host$_{registry}$

Release 0.1.0

Oct 10, 2020

Contents:

1 README of host_registry **1**
- 1.1 Presentation . . . 1
- 1.2 Getting started . . . 1
- 1.3 Common tasks . . . 1

2 Notes on host_registry **3**
- 2.1 Presentation . . . 3
- 2.2 Typical topology . . . 3
- 2.3 The new fact . . . 6
- 2.4 Port-redirection . . . 6
- 2.5 Service registry . . . 9

3 Notes on web-app concepts **11**
- 3.1 Introduction . . . 11
- 3.2 The multi-user application . . . 11

4 Indices and tables **13**

CHAPTER 1

README of host_registry

1.1 Presentation

This repository contains experimental code for replacing *reverse-proxy* with *port-redirection*.

More information under readthedocs

Or visit the example page blabla

1.2 Getting started

In a bash-terminal:

```
git clone https://github.com/charlyoleg/host_registry
cd host_registry
npm i
npm run start_hrs
```

In a second bash-terminal:

```
curl -k https://ZZZ.LocalHost:8443/aa
```

1.3 Common tasks

Update the server:

```
pm2 stop rediry
git pull
npm run build_hrs
pm2 restart rediry
```

Update the access_log-report:

```
npm run accesslog_html
```

and visit the page accesslog_report

CHAPTER 2

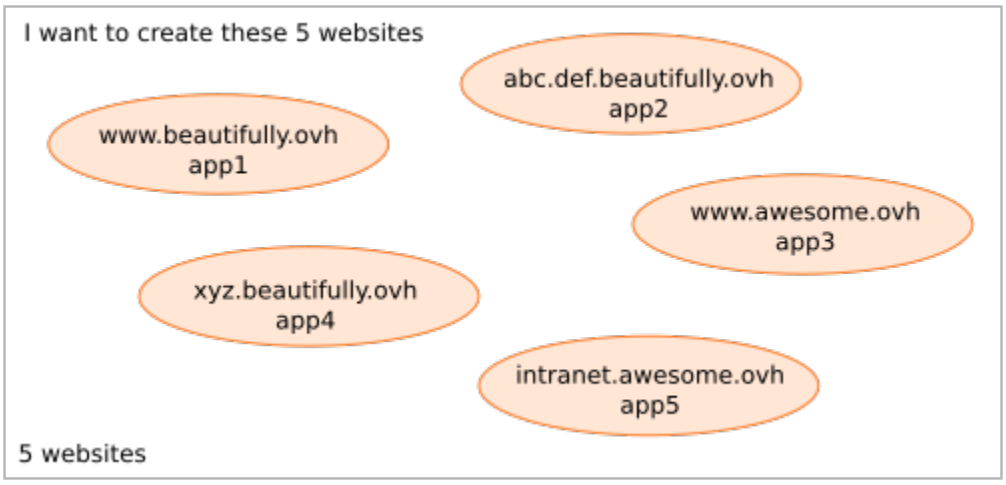
Notes on host_registry

2.1 Presentation

This document aims at exploring possibilities to setup a website.

