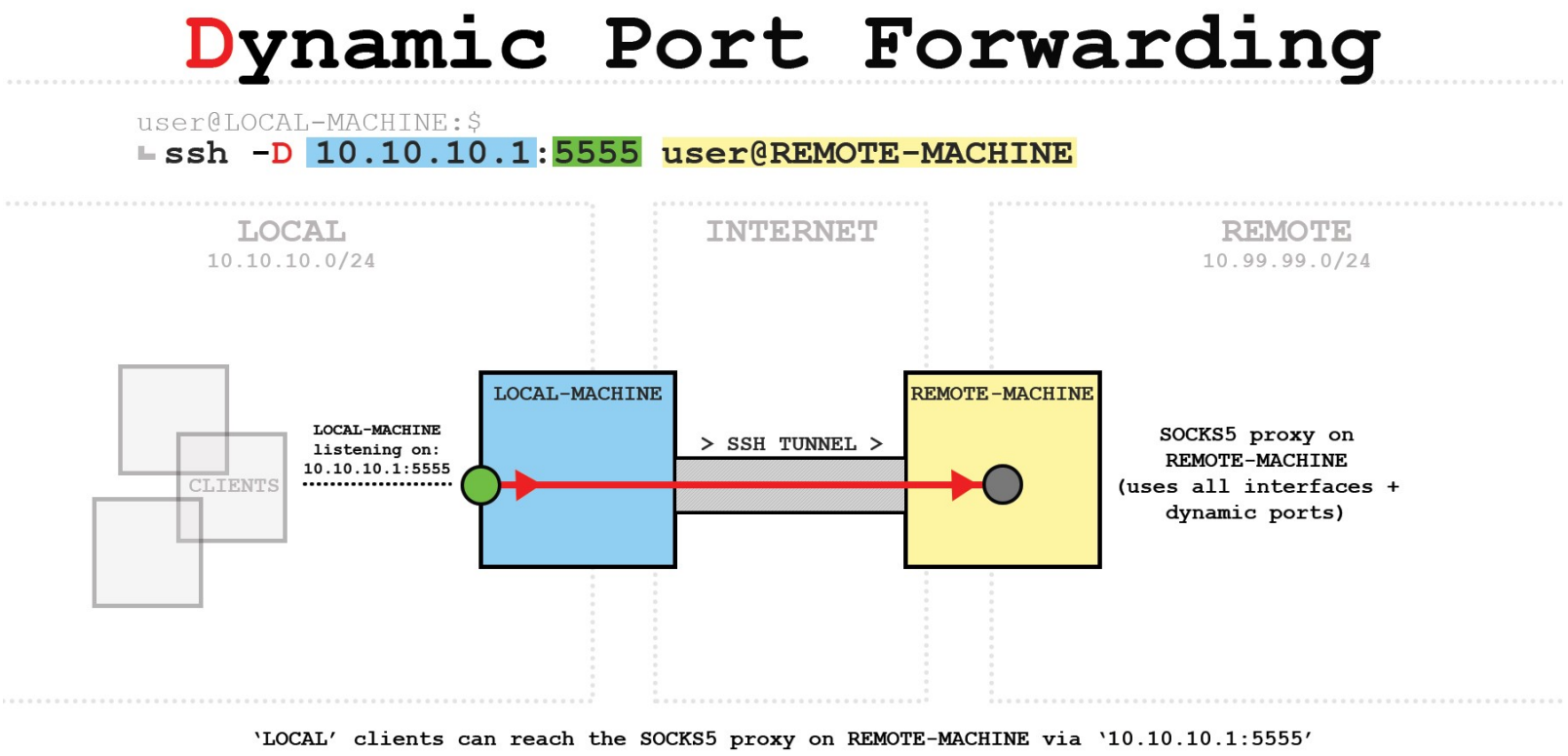
To forward more than one port, SSH uses the SOCKS protocol. This is a transparent proxy protocol and SSH makes us of the most recent version SOCKS5.

Default port for SOCKS5 server is 1080 as defined in RFC 1928.

The client must be configured correctly to use a SOCKS proxy. Either on the application or OS layer.

