Going global with localhost

Talk by Maciej Pędzich

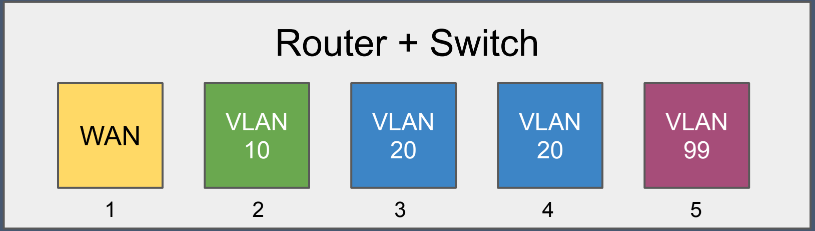
About me

Demo

My network’s diagram

Virtual Local Area Networks

- ➔ Physically uniform network is split into logical segments
- ➔ Ports 2-5 are assigned a VLAN ID
- ➔ Outgoing packets get assigned a tag with their source VLAN's ID
- ➔ If a packet is allowed through, its VLAN ID tag gets removed
- ➔ Appropriate firewall rules ensure devices on different VLANs are separated…
- ➔ … but they may allow a service living on one VLAN to be accessed from other VLANs



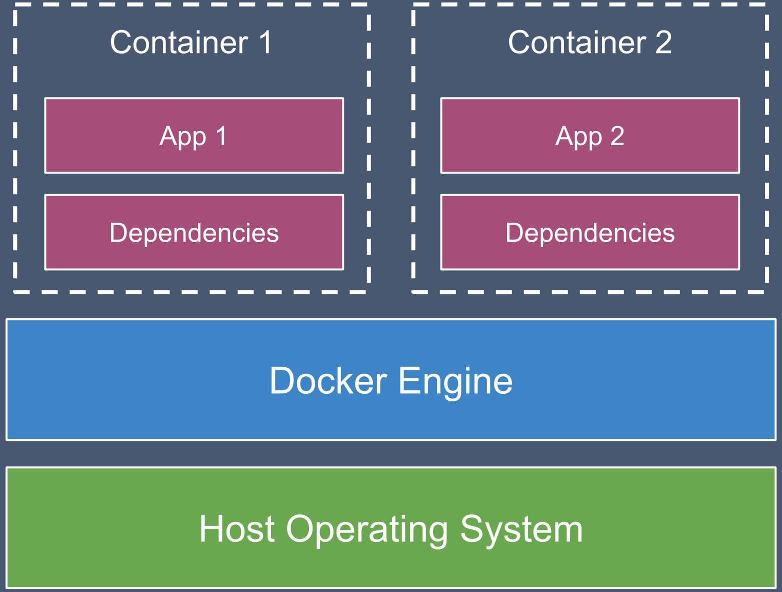
- ➔ VLAN 10 (Homelab) - 10.0.10.1/24
- ➔ VLAN 20 (Guest) - 10.0.20.1/24
- ➔ VLAN 99 (Admin) - 192.168.99.1/24

CIDR notation by example

- **Task:** convert 10.0.10.1/24 notation to *first IP - last IP*
- **Step 1:** take the part before the slash - that's the *first IP*
- **Step 2:** convert each decimal number in that address to 8-bit binary and then write *number after slash* 1s underneath (starting from the left-hand side)
  ```
  00001010.00000000.00001010.00000001
  11111111.11111111.11111111.00000000
  ```
- **Step 3:** to get the last address, write down a digit from the top if there's a 1 underneath it, and write 1 if there's a 0 underneath
  ```
  00001010.00000000.00001010.11111111
  ```
- **Step 4:** convert the result back to decimal - 10.0.10.255
- **Result:** 10.0.10.1 - 10.0.10.255

Docker containers

- ➔ **Why:** we want to mitigate dependency hell and environment variable conflicts when running multiple apps on the same machine
- ➔ **How:** each application’s configuration, assets, required libraries, and so on get packaged up into an image that can be used to create an isolated sandbox for the app to run in