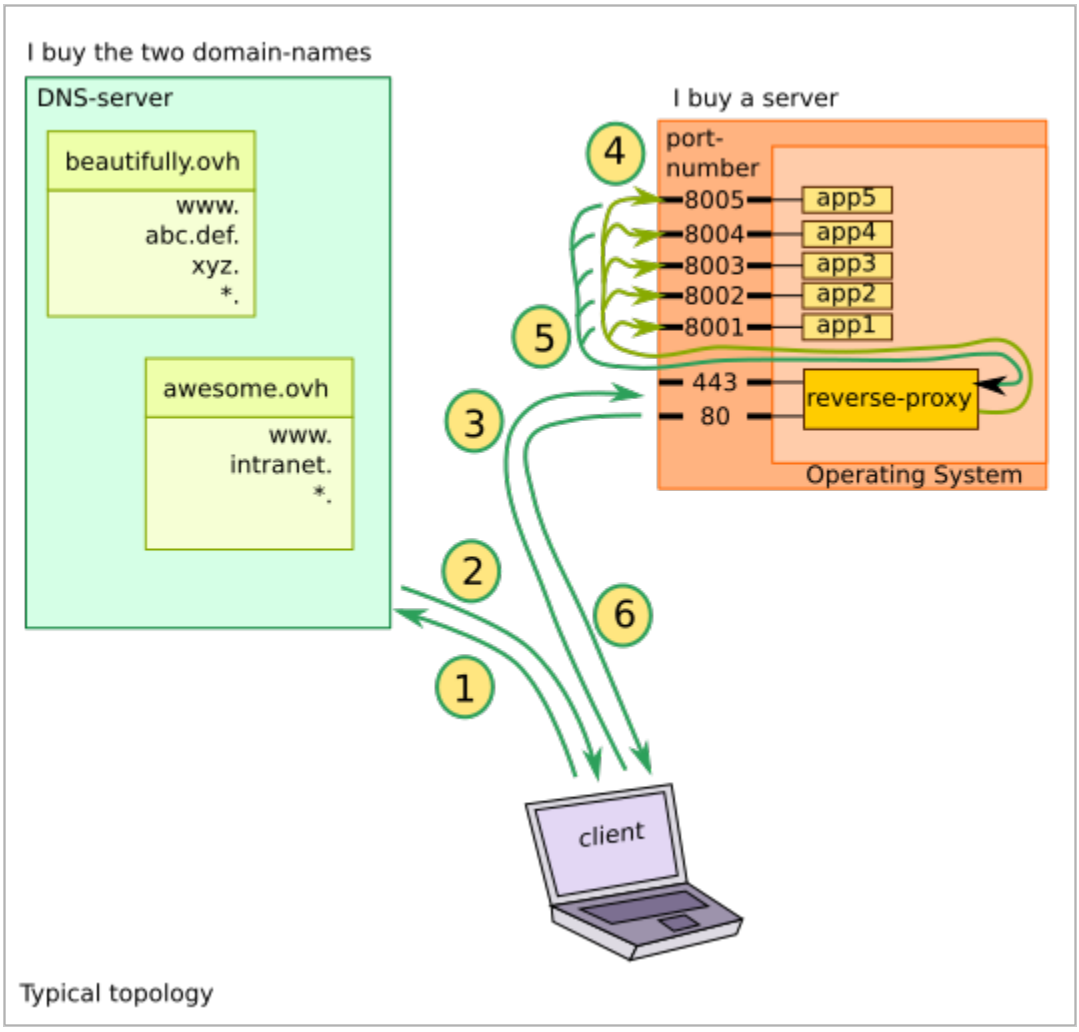
2.2 Typical topology

Let's say, I want to create 5 websites with the following names:



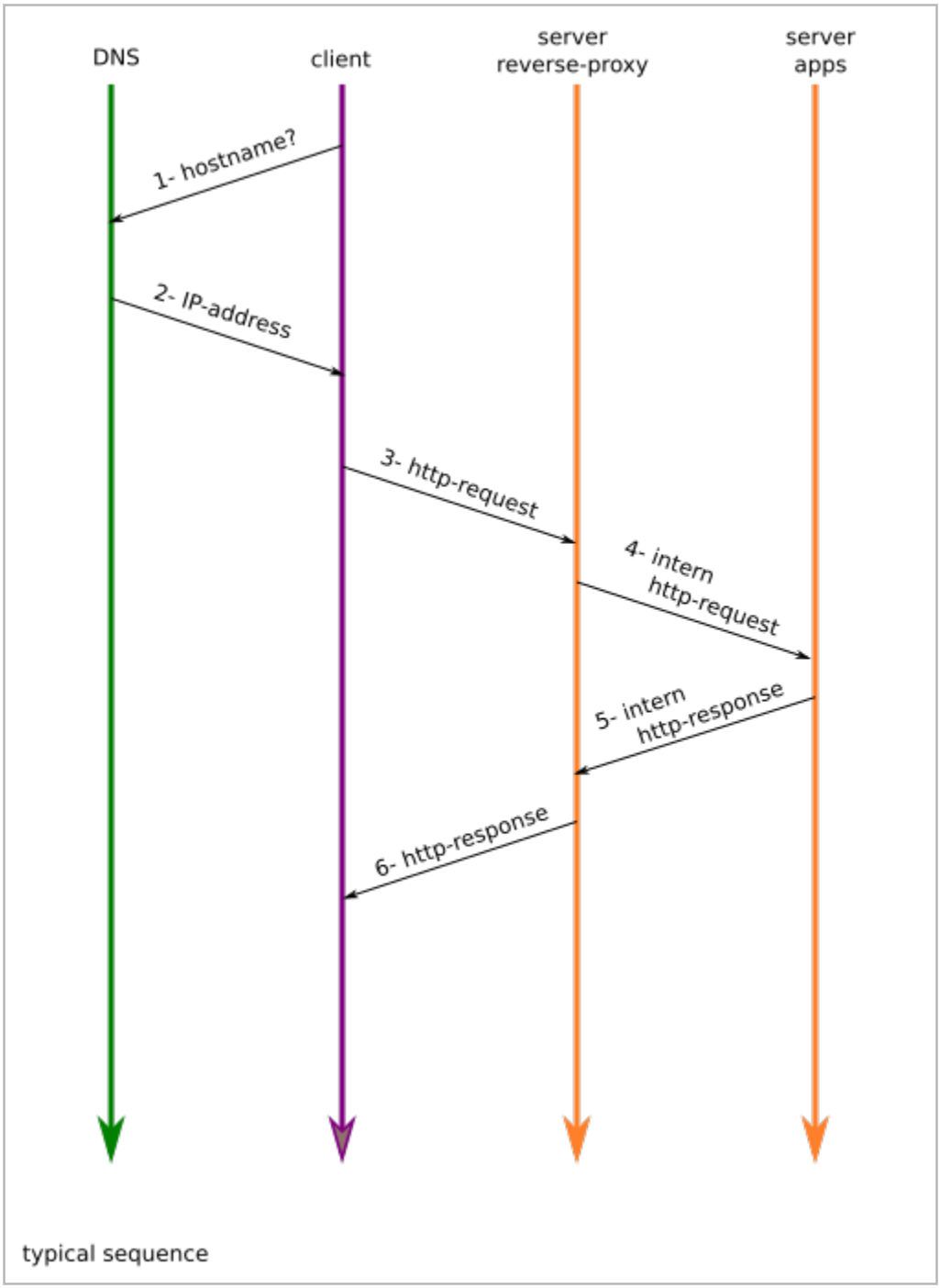
I have to buy the two **domain-names**, *beautifully.ovh* and *awesomely.ovh*. I also have to buy a server (i.e. a computer working 24 hours a day and connected to internet), such as vps, where my web-applications and databases will run.

For each *domain-name* I can register (almost) any sub-domains. For each sub-domain I can register a A-record (IPv4), a AAAA-record (IPv6) or a CNAME-record (i.e. an alias to an other hostanme) . The wilcard * let you provide an IP-address to all sub-domains, not explicitly registered.



With this typical topology, when someone wants to visit one of the websites from his client-laptop, the following sequence happens:

1. the laptop asks a DNS server the translation of the hostname of the URL
2. the DNS returns the corresponding IP-address
3. the laptop send the http-request to my server
4. the rever-proxy listening to the standard port-numbers forwards the requests
5. the web-application process the request and provides the result to the reverse-proxy
6. the reverse-proxy forward the result to the laptop



By the second http-request, the two first steps are skipped, as the laptop knows already the IP-address of my server.

The *reverse-proxy* manages to forward the requests to the right application thanks to the *destination-hostname* written in the *http-header*. So the *reverse-proxy* won't work if you replace the server-hostname with its IP-address in the URL.

2.2.1 Pros and Cons

Pros:

- the standard port-numbers are used in the http-request, so the port-number is not shown in the URL

- the reverse-proxy can also act as *load-balancer*

Cons:

- Websocket runs over the intermediate reverse-proxy
- restriction by ssl / https certificates
- the reverse-proxy process might become a bottle-neck
- the equivalence *ip-address:port* <> *hostname:port* is broken

2.3 The new fact

nodejs offers the capacity of directly serving http-requests from internet. Before nodejs, when generating the html-pages with perl, php or python, a *revers-proxy*, such as apache or nginx, was required.