Example

```
ssh -D 10.10.10.1:5555 user@REMOTE-MACHINE
```



Use `curl` on a 'LOCAL' client to test the correct connection/path:

```
curl -L -x socks5://10.10.10.1:5555 brrl.net/ip
```

If everything works out, you should get the public IP of the REMOTE-MACHINE back

SSH TUN/TAP tunneling

I won't go into detail, but you can create a bi-directional TCP tunnel with the `-w` flag. The interfaces must be created beforehand, and I haven't tested it yet.

```
-w local_tun[:remote_tun]
```

How to run SSH in the background

The native way to run the tunnel in the background would be `-fN`:

`-f` - run in the background

`-N` - no shell

```
ssh -fN -L 8001:127.0.0.1:8000 user@REMOTE-MACHINE
```

Others than that: use screen or some other tools.

Stop the SSH running in the background

```
user@pleasejustwork:~$ ps -ef | grep ssh
[...]
user      19255       1  0 11:40 ?        00:00:00 ssh -fN -L 8001:127.0.0.1:8000 user@REMOTE
[...]
```

Kill the process with the PID:

```
kill 19255
```

Keep SSH connection alive

I won't go into detail, but there are different ways to keep the SSH connection alive.

Handle timeouts with heartbeats

Both options can be set on the client or server, or both.

`ClientAliveInterval` will send a request every `n` seconds to keep the connection alive:

```
ClientAliveInterval 15
```

`ClientAliveCountMax` is the number of heartbeat requests sent after not receiving an respond from the other side of the connection before terminating the connection:

```
ClientAliveCountMax 3
```

`3` is the default, and setting it to `0` will disable connection termination. In this example, the connection would drop after around 45 seconds without any responds.

Reconnecting after termination

There are mutliple ways to do it; autossh, scripts, cronjobs, and so on.

This is beyond this post and I might write about in the future.

Limitations

UDP

SSH depends on a reliable delivery to be able to decrypt everything correctly. UDP does not offer any reliability and is therefore not supported and recommended to use over the SSH tunnel.

That said, there are ways to do it as described in this post. I still need to test it.

TCP-over-TCP

It lowers the throughput due to more overhead and increases the latency. On connections with packet loss or high latencies (e.x. satellite) it can cause a TCP meltdown.

This post is a great write-up.

Nevertheless, I'd been using OpenVPN-over-TCP for a while, and it worked flawlessly. Less throughput than UDP, but reliable. So, it highly depends on your setup.

Not a VPN replacement

Overall, it is not a VPN replacement. SSH tunneling can be used as such, but a VPN is better suited for better performance.

Potential security risk

If you do not need those features, it is recommended to turn them of. Threat actors could use said features to avoid firewalls and other security measures.

General links:

SSH manual

sshd_config manual

The inspiration of this blog post are the following unix.stackexchange answer and blog post of Dirk Loss.

Thanks to Frank and ruffy for valuable feedback!

---

7 Comments

Name (optional) John Doe

E-mail (optional) johndoe@example.com

Preview Submit

Diggles • 4 days ago

This is a great article thanks. There is a typo where "1" is missing: 92.160.120.207 - public IP of LOCAL-MACHINE.
Just letting you know so that new learners don't get confused by this.

Reply

NTOSLinux • 4 days ago

Which tool is used to draw those beautiful illustrations?

Reply

CF • 3 days ago

Photoshop

Reply

maoif • 4 days ago

Very nice and clean illustrations, thanks!

Reply

Dirk Loss • 4 days ago

Excellent! Thank you for improving upon my diagrams.

Reply

OwenChia • 3 days ago

you can also use `ssh -R 9050 user@server` to achieve same function like `ssh -D` but reverse.

Reply

Anonymous • 2 days ago

You may use "GatewayPorts clientspecified" instead of "GatewayPorts yes". If the GatewayPorts setting is set to "yes", the specified remote bind address (10.99.99.2) will be ignored and "*" will be used.

Reply

Most recent Articles:

- Dummy IP & MAC Addresses for Documentation & Sanitization
- Deploying ISSO Commenting System for Static Content using Docker
- Generate a Vanity v3 Hidden Service Onion Address with mkp224o
- ssh-audit Primer - Audit your SSH Server
- mtr - More Detailed Traceroute - Network Troubleshooting