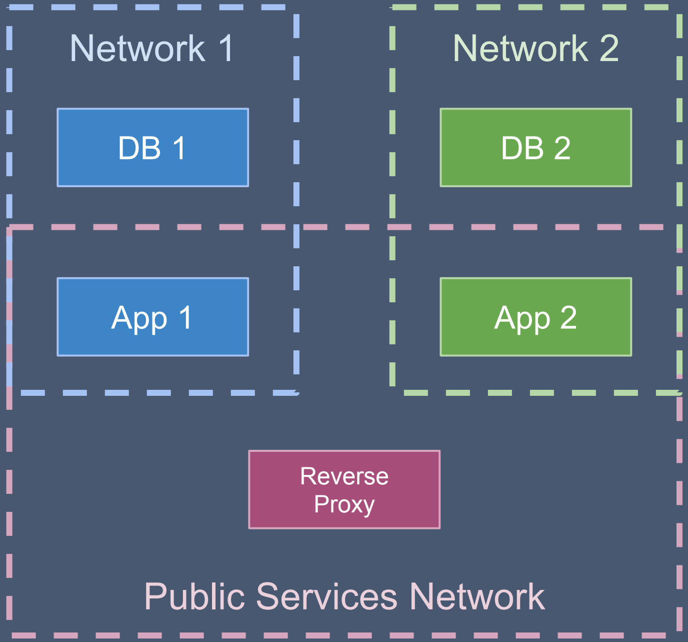
Showcase of my own PaaS

with dark theme and webhooks

Docker networks

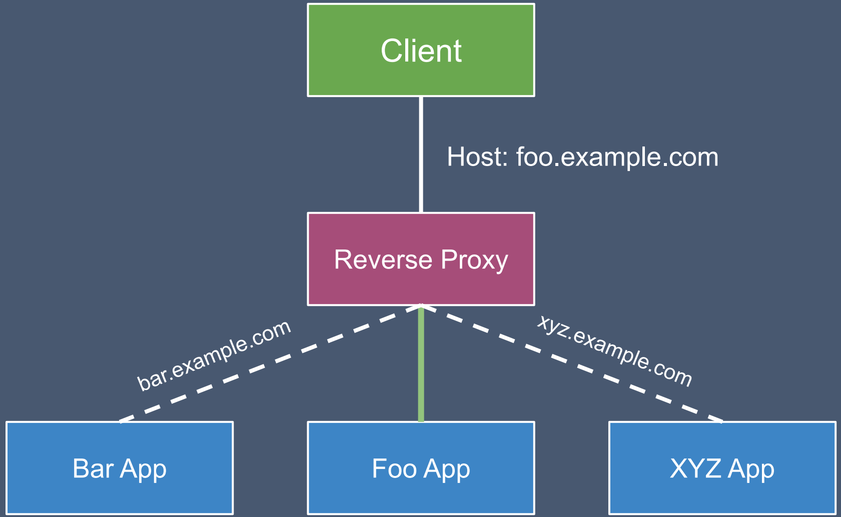
➔ **Why:** we want certain containers to be able to talk to each other using the network while maintaining isolation between them and the host

➔ **How:** Docker creates a software bridge, which allows containers on the same network to communicate, but also adds firewall rules on the host to ensure separation



Reverse Proxy

- ➔ **Why:** we want to access our web apps via memorable domain names without having to expose and remember unique ports on the host for each service
- ➔ **How:** reverse proxy behaves like a middleman between the client and web servers, forwarding appropriate requests to the right services (but also handling TLS encryption, access control, and more)