Notice, that nowadays, some *reverse-proxy* are implemented with nodejs with some of the following solutions:

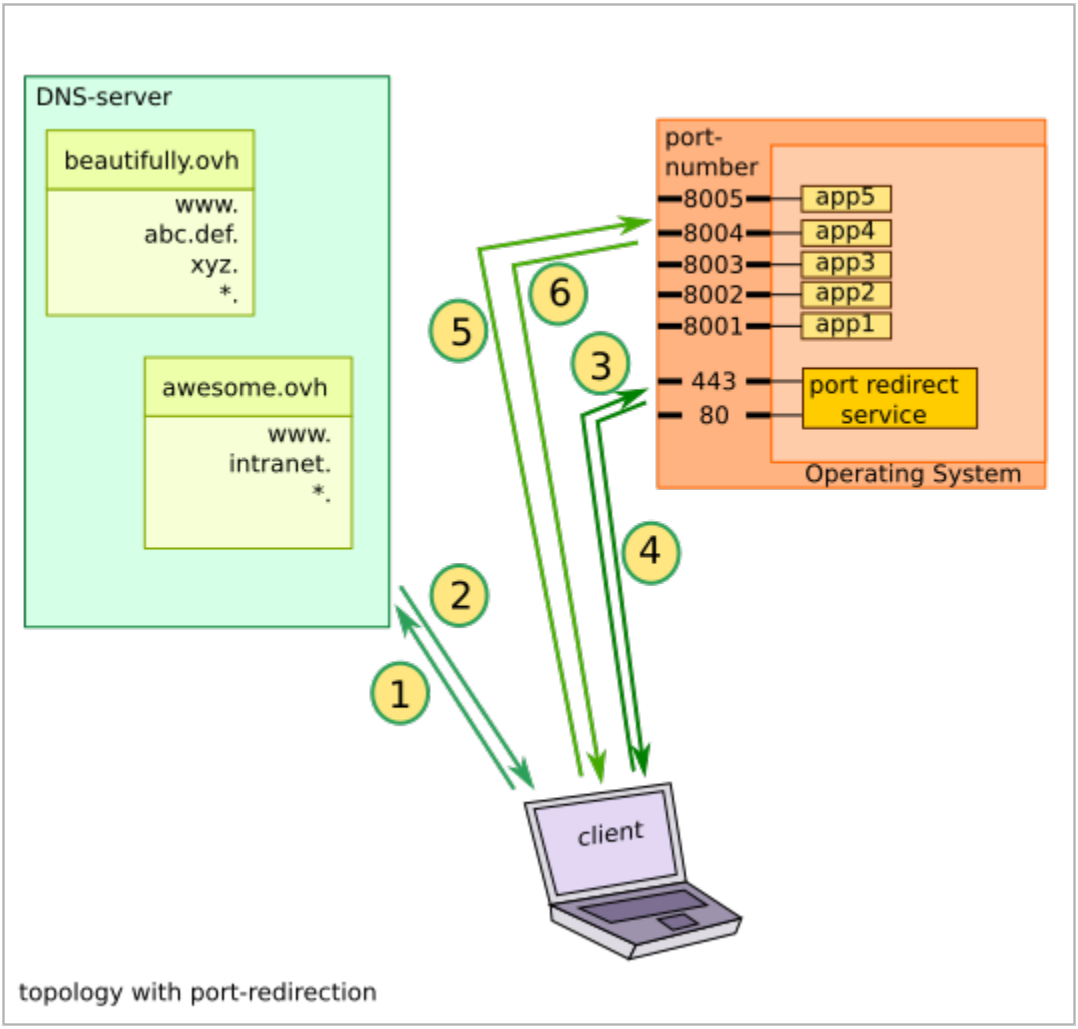
- https://github.com/http-party/node-http-proxy
- https://github.com/chimurai/http-proxy-middleware
- https://github.com/expressjs/vhost
- https://github.com/OptimalBits/redbird
- https://github.com/villadora/express-http-proxy

