Tampere University

COMP.SE.140
Lecture about ”orchestration”

Tampereen yliopisto
Tampere University

Vagrant

Vagrant intro

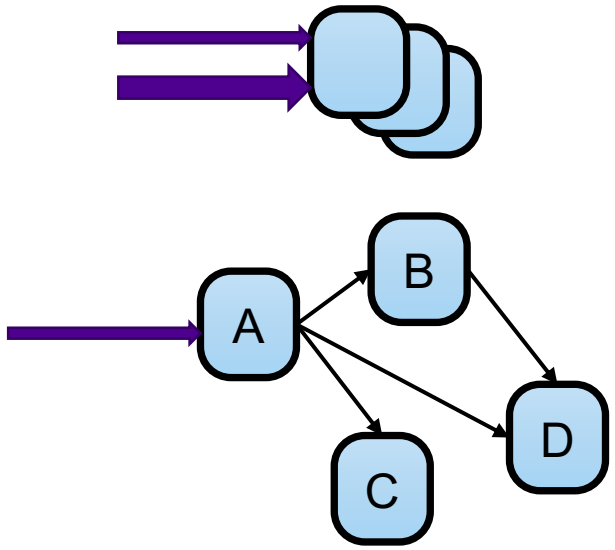
- A way to create and distribute **development environments** as virtual machine (full VMs – not containers)
- If time lets look: https://www.vagrantup.com/intro/index.html

Vagrant vs Docker (https://www.vagrantup.com/intro/vs/docker.html)

- Vagrant is a tool focused on providing a consistent development environment workflow across multiple operating systems. Docker is a container management that can consistently run software as long as a containerization system exists.
- Containers are generally more lightweight than virtual machines, so starting and stopping containers is extremely fast. Docker uses the native containerization functionality on macOS, Linux, and Windows.
- Currently, Docker lacks support for certain operating systems (such as BSD). If your target deployment is one of these operating systems, Docker will not provide the same production parity as a tool like Vagrant. Vagrant will allow you to run a Windows development environment on Mac or Linux, as well.
- For microservice heavy environments, Docker can be attractive because you can easily start a single Docker VM and start many containers above that very quickly. This is a good use case for Docker. Vagrant can do this as well with the Docker provider. A primary benefit for Vagrant is a consistent workflow but there are many cases where a pure-Docker workflow does make sense.
- Both Vagrant and Docker have a vast library of community-contributed "images" or "boxes" to choose from.

What are typical cloud applications

- Networks of containers!



Logically like:

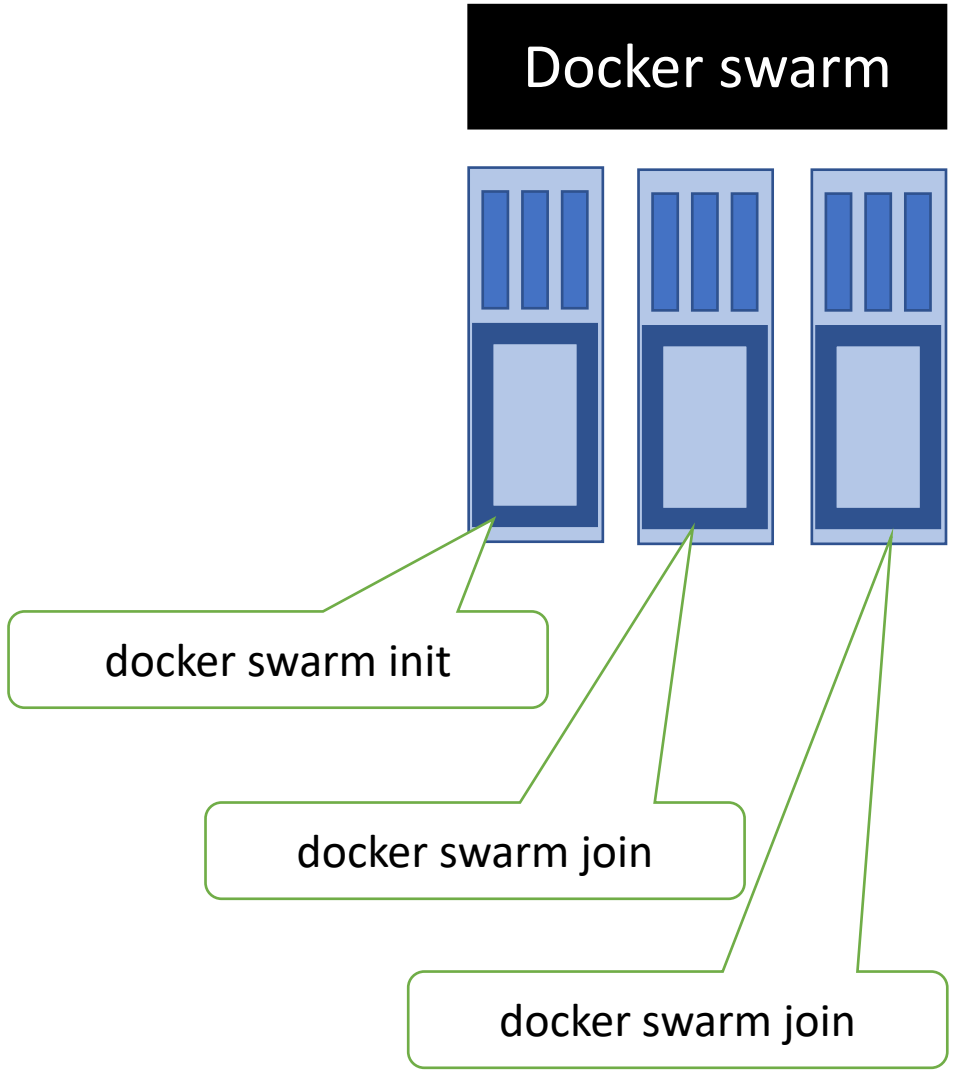
```
A() {
    B();
    C();
    D();
}
```

But implemented as inter-process communication.

```
A() {
    http.get(B:80);
    http.get(C:80);
    http.get(D:80);
}
```

Docker Swarm

- Clustering for scalability
- A swarm is a group of host running docker in swarm mode
- A host can be either a *manager* or *worker*
- Workers run *services*
- Manager assigns tasks to worker nodes
  - *Load balancing*



