

Distributed Systems (DSy)

Load Balancing

Thomas Bocek

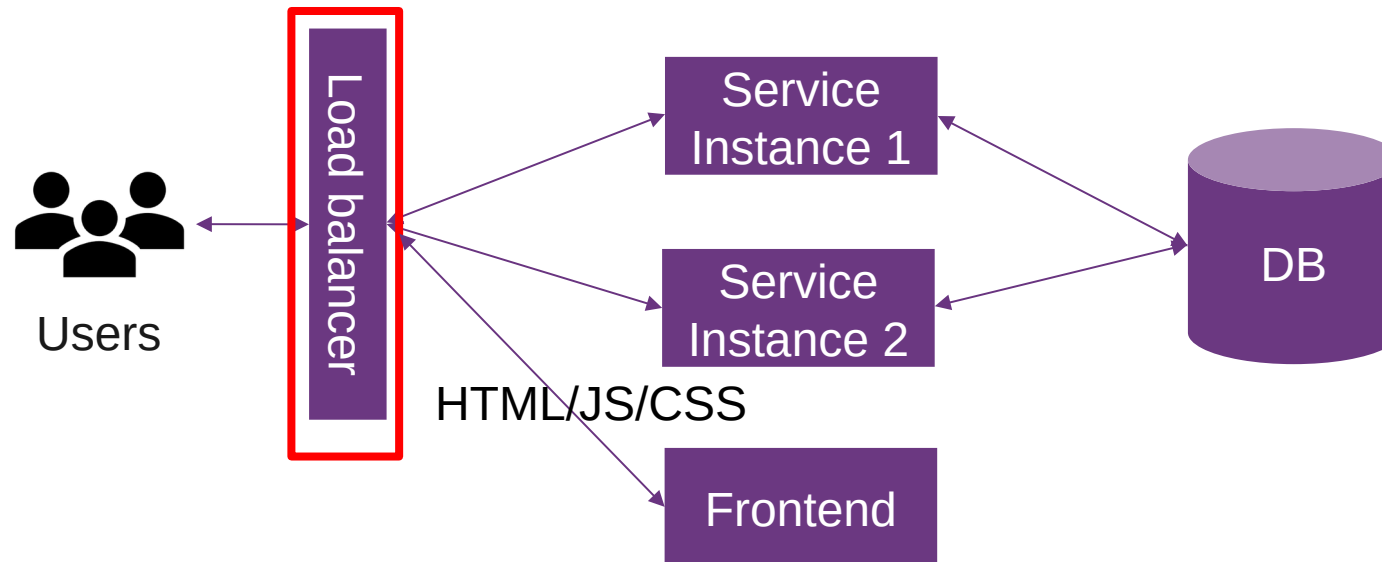
15.03.2025

Learning Goals

- Lecture 5 (Load Balancing)
 - What types of LB exists?
 - Which one to pick?
 - How can a LB be used for the challenge task? (DSy)

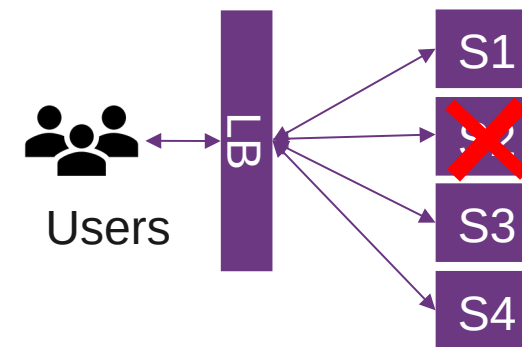
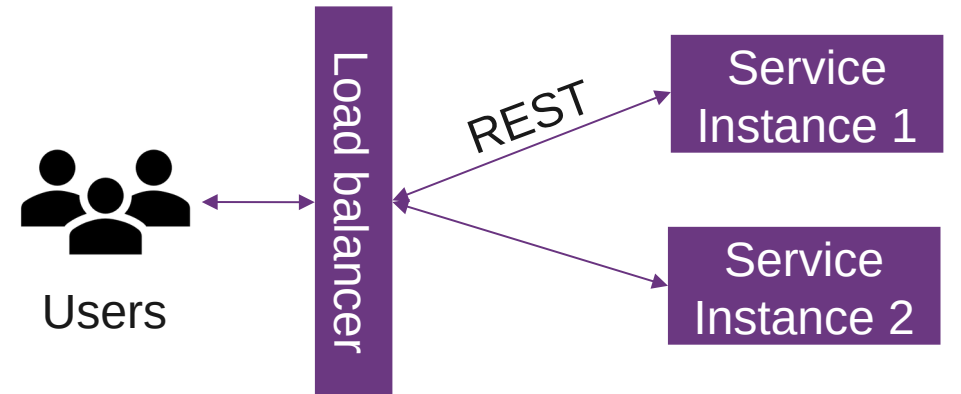
Load Balancing