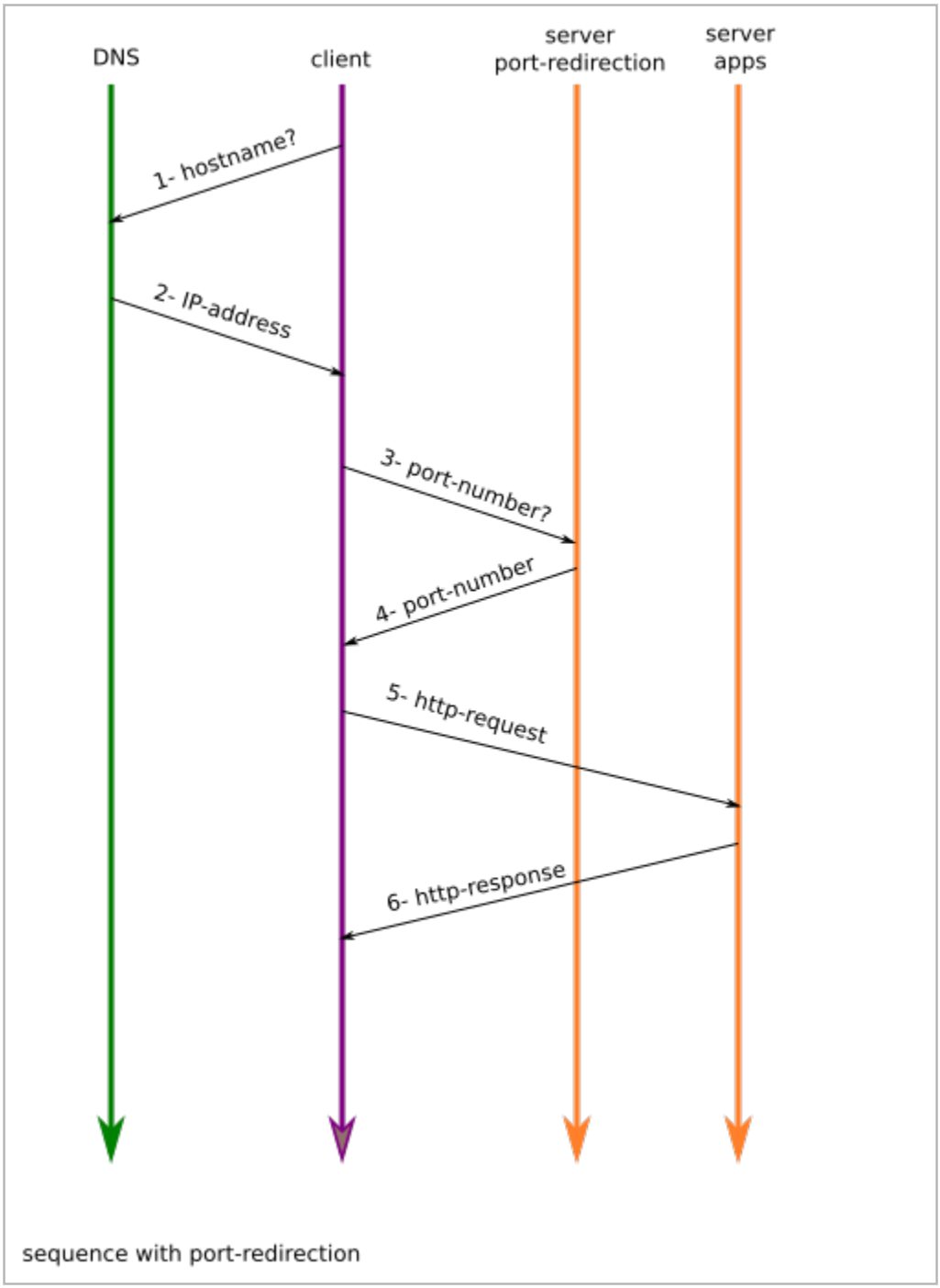
So now, each web-application, implemented with nodejs, can directly face internet. The wish to get rid of the *reverse-proxy* is getting higher. Two options are described below:

- a port-redirection service
- a service-registry

2.4 Port-redirection

This solution is implemented in this git-repository.


DNS-server
beautifully.ovh
www.
abc.def.
xyz.
*.
awesome.ovh
www.
intranet.
*.
port-
number
8005
app5
8004
app4
8003
app3
8002
app2
8001
app1
443
80
port redirect
service
Operating System
1
2
3
4
5
6
client
topology with port-redirection



By the second http-request, the four first steps are skipped, as the laptop knows already the IP-address and the port-number of the web-application.