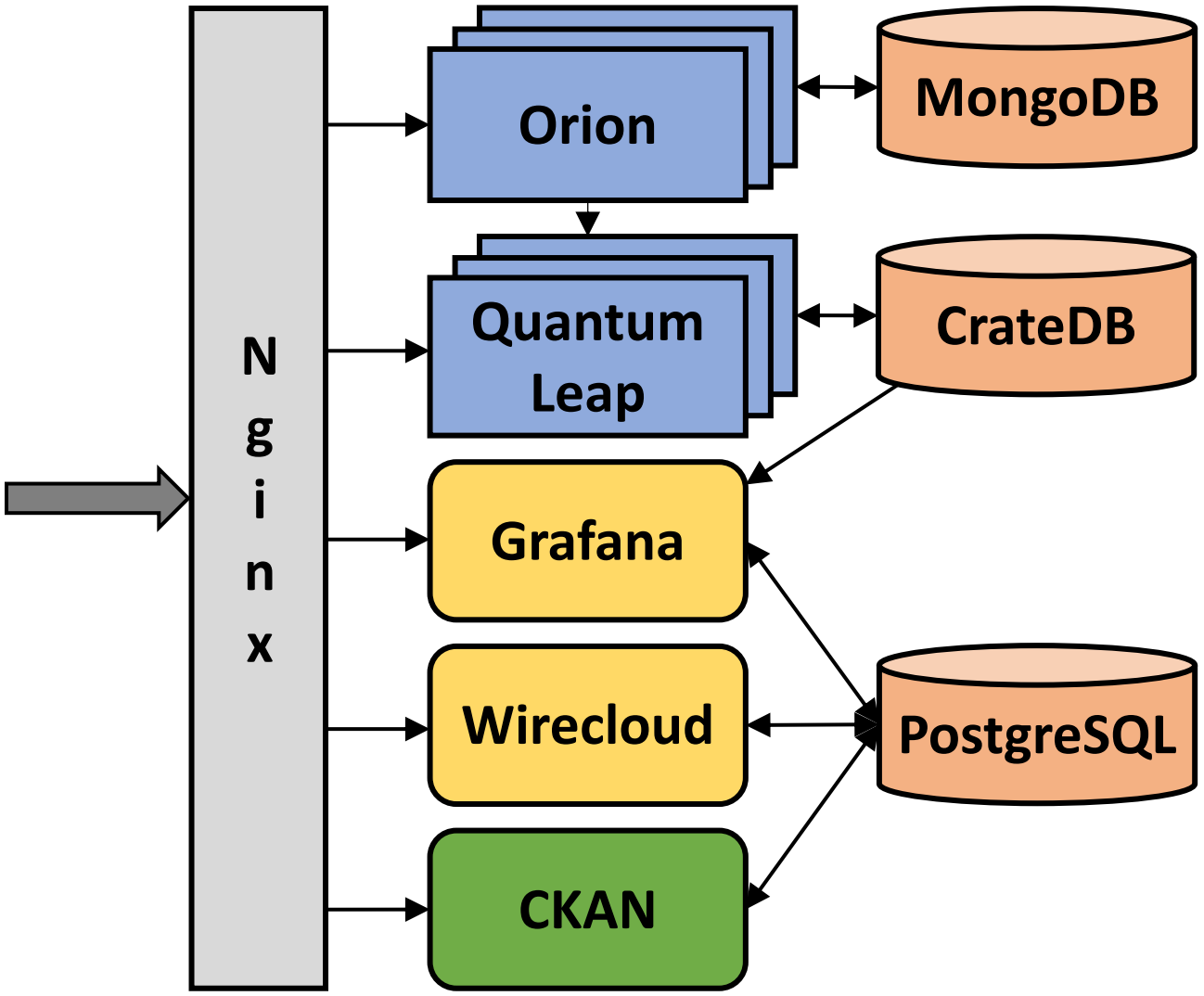
Tampereen yliopisto
Tampere University

From docs.docker.com

```
$ docker swarm init --advertise-addr 192.168.99.100
Swarm initialized: current node (dxn1zf6l61qsb1josjja83ngz) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c \
    192.168.99.100:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

```
$ docker swarm join \
  --token  SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c \
  192.168.99.100:2377

This node joined a swarm as a worker.
```

FIWARE platform architecture



FIWARE Core Component

Dashboard Component

Data Management Component

Database

Access control, proxy server

FIWARE access control components (Keyrock, Wilma and AuthZForce) are not included in this document.

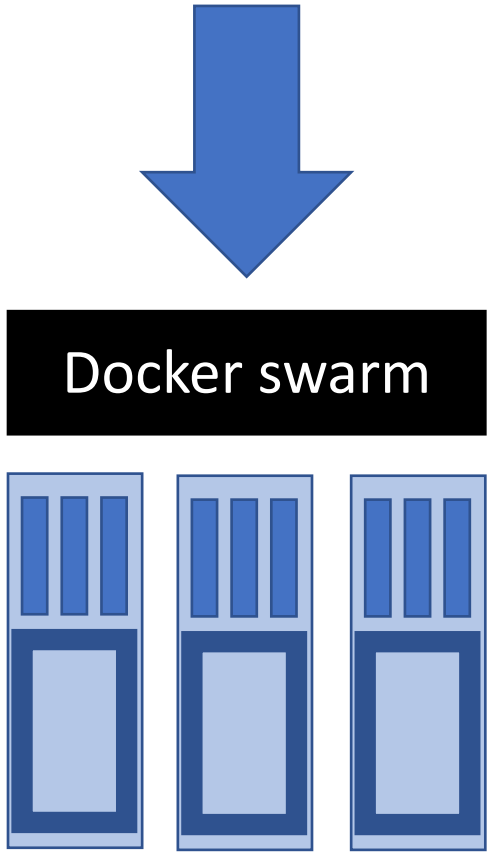
https://github.com/cityiot/CityIoT-platform/blob/master/start_fiware.sh