- Challenge Task Requirement
 - 1) Load balancing with scalable service
 - 2) Failover of a service instance



Load Balancing

- What is load balancing
 - Distribution of workloads across multiple computing resources
 - Workloads (requests)
 - Computing resources (machines)
 - Distributes client requests or network load efficiently across multiple servers [\[link\]](#)
 - E.g., service get popular, high load on service
- horizontal scaling
- Why load balancing
 - Ensures high availability and reliability by sending requests only to servers that are online
 - Provides the flexibility to add or subtract servers as demand dictates



3 Types: Hardware, Cloud-based, Software load balancer

- Hardware load balancer
 - HW-LB use proprietary software, which often uses specialized processors
 - Less generic, more performance
 - Some use open-source SW, e.g., **HAProxy**
 - E.g., loadbalancer.org, F5, Cisco
 - Only if you control your datacenter



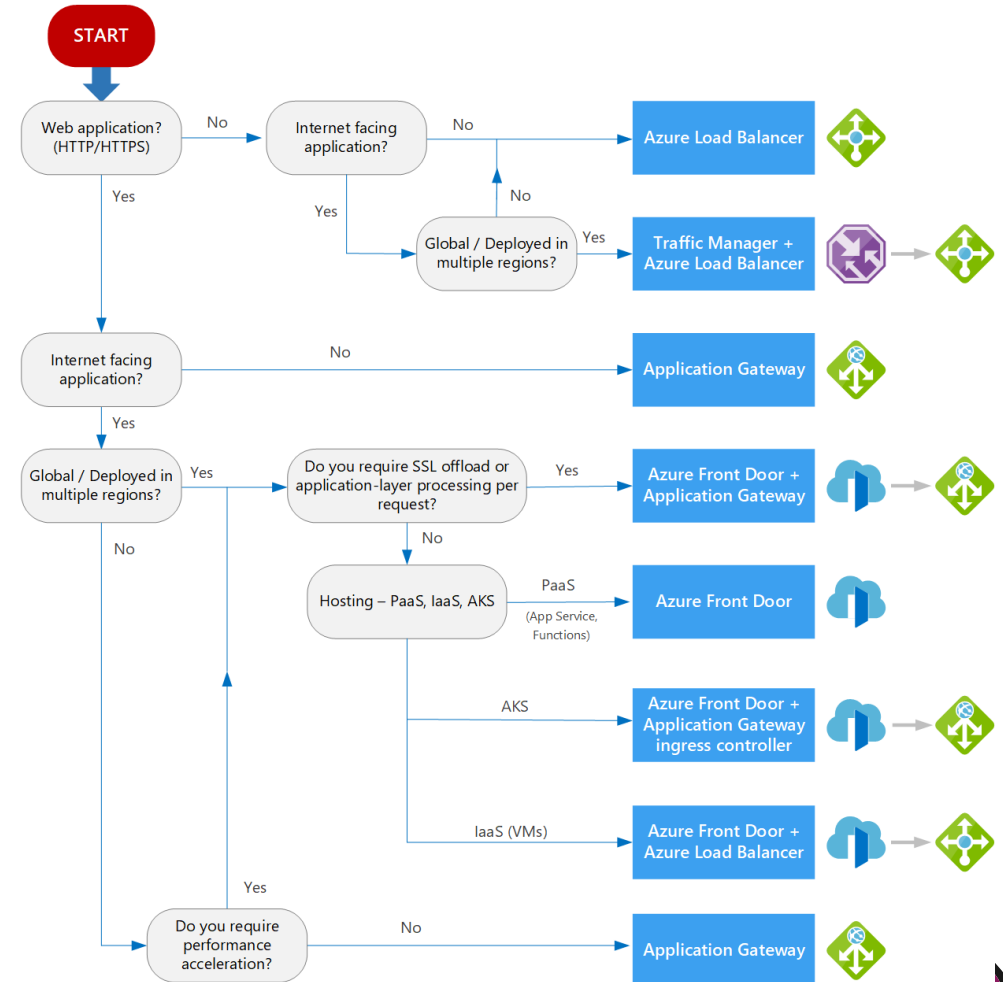
<https://www.loadbalancer.org/products/hardware/>