2.4.1 Pros and Cons

Pros:

- each web-application works nicely independently. No central process.
- Websocket and https certificates are served directly

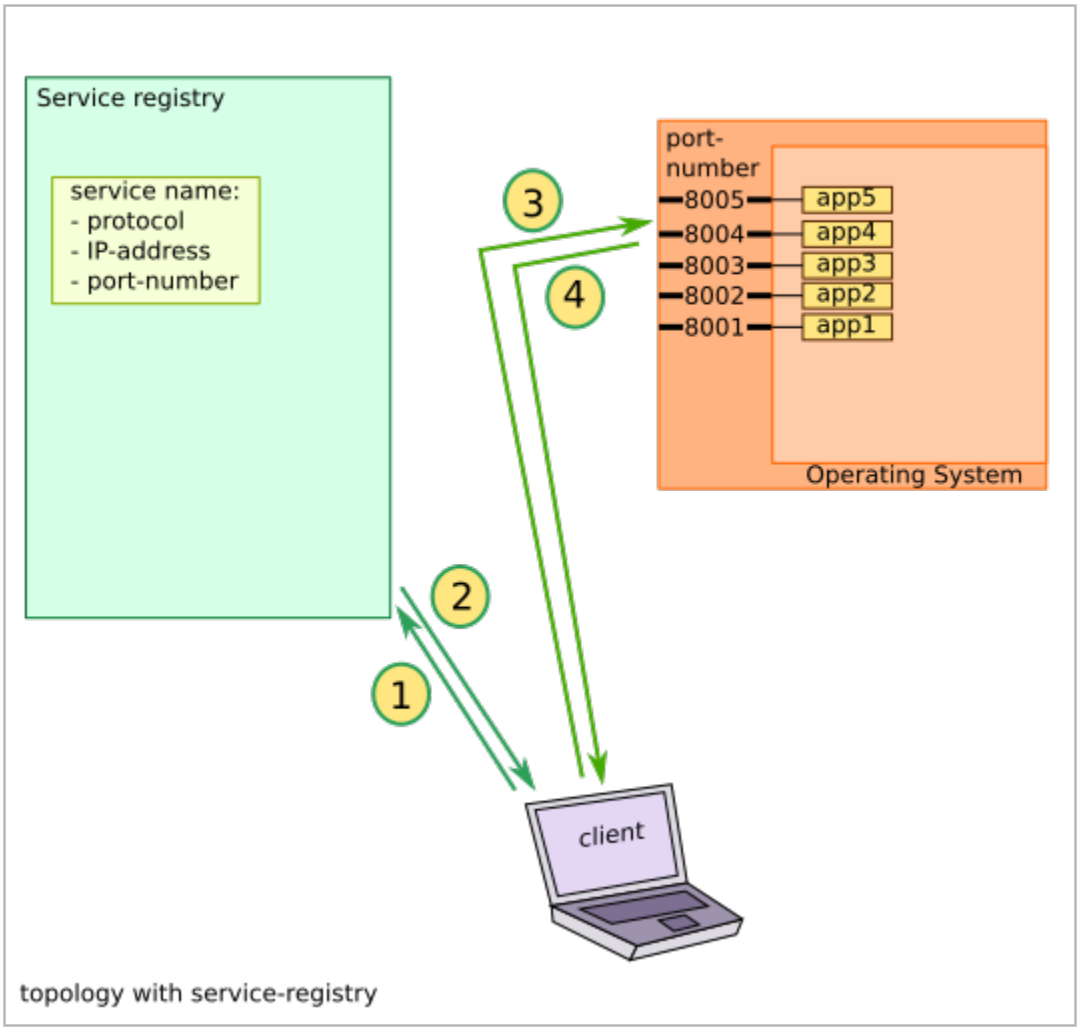
- the equivalence *ip-address:port* <> *hostname:port* works as expected