```
docker service ls

ID                  NAME                MODE                REPLICAS            IMAGE
6eenqanud5k3        orion_orion         replicated          5/5                 fiware/orion:2.3.0
vncsavctf9ib        mongo-rs_controller replicated          1/1                 smartsdk/mongo-rs-con…
lx6muj0xkwrl        mongo-rs_mongodb    global              1/1                 mongo:3.6.16
o3q85b6rfpli        ql_quantumleap      replicated          3/3                 smartsdk/quantumleap:0.
z669mqi1ir0f        ql_cratedb          global              1/1                 crate:3.3.5
n8dlhqmczecd        nginx_nginx         global              1/1                 nginx:1.15.8
```

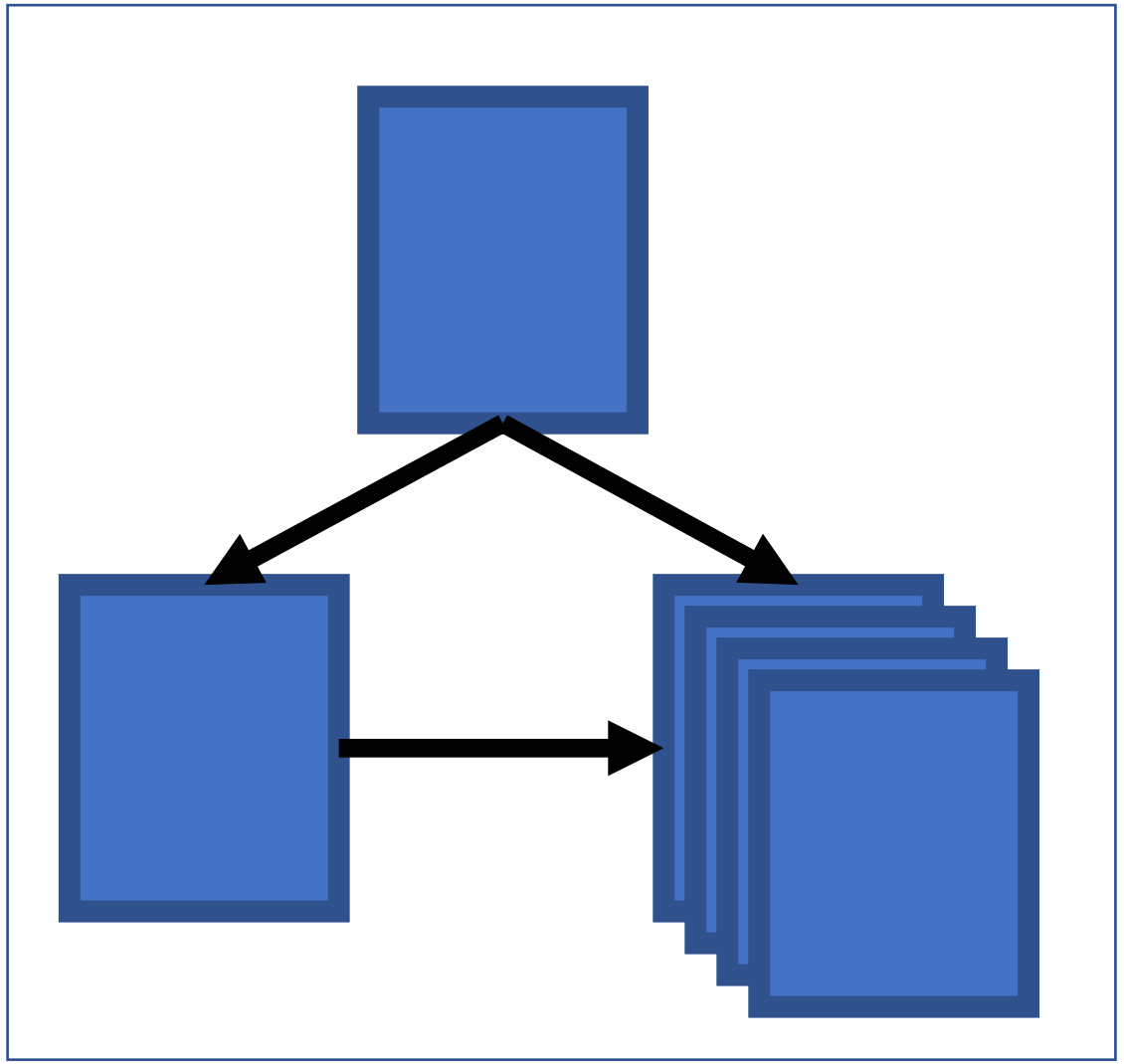
Tampereen yliopisto
Tampere University

Docker swarm - docker compose

Orchestration

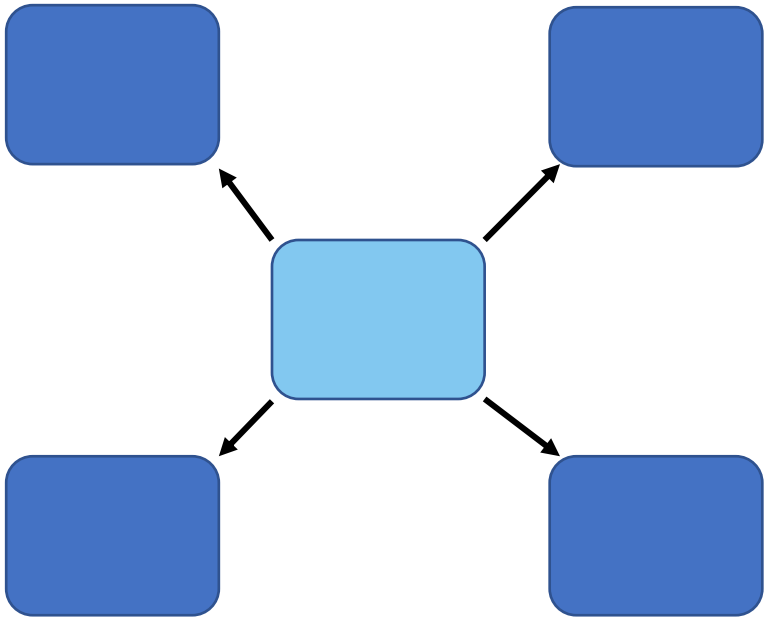


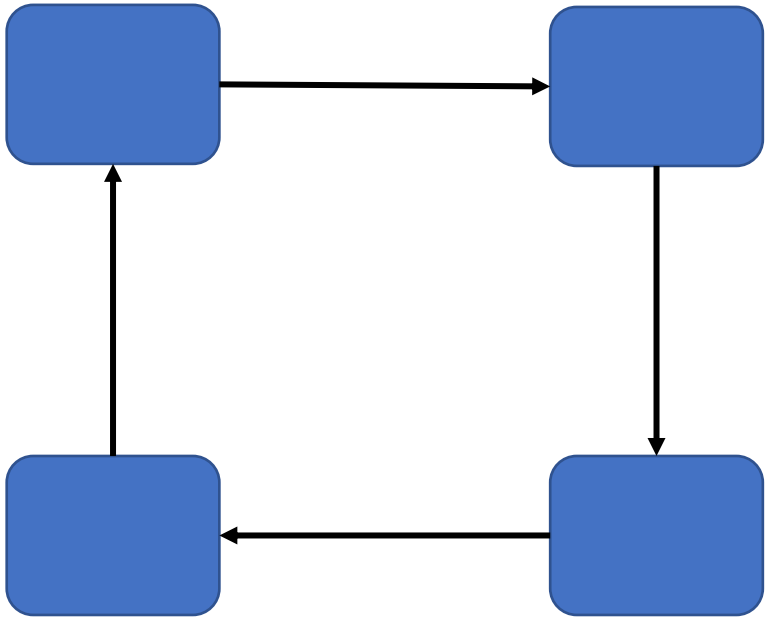
Docket compose



Tampereen yliopisto
Tampere University