- Software load balancer
 - L2/L3: **Seesaw**
 - L4: **LoadMaster**, **HAProxy** (desc), **ZEVENET**, **Neutrino**, **Balance** (C), **Nginx**, **Gobetween**, **Traefik**
 - L7: **Envoy** (C++), **LoadMaster**, **HAProxy** (C), **ZEVENET**, **Neutrino** (Java/Scala), **Nginx** (C), **Traefik** (golang), **Gobetween** (golang), **Eureka** (Java) – services register at Eureka
- SW vs. SW / SW vs. HW
 - **strong opinions**, **funny opinions**, **other opinion**, but:
“We encourage users to benchmark Envoy in their own environments with a configuration similar to what they plan on using in production [source]”
- Benchmark, benchmarks

Types Load balancing

- Cloud-based load balancer
 - Pay for use
 - Many offerings
 - DIY? - No control over datacenter
 - **AWS**
 - Application Load Balancer ALB, (L7)
 - Network Load Balancer, (L4)
 - Classic Load Balancer (legacy)
 - Google Cloud, (L3, L4, L7)
 - Cloudflare (L4, L7)
 - DigitalOcean (L4)
 - Azure (L4, L7)

- Choices, choices, choices... e.g., Azure:



Software-based load balancing

- Layer 7: HTTP(S), layer 7: DNS
- DNS Load balancing
 - Round-robin DNS, very easy to setup, static, caching with no fast changes
 - **Split horizon DNS** - different DNS information, depending on source of the DNS request
- Reduced Downtime, Scalable, Redundancy
 - Client can decide what to do
 - **Negative caching impact!**
 - Used in bitcoin: dig dnsseed.emzy.de
- Layer 3: Anycast
 - You need an **AS** for that, **difficult and time consuming** – return the IP with lowest latency, e.g., **anycast as a service**, **Global Accelerator**

```
$TTL 3D
$ORIGIN tomp2p.net.
@ SOA ns.nope.ch. root.nope.ch. (2018030404 8H 2H 4W 3H)
                                NS      ns.nope.ch.
                                NS      ns.jos.li.
                                MX      10 mail.nope.ch.
                                A       188.40.119.115
                                TXT     "v=spf1 mx -all"
www                             A       188.40.119.115
lb                              A       188.40.119.115
Lb                              A       152.96.80.48
$INCLUDE "/etc/opendkim/keys/mail.txt"
$INCLUDE "/etc/bind/dmarc.txt"
```

dig lb.bocek.ch

```
--- lb.bocek.ch ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.025/0.035/0.046/0.012 ms
draft@gserver:~$ ping lb.bocek.ch
PING lb.bocek.ch (188.40.119.115) 56(84) bytes of data.
64 bytes from jos.li (188.40.119.115): icmp_seq=1 ttl=64 time=0.026 ms
--- lb.bocek.ch ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.026/0.026/0.026/0.000 ms
draft@gserver:~$ ping lb.bocek.ch
PING lb.bocek.ch (152.96.80.48) 56(84) bytes of data.
64 bytes from srifs05.ost.ch (152.96.80.48): icmp_seq=1 ttl=53 time=23.1 ms
```