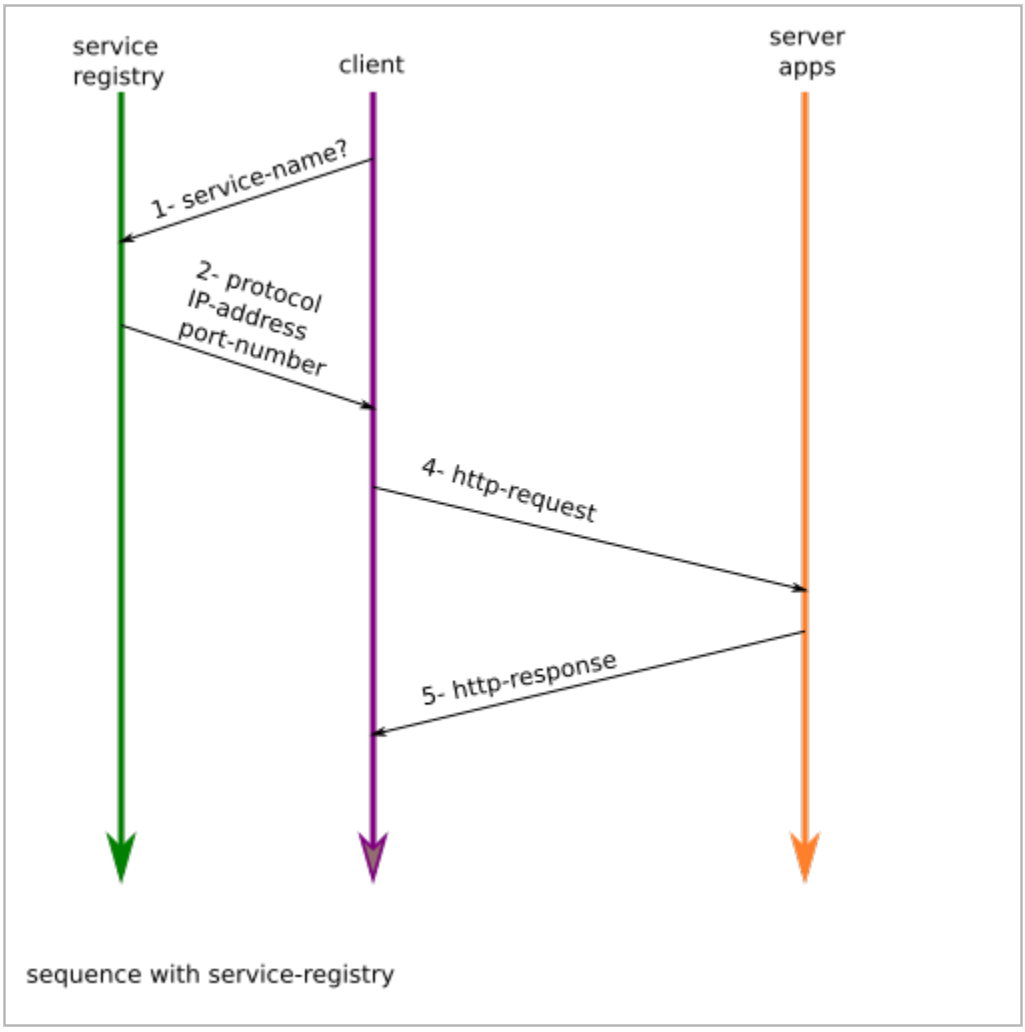
Cons:

- the port-number of the web-application if visible in the URL of the http-request

2.5 Service registry

The idea is more futuristic and not implemented yet.





Some more ideas for the service registry:

- hash instead of a service-name
- the service could update its IP-address and port-number dynamically
- the *load-balancing* could be implemented from the client-side (with a list of servers for each service)