Orchestration vs Choreography



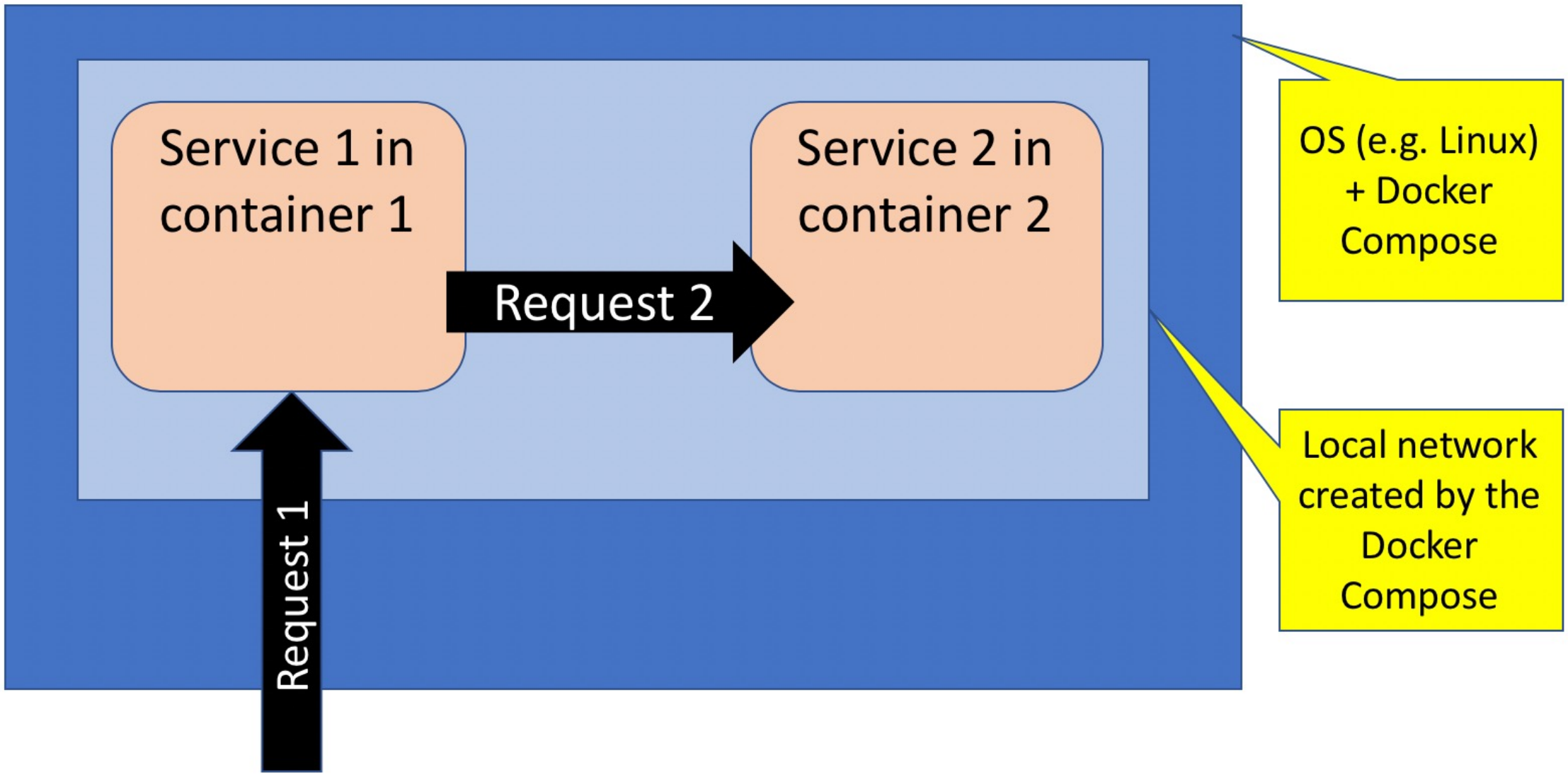


Tampereen yliopisto
Tampere University

What is ”cloud orchestration”?

Two results of googling

- **Orchestration** is the automated configuration, coordination, and management of computer systems and software
- Cloud orchestration is the use of programming technology to manage the interconnections and interactions among workloads on public and private cloud infrastructure. It connects automated tasks into a cohesive workflow to accomplish a goal, with permissions oversight and policy enforcement.


Service 1 in container 1
Request 2
Service 2 in container 2
Request 1
OS (e.g. Linux) + Docker Compose
Local network created by the Docker Compose

Tampereen yliopisto
Tampere University

Docker compose

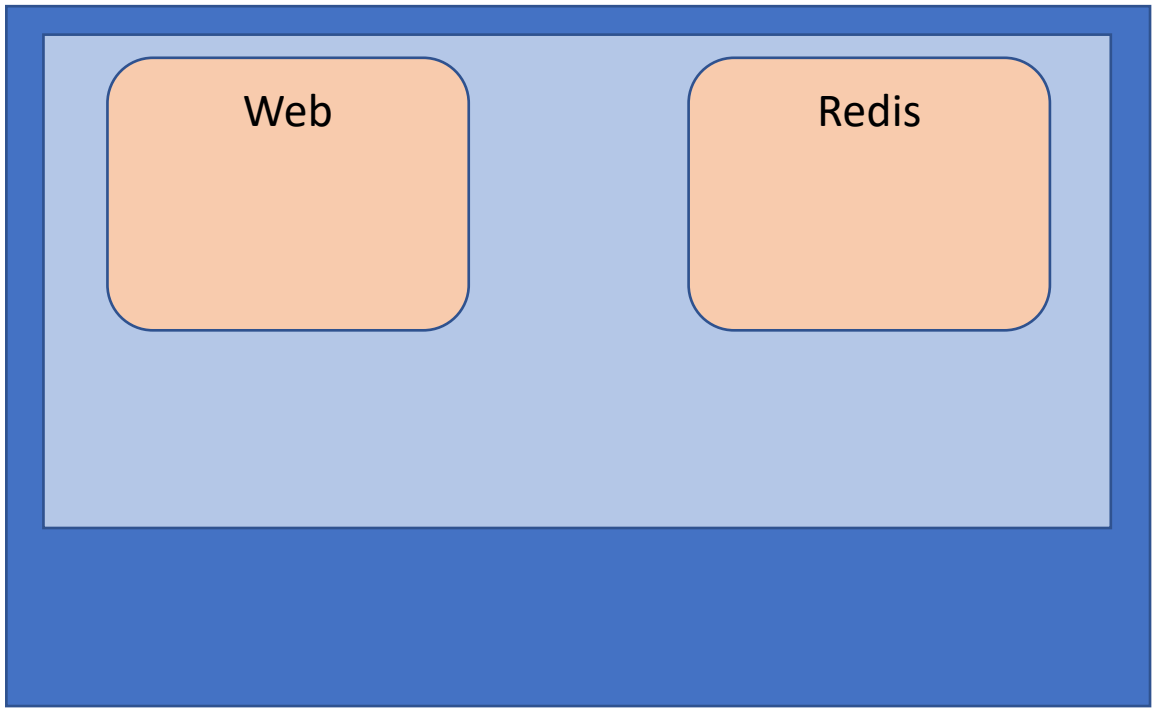
```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

```
$ docker-compose up –d
$ ./run_tests
$ docker-compose down
```