Load Balancing Algorithms

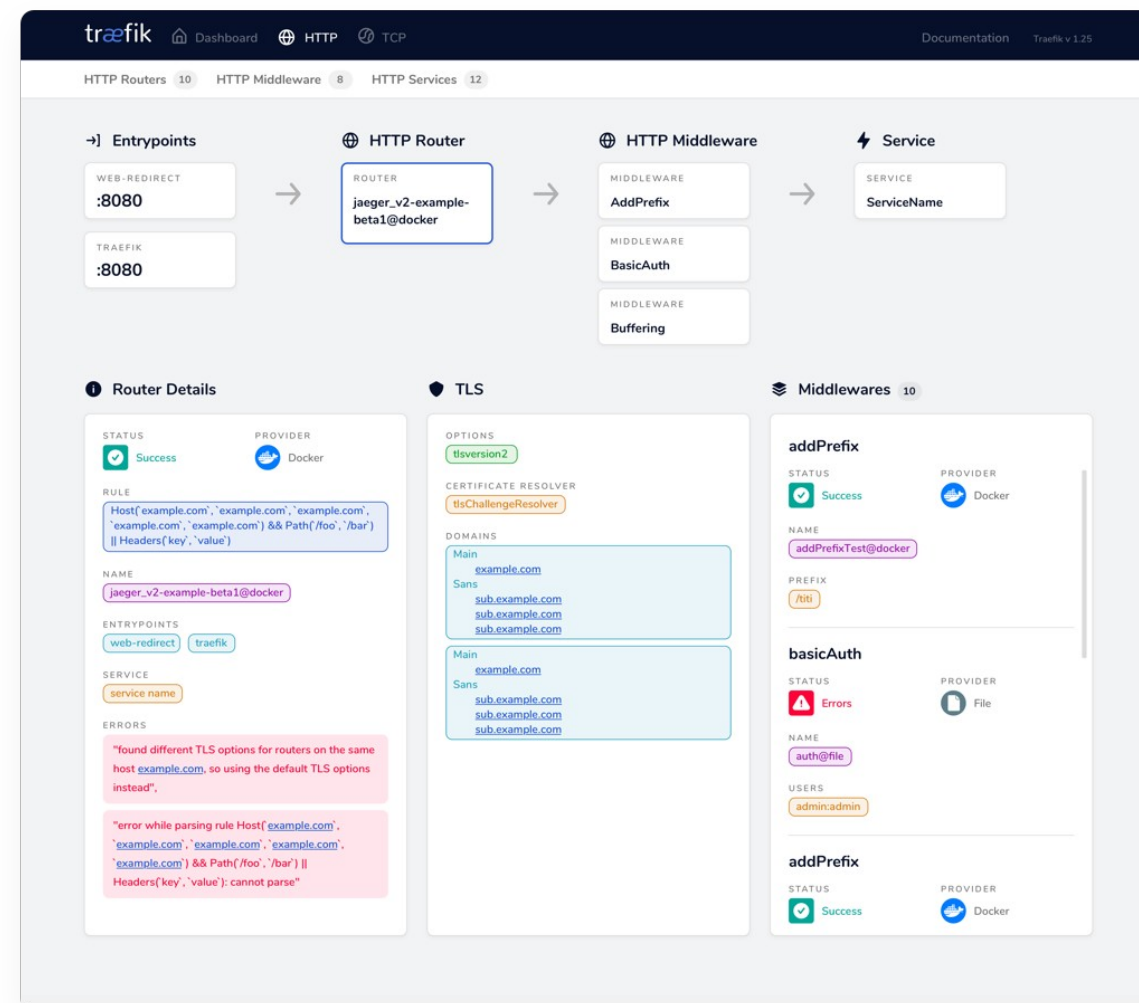
- Load Balancing Algorithms (**visualized**)
 - Round robin – loop sequentially
 - Simple algorithm, often default
 - But may drop requests on congested nodes
 - Weighted round robin – some server are more powerful
 - You can put weighted in from of everything
 - More powerful machines gets more work
 - But high variance in server load may drop requests
 - Least connections – fewest current connections to clients
 - Keep track of outstanding requests
 - Send work to the one with the least outstanding requests
 - But not the best for latency
- Peak exponentially weighted moving average
 - Considers latency
 - Complexity increases
- Others e.g., : ip_hash, least_time, random (**nginx**), uri_hash, cookie (**caddy**)