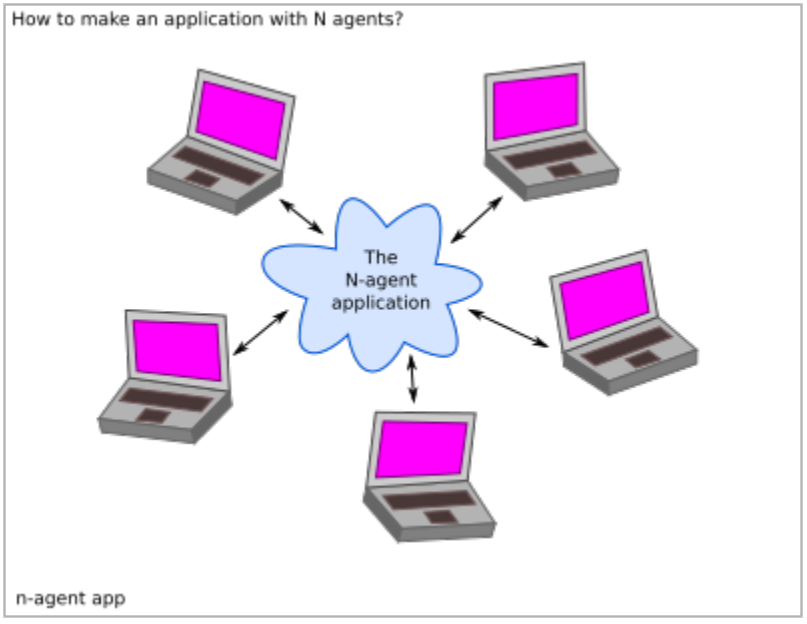
CHAPTER 3

Notes on web-app concepts

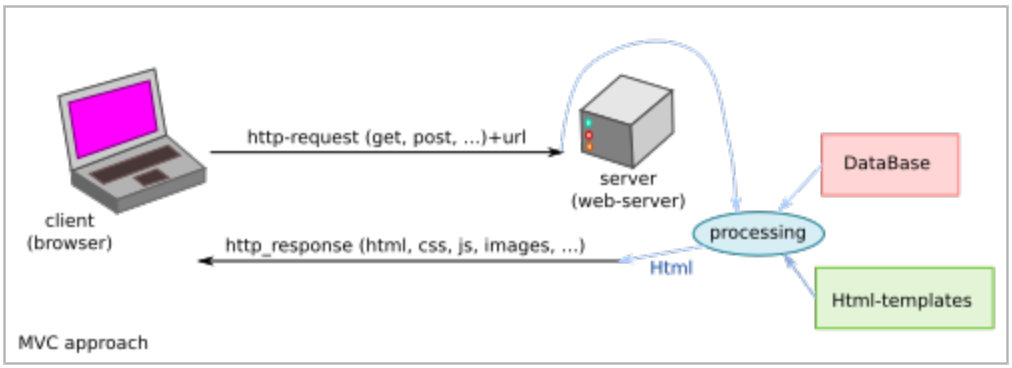
3.1 Introduction

The world of web technologies is so prolific, that man may lost the direction. From time to time, we have to rethink: Which problematic? Which solutions?

3.2 The multi-user application

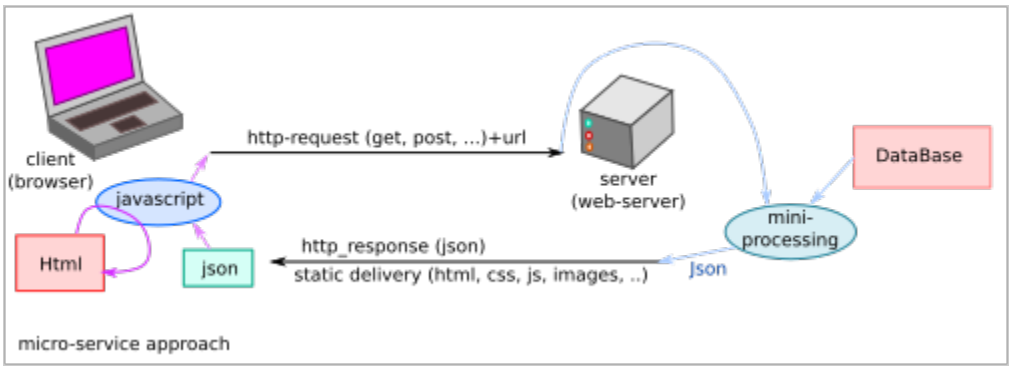


How to create one application for multiple users or agents?



The MVC approach is the historical solution. One big application running on one *central server* generates *html* interpreted remotely by browser. The most popular frameworks with the MVC approach are:

- ruby-on-rails
- sails



The micro-service aims at transferring the core of the application on the client-side. The server-side is reduced to the minimum. The backend is split as much as possible in independent parts.

The frontend could be bundled in an electron, nw or pwa application.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search