Tampereen yliopisto
Tampere University

Docker compose

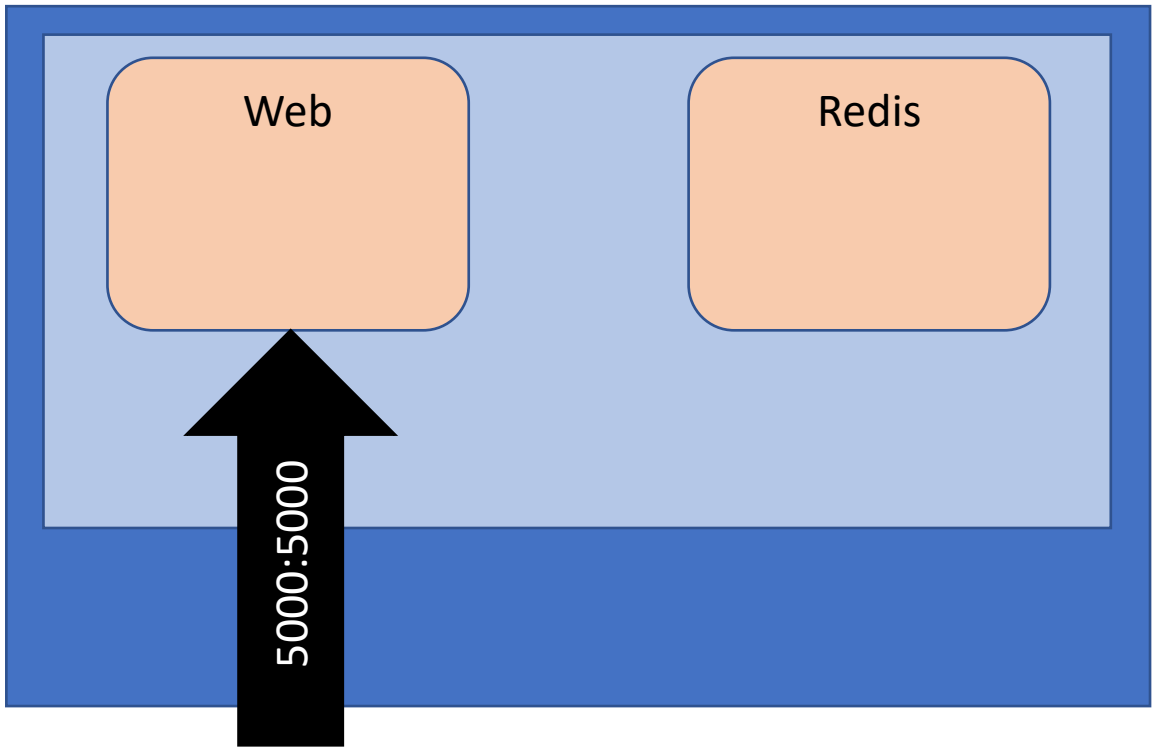
```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```



Tampereen yliopisto
Tampere University

Docker compose

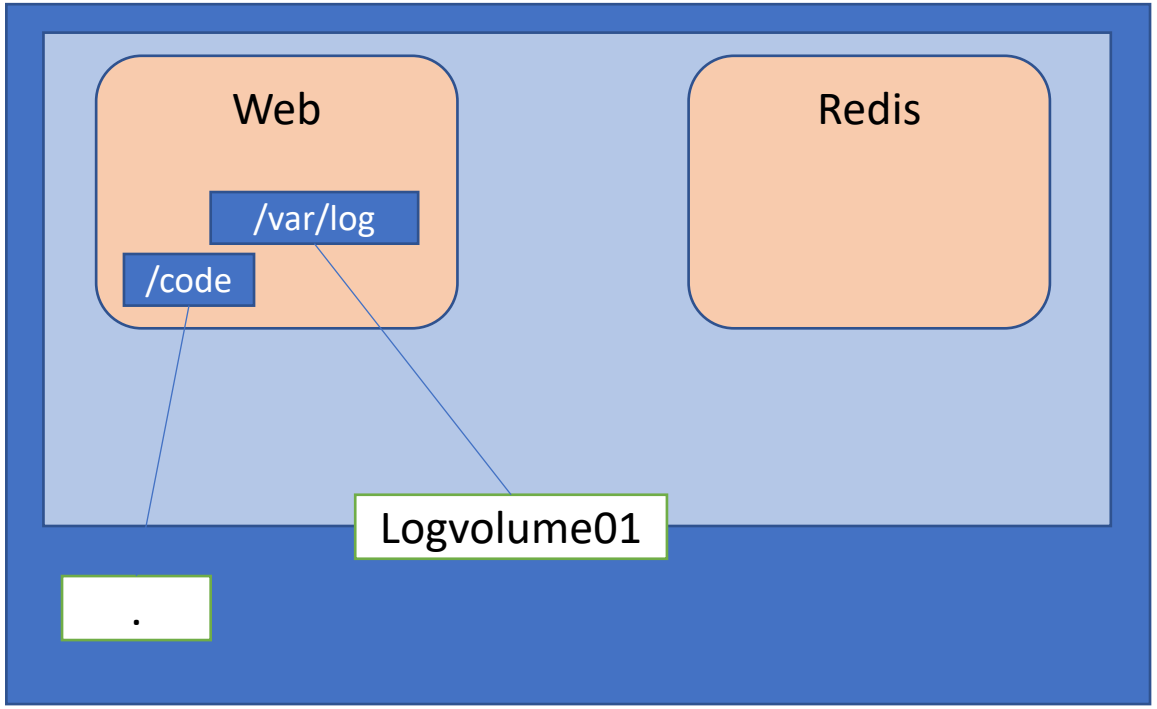
```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```



Tampereen yliopisto
Tampere University

Docker compose

```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```



Tampereen yliopisto
Tampere University

Docker compose

```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

```
$ docker-compose up –d
$ ./run_tests
$ docker-compose down
```

Tampereen yliopisto
Tampere University

Many options: e.g., automatic restarting

```yaml
version: "3.7"
services:
  redis:
    image: redis:alpine
    deploy:
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
```

Tampereen yliopisto
Tampere University

YAML

- Wikipedia: **YAML** ("YAML Ain't Markup Language") is a human-readable data-serialization language. It is commonly used for configuration files
- Spaces for indentation – have a syntactical meaning
- https://www.tutorialspoint.com/yaml/yaml_basics.htm