Load Balancing Algorithms

- Easiest: round-robin / random
 - Make sure your services are stateless!
- Stateless ~ don't store anything in the service
 - If you do, you need a stick session (try to avoid this) - same user to same service
 - Eg., cookie, ip_hash – send to same machine
- Health checks: tell your load balancer if you are running low on resources
 - Active: send active probes, e.g., every 3s
 - OOB – out of band (API to check health), e.g., necessary with DB, as connection may be OK, but database not
- Passive: only check with request
- Inline within service
- Different behavior:
 - Nginx: passive, caches request, so if an upstream fails, it uses another.
 - Caddy: passive, does not cache, but marks upstream as failed for the next request.
- L7 load balancing is more resource-intensive than packet-based L4
 - Terminates TLS and HTTP

Traefik

- Open Source, software-based load balancer:
<https://github.com/traefik/traefik>
- “The Cloud Native Edge Router”
- L4/L7 load balancer
- Golang, single binary
- Authentication
- Experimental HTTP/3 support
- Dashboard
- Official [traefik](#) docker image





Traefik

- Run it: `./traefik`
- Now lets configure
- Redirect 8888 to access dashboard
- <http://127.0.0.1:8888/dashboard/>

```
[entryPoints.web  
address = ":80"
```

```
[api]  
dashboard = true  
  
[providers.file]  
filename =  
"dynamic_load.toml"
```

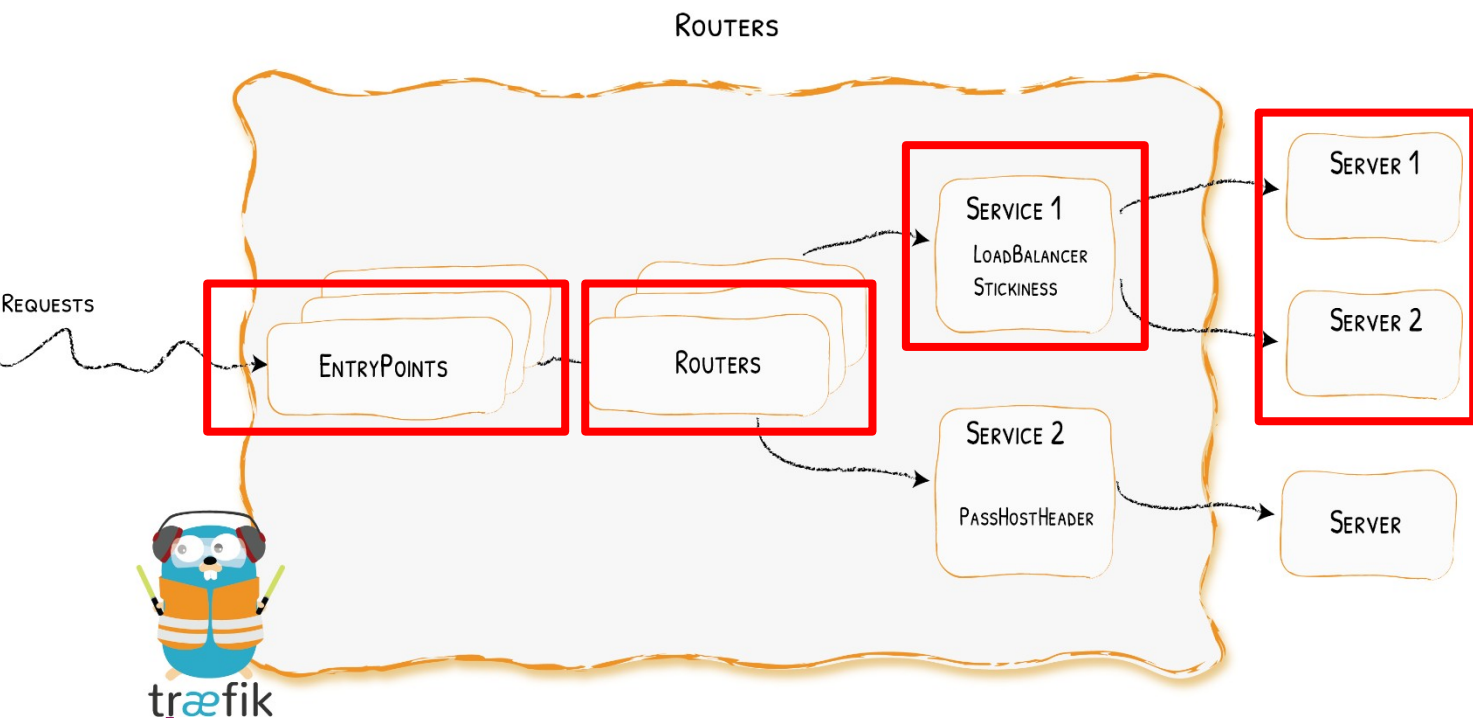
```
[log]  
#filePath = "traefik.log"  
level = "INFO"
```

```
[accessLog]  
  
[http.routers.dashboard]  
rule = "PathPrefix(`/api`) ||  
PathPrefix(`/dashboard`)"  
entrypoints = ["web"]  
service = "api@internal"  
middlewares = ["auth"]
```

```
[http.middlewares.auth.basicAuth]  
users = ["test:  
$apr1$H6uskkkW$IgXLP6ewTrSuBkTrqE8wj/"]
```

```
[http.routers.coinservice]  
rule = "PathPrefix(`/`)"  
entrypoints = ["web"]  
service = "coinservice"
```

```
[[http.services.coinservice.loadBalancer.servers]]  
url = "http://127.0.0.1:8080"  
[[http.services.coinservice.loadBalancer.servers]]  
url = "http://127.0.0.1:8081"
```