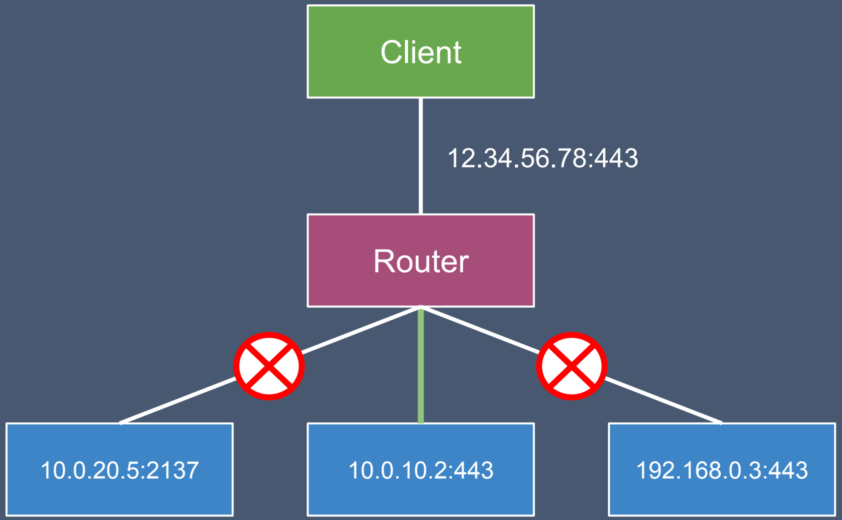
Dynamic DNS

- ➔ **Why:** chances are your router's got a dynamic IP address, meaning you have to update your domain name's records every time it changes
- ➔ **How:** employ a DDNS service that will periodically update DNS records for a free domain name, or create your own script if your name server provider allows you to do that via a Web API

It's not DNS
There's no way it's DNS
It was DNS

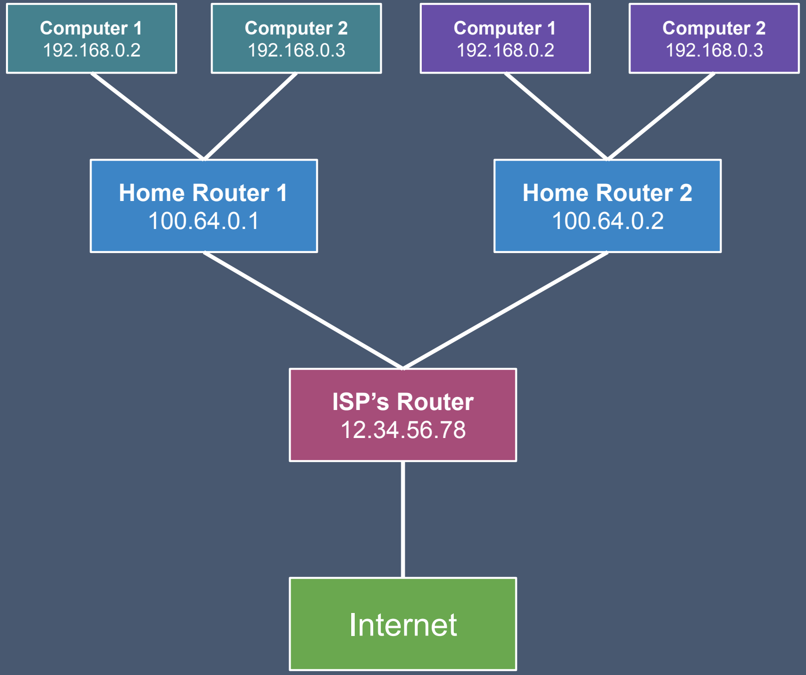
Port forwarding

- ➔ **Why:** while your router can keep track of outgoing connections and the local IPs to which appropriate response packets should be sent, it can't automatically do the same for incoming connections
- ➔ **How:** it's necessary to add a special rule in your router that will tell it to forward all incoming TCP packets on ports 80 and 443 to the same ports on your server's local IP address



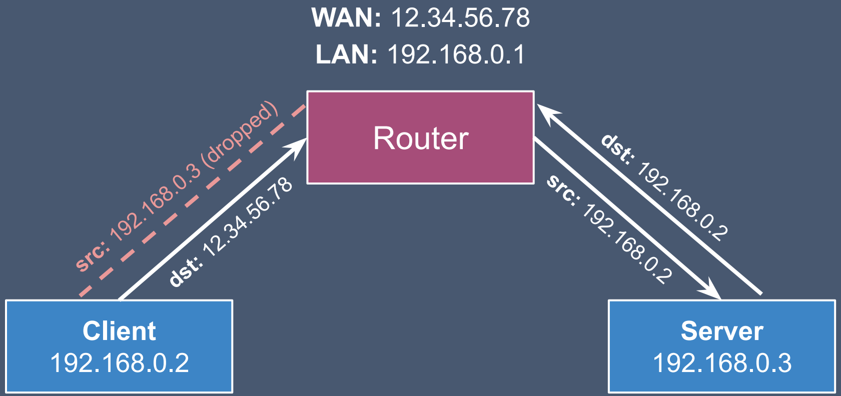
Carrier-grade NATs

- **Problem:** even if you forward the right ports on your home router, they might get blocked by your ISP's router, as the former is effectively sitting behind a NAT managed by your carrier - hence the name