Tampereen yliopisto
Tampere University

YAML -> JSON

```yaml
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

```json
{
  "version": "3",
  "services": {
    "web": {
      "build": ".",
      "ports": [
        "5000:5000"
      ],
      "volumes": [
        ".:/code",
        "logvolume01:/var/log"
      ],
      "links": [
        "redis"
      ]
    },
    "redis": {
      "image": "redis"
    }
  },
  "volumes": {
    "logvolume01": {}
  }
}
```

Tampereen yliopisto
Tampere University

Nice looking tutorial

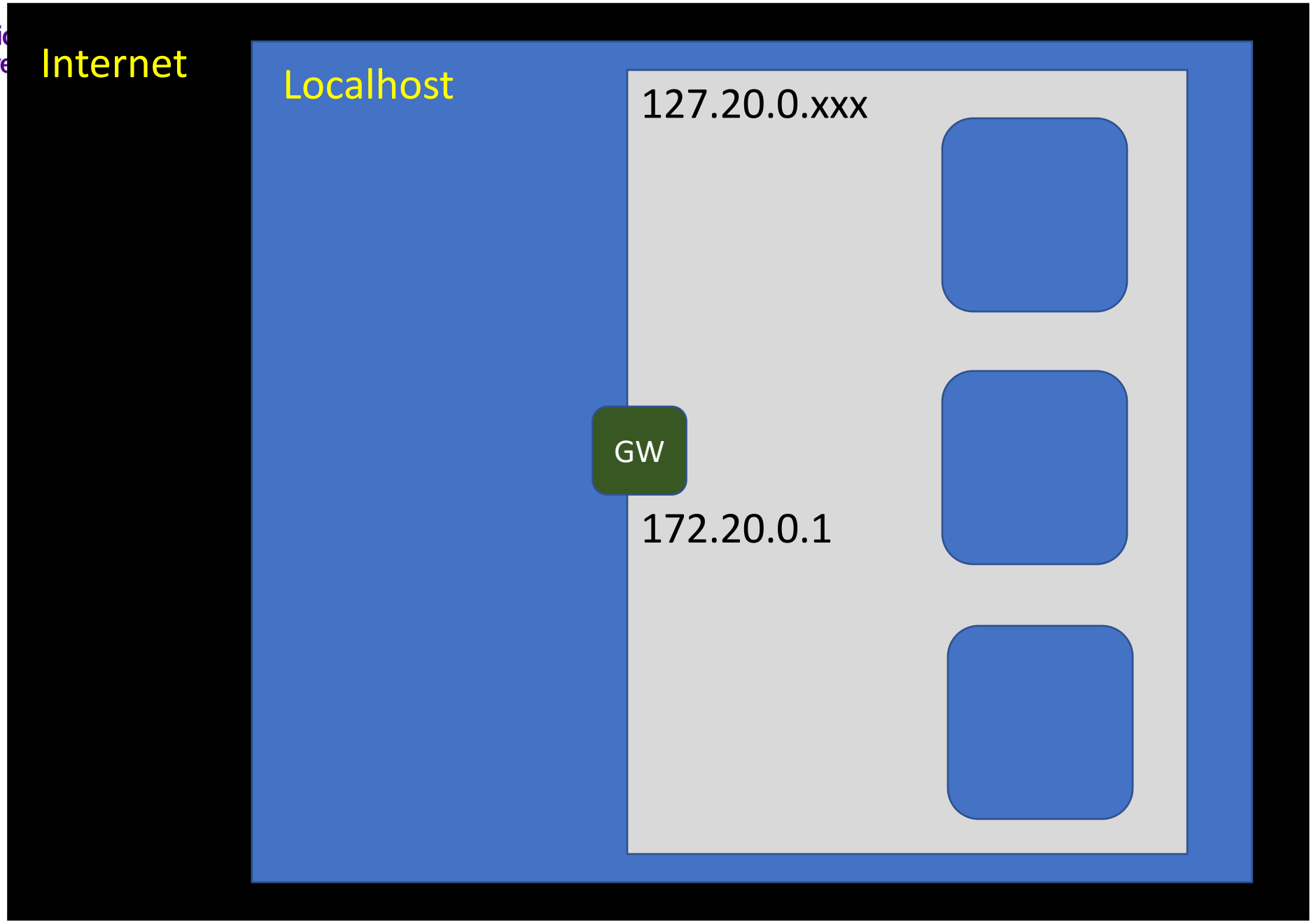
- https://www.baeldung.com/docker-compose

Tampereen yliopisto
Tampere University

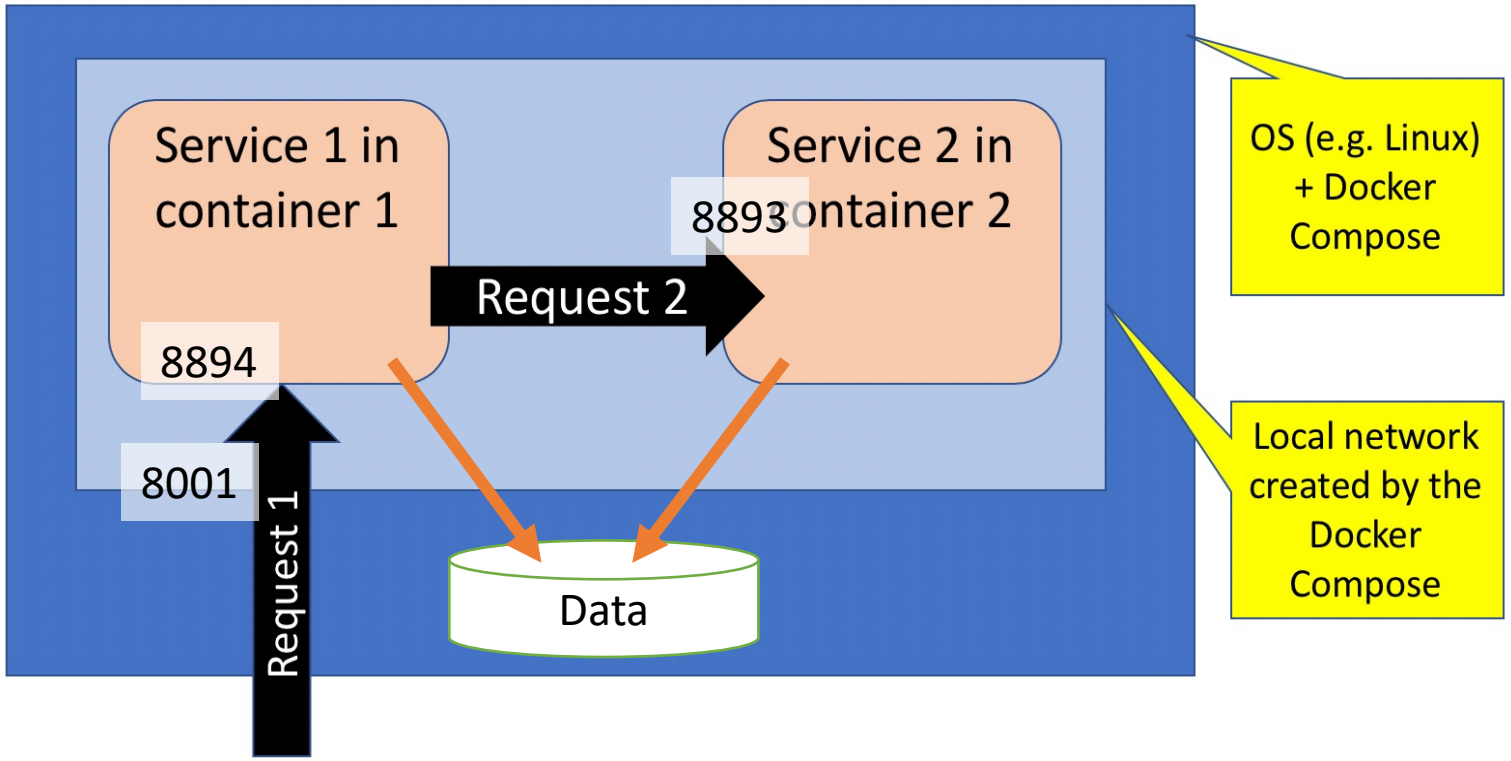
Networking aspects

```
version: '3'
services:
  pinger:
    image: "pinger"
    ports:
      - "8893:8893"
    networks:
      - pingnet
    volumes:
      - ./data:/data
    environment:
      ServiceName: service_2
  pingrelay:
    build: "pingrelay"
    ports:
      - "8004:8894"
    networks:
      - pingnet
    volumes:
      - ./data:/data
    environment:
      ServiceName: service_1
networks:
   pingnet:

volumes:
  data: {}
```


```
[
  {
    "Name": "composetest_pingnet",
    "Id": "42d79573d3b3cf…",
    "Created": "2019-02-14T20:08:36.226402086+02:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.20.0.0/16",
          "Gateway": "172.20.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": true,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {},
    "Labels": {
      "com.docker.compose.network": "pingnet",
      "com.docker.compose.project": "composetest",
      "com.docker.compose.version": "1.23.1"
    }
  }
]
```


Internet
Localhost
127.20.0.xxx
GW
172.20.0.1

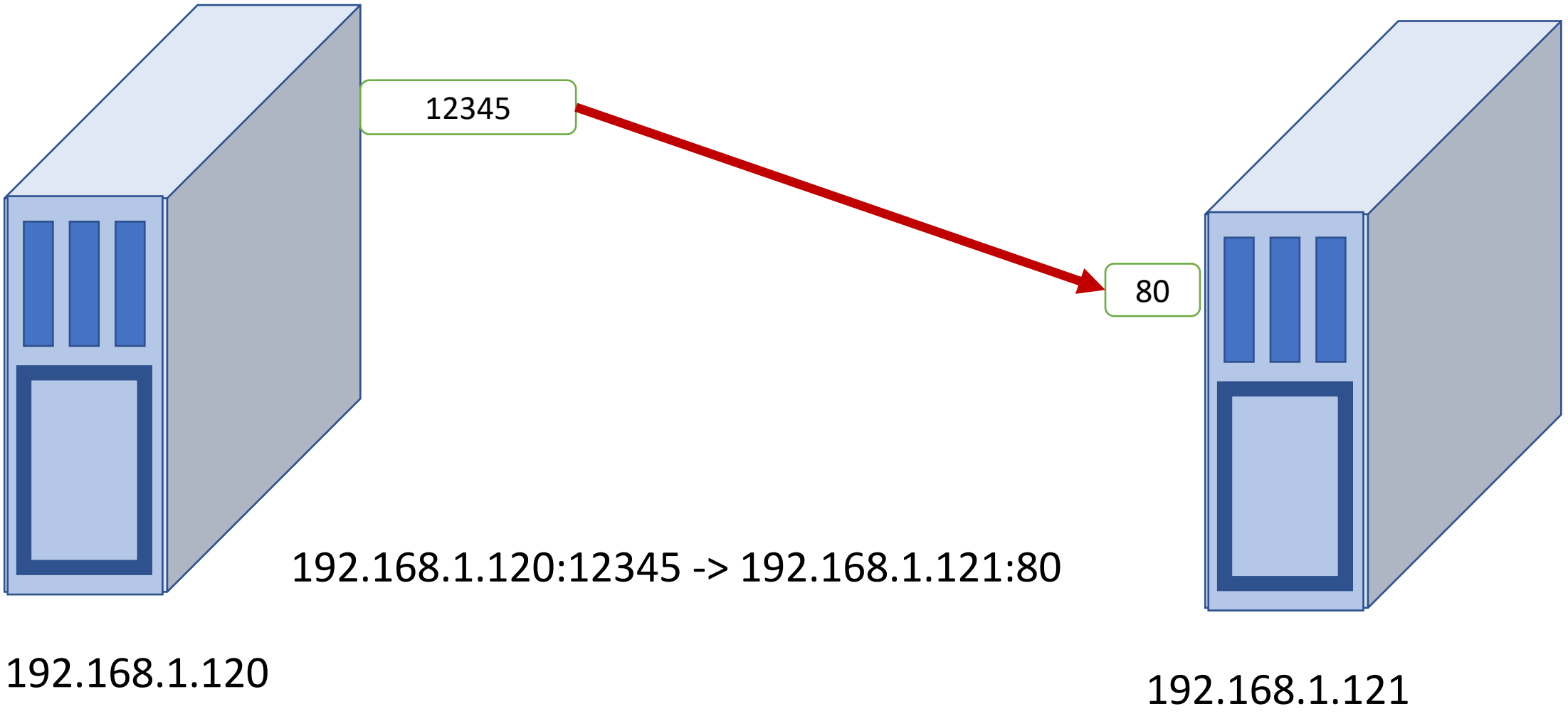
```
version: '3'
services:
  pinger:
    image: "pinger"
    ports:
      - "8893:8893"
    networks:
      - pingnet
    volumes:
      - ./data:/data
    environment:
      ServiceName: service_2
  pingrelay:
    build: "pingrelay"
    ports:
      - "8004:8894"
    networks:
      - pingnet
    volumes:
      - ./data:/data
    environment:
      ServiceName: service_1
networks:
   pingnet:

volumes:
  data: {}
```



Do you see ”errors”?

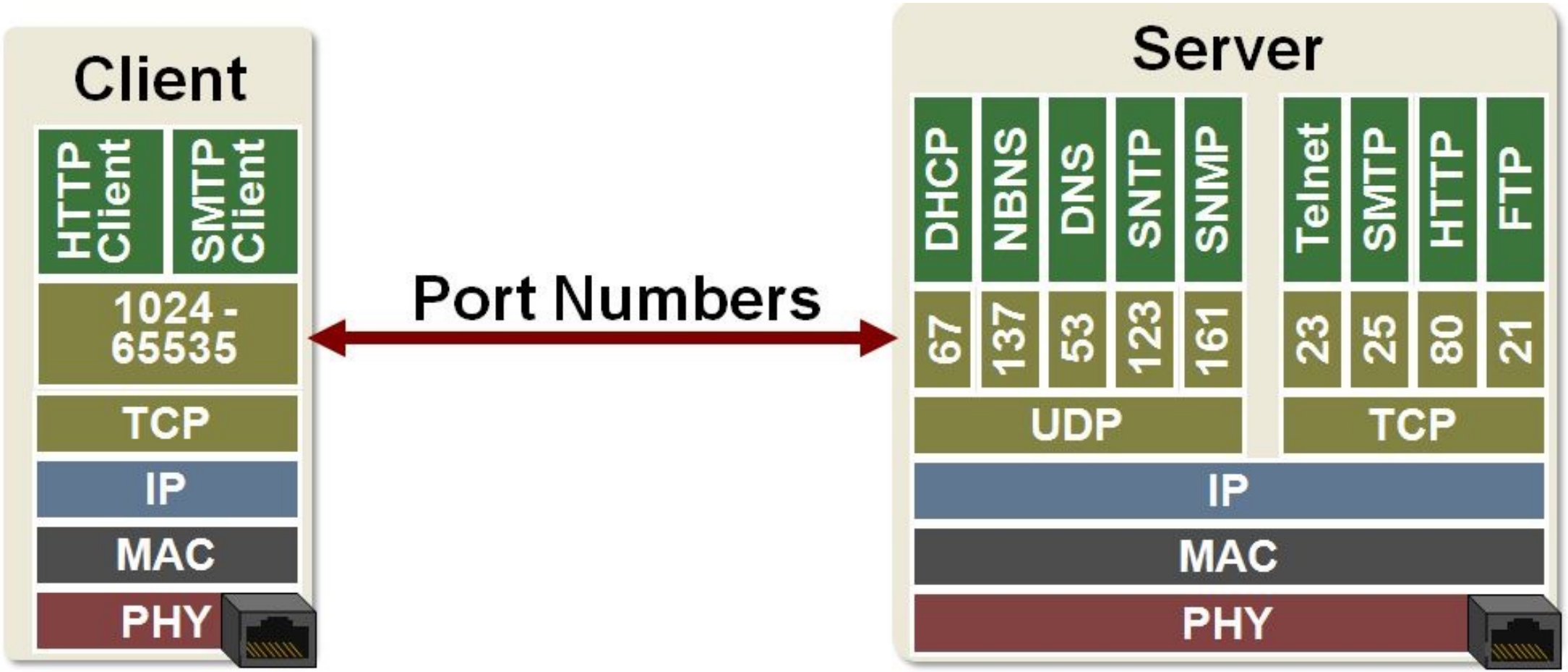
Tampereen yliopisto
Tampere University

Something very basic

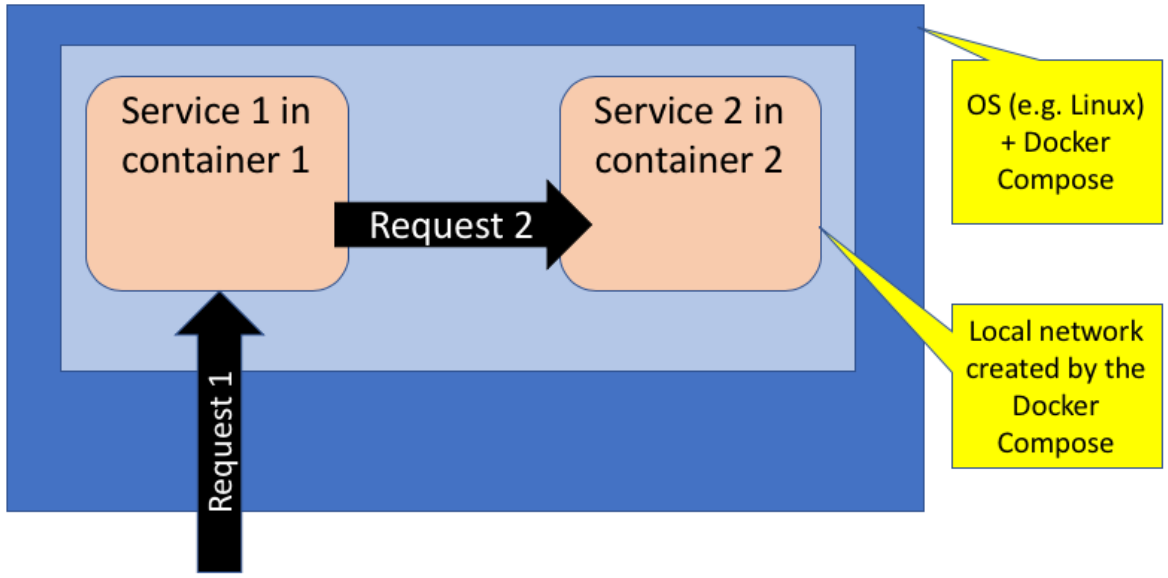


Tampereen yliopisto
Tampere University

https://microchipdeveloper.com/tcpip:tcp-ip-ports



Your task



Service/application 1 should:

- As a response to incoming Request 1 send an HTTP GET request to Service2
- Compose a response from (4 lines of text)
  - “Hello from “ + <Remote IP address and port of the “ to “ + <Local IP address and port of Service1>
    Response of the above request to Service2
  - Return the composed response

Service/application 2 should

- As a response to incoming Request 2 compose a response from
  - “Hello from “ + <Remote IP address and port of the incoming Request2>
  - “ to “ + <Local IP address and port of Service2>
- Return the composed response

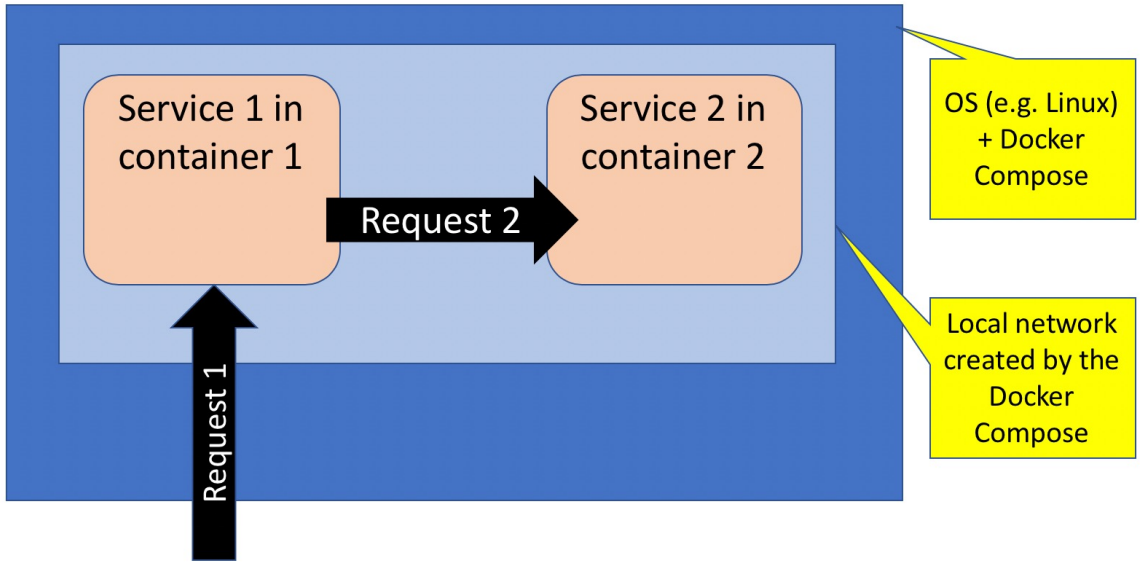
Remember to check the final version!

- By remote address/port we means the address of the host that sent the request. For example, in nodejs these can be tested with the following code:

```
http.createServer(function (req, res) {
 console.log("Req came from " + req.client.remoteAddress
+
              ":" + req.client.remotePort);
 console.log("Req served at " + req.client.localAddress
+
              ":" + req.client.localPort);
}).listen(port);
```

- Note that the above does not exactly meet requirements

Your task



- You should write *Dockerfiles* for the both services and *docker-compose.yaml* to start both containers so that Service1 is exposed in port number 8001. The docker-compose should also create a private network that allows Services 1 and 2 to communicate with each other but the only external Service 1.
- The service1 is assumed to be under development, so the image is rebuilt often (hint you may use ”build:” -primitive in *docker-compose.yaml*. Service2 is a reused service and you may pre-build the image. Image can be stored locally though.
- After the system is ready the student should return.
- Content of Docker and docker-compose.yaml files
- Explained response to Request 1 (that contains also response from Request 2). E.g. a Word or PDF-file where you also explain why the addresses and port-numbers are like they are. (We want to ensure that you understand how your program works).
- Source codes of the applications in some git.

Tampereen yliopisto
Tampere University

How this will be checked

```
$ git clone <the git url you gave>
$ docker-compose up –build
$ curl localhost:8001
<output should follow the above requirement>
$ docker-compose down
```

Tampereen yliopisto
Tampere University

Hints

- Remember to backup your application and docker and compose files – you will need them in the future. E.g. to gitlab.
- It might be a good idea to create and test the applications first.
- You may need to visit https://docs.docker.com/compose/ and https://docs.docker.com/compose/networking/
- Docker images are easy to access, if they are tagged when build
- $ docker build --tag=pinger .
- If Docker image is rebuilt, docker-compose should also be given a hint that rebuilt should override the existing one
- $ docker-compose up --build

Tampereen yliopisto
Tampere University

Infrastructure as code

From: https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code

Infrastructure as Code (IaC) is

- the management of infrastructure (networks, virtual machines, load balancers, and connection topology) in a descriptive model,
- using the same versioning as DevOps team uses for source code.
- Like the principle that the same source code generates the same binary, an IaC model generates the same environment every time it is applied.
- IaC is a key DevOps practice and is used in conjunction with continuous delivery.

Tampereen yliopisto
Tampere University

Where are we now