Traefik

- Labels, configure inside docker-compose, no extra project / Dockerfile

```
traefik:
  image: traefik:latest
  command:
    - "--api.dashboard=true"
    - "--providers.docker=true"
    - "--providers.docker.exposedbydefault=false"
    - "--entrypoints.web.address=:80"
  ports:
    - "80:80"
    - "8080:8080" # Dashboard
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock:ro
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.dashboard.rule=PathPrefix(`/dashboard`) || PathPrefix(`/api`)"
    - "traefik.http.routers.dashboard.service=api@internal"
    - "traefik.http.routers.dashboard.entrypoints=web"
```

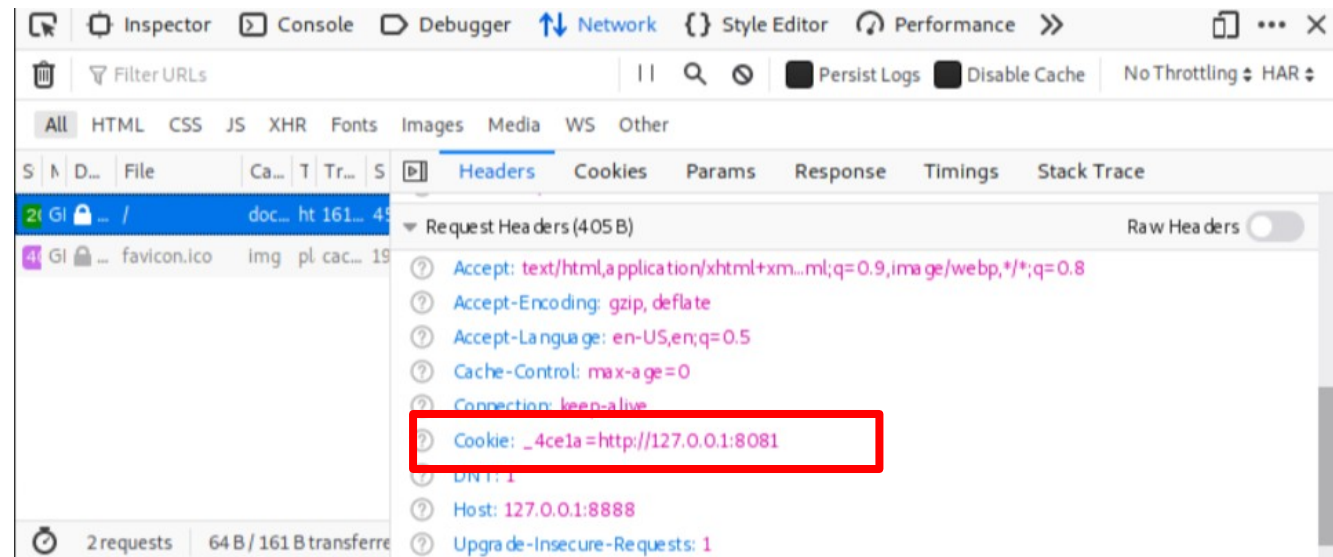
- Define 2 services
- Or 1 service and run:
docker-compose up --scale go-service=5