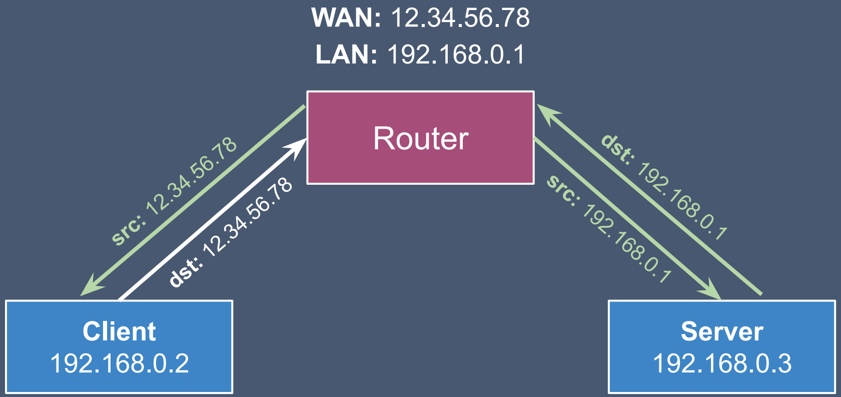
Connecting to a public server via LAN

➔ **Problem:** right now, any attempt to connect to a public server using any device on the local network will fail, because that device will be expecting a packet from the router's public IP, but instead it will receive an unexpected packet from the server's local IP



Solution 1: NAT hairpinning

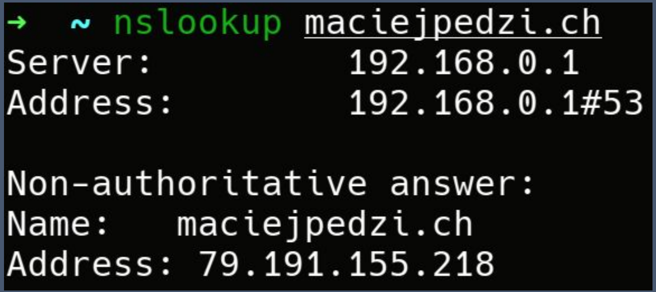
- ➔ Some routers may allow you to mask the local device's IP behind the one used by the router and have it track the connection to ensure that the server's local IP will later get translated back to the router's public IP address



Solution 2: Split-horizon DNS

- ➔ Alternatively, you can set up an internal DNS server (although your router should provide one) and add records that will have the “naked” domain and all subdomains point to the server’s local IP address
- ➔ This is generally the recommended solution, because the router doesn’t have to hide the requester’s IP, as it’s in the same network with the server

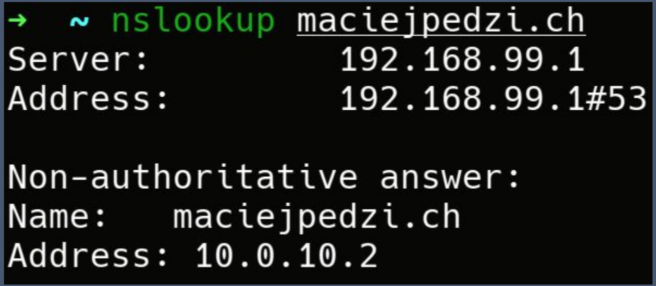
External DNS lookup



```
➜  ~ nslookup maciejpedzi.ch
Server:         192.168.0.1
Address:        192.168.0.1#53

Non-authoritative answer:
Name:   maciejpedzi.ch
Address: 79.191.155.218
```

Internal DNS lookup



```
➜  ~ nslookup maciejpedzi.ch
Server:         192.168.99.1
Address:        192.168.99.1#53

Non-authoritative answer:
Name:   maciejpedzi.ch
Address: 10.0.10.2
```

Pros of self-hosting

- ➔ Freedom to run whatever software you want
- ➔ Full control of your own data
- ➔ Potentially cheaper than a subscription
- ➔ Giving your unused computer a new life
- ➔ Fun learning experience

Cons of self-hosting

- ➔ Initial learning curve
- ➔ Potentially more expensive than a subscription
- ➔ Poor scalability (both up and down)
- ➔ Inconsistent loading times across the globe
- ➔ Responsibility for security and maintenance

Thank you!