Service

- As a start, stateful service
 - Golang
- Stickiness with cookies
- Let's add a health check
- Weighted round robin
 - load balance between services and not between servers (example)

```
[http.services.coinservice.loadBalancer.healthCheck]  
path = "/health"  
interval = "3s"  
timeout = "1s"
```

```
[http.services.coinservice.loadBalancer.sticky.cookie]
```





Caddy

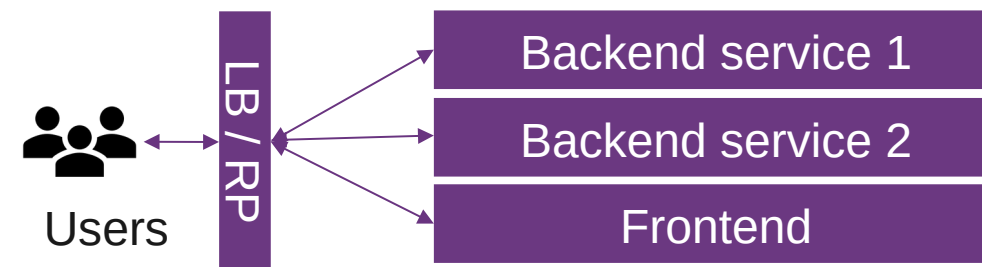
- Configuration: dynamic
 - Static: Caddyfile
- One-liners:
 - Quick, local file server: `caddy file-server`
 - Reverse proxy: `caddy reverse-proxy --from example.com --to localhost:9000`

```
:7070
reverse_proxy 127.0.0.1:8081 127.0.0.1:8080 {
    unhealthy_status 5xx
    fail_duration 5s
}
```

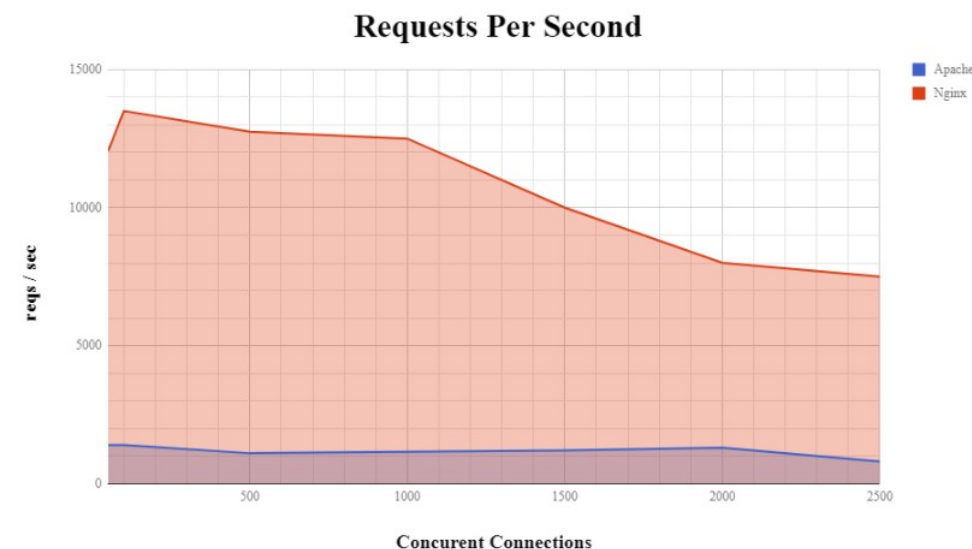
- Open Source, software-based load balancer: <https://github.com/caddyserver/caddy>
 - “Caddy 2 is a powerful, enterprise-ready, open source web server with automatic HTTPS written in Go”
 - L7 load balancer
 - Reverse proxy
 - Static file server
 - HTTP/1.1, HTTP/2, and experimental HTTP/3
 - Caddy on [docker hub](#)

NGINX

- Free + commercial version
 - Fast webserver, ~35% market share
 - Acquired by F5 Networks (slide 7) in 2019
 - HTTP proxy, Mail proxy, reverse proxy, load balancer
 - Reverse proxy vs. load balancer
 - No active health checks, no sticky sessions (not usable in prod env) [source]
- Performance tuning – some ideas



- Benchmarks, benchmarks





NGINX

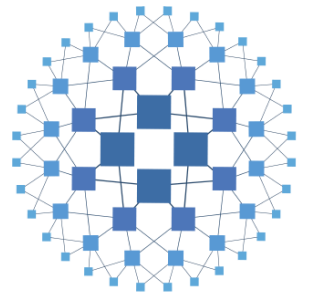
- Add configuration
- Health check
 - Inband/passive (active - commercial)
- Default: round robin
 - Least connected (least_conn)
 - Sticky (ip_hash), cookie (commercial)
 - Weighted balancing (weight=1)

```
#!/tmp/nginx.conf

events {
    worker_connections 1024;
}

http {
    upstream coinservice {
        #least_conn;
        server 127.0.0.1:8080 weight=1;
        server 127.0.0.1:8081;
    }

    server {
        listen 7070 default_server;
        listen [::]:7070 default_server;
        location / {
            proxy_pass http://coinservice;
        }
        # You may need this to prevent return 404
        recursion.
        location = /404.html {
            internal;
        }
    }
}
```



HAPROXY

HAproxy

- L4 and L7 load balancer and reverse proxy
 - **Open source** option: commercial support (HAProxy Technologies)
 - Widely used: stack overflow, github, ...
- Performance: fast, small Atom server in **2011** ~2300 SSL TPS
 - **2017**: tuned to 2.3m SSL connections (32cores/64GB RAM)
- Install: apk add haproxy
- Configure and run: /etc/init.d/haproxy start
 - Algorithms: roundrobin, leastconn, source
 - Sticky session: appsession
 - check → health checks (inband)
- Primary/secondary

- app1 by default, 3 checks at 10s interval fail, app2 will be used:

```
balance roundrobin
server app1 127.0.0.1:8080 check inter 10s fall 3
server app2 127.0.0.1:8081 check backup
```

```
#!/etc/haproxy/haproxy.cfg
defaults
    retries 3
    timeout client 30s
    timeout connect 4s
    timeout server 30s

frontend www
    bind :80
    mode http
    default_backend coinservice

backend coinservice
    mode http
    balance roundrobin
    server app1 127.0.0.1:8080 check
    server app2 127.0.0.1:8081 check
```

Dockerfile / docker compose

- Example: caddy as LB, go as Service
 - docker-compose up --scale services=5
 - Nginx requires more configuration

```
#docker-compose.yml
version: '3'
services:
  services:
    build: .
    ports:
      - "8080-8085:8080"
  lb:
    image: caddy
    ports:
      - "7070:7070"
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile
```

```
#Caddyfile
:7070
reverse_proxy * {
  to http://dsy-services-1:8080
  to http://dsy-services-2:8080
  to http://dsy-services-3:8080
  to http://dsy-services-4:8080
  to http://dsy-services-5:8080

  lb_policy round_robin
  lb_try_duration 1s
  lb_try_interval 100ms
  fail_duration 10s
  unhealthy_latency 1s
}
```

Dockerfile / docker compose

```
#docker-compose.yml
version: '3'
services:
  serviceX:
    build: .
    ports:
      - 9090:8080
  lb:
    image: caddy
    ports:
      - 7070:7070
```

- Docker compose use its own DNS
 - Within docker compose it knows the names: "serviceX", "lb" → ping
 - serviceX if in the same network
 - Different networks – no connection

- External port
 - Needs mapping to reach component from outside → reachable via localhost
- Internal port, can be reached in the same network – use DNS, localhost does not work

```
#docker-compose.yml
version: '3'
services:
  serviceX:
    build: .
    ports:
      - "9090:8080"
    networks:
      - backend
      - database

networks:
  frontend:
  backend:
  database:
```