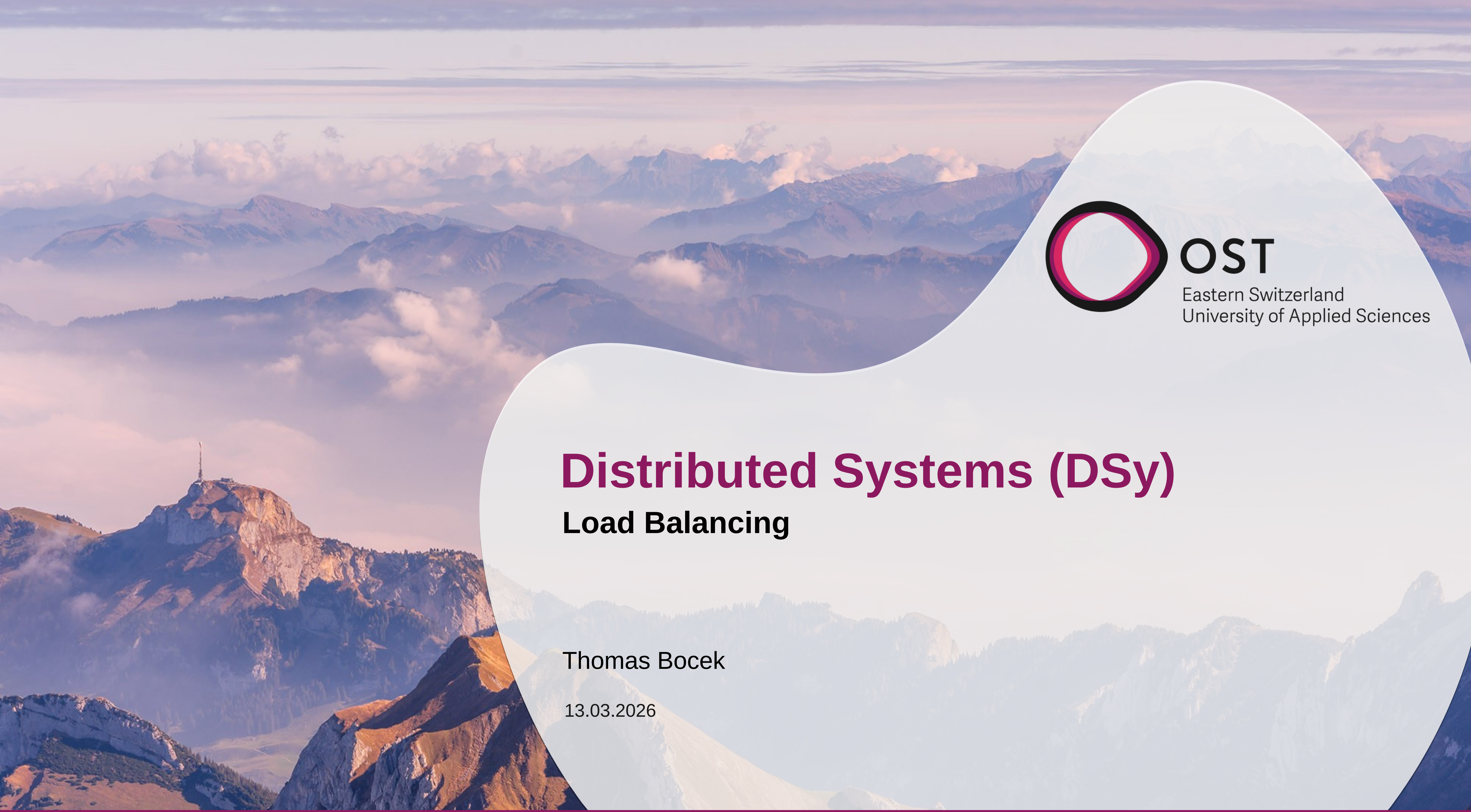


OST
Eastern Switzerland
University of Applied Sciences

Distributed Systems (DSy)

Load Balancing

Thomas Bocek

13.03.2026

Learning Goals

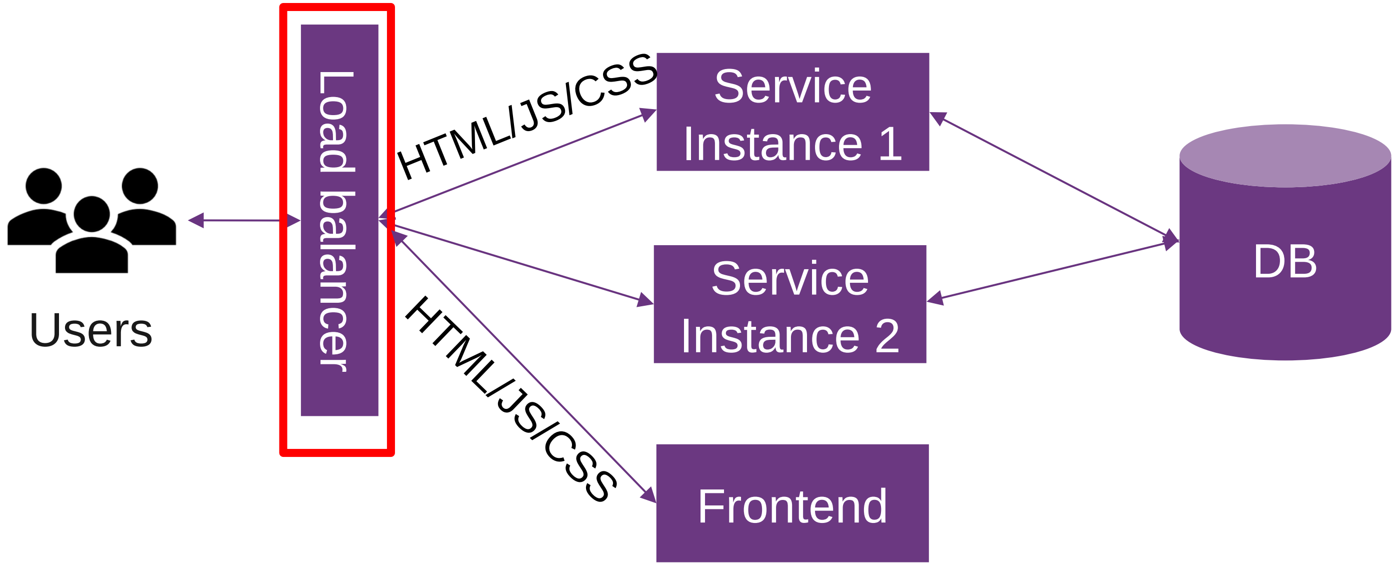
- Lecture 5 (Load Balancing)
  - What types of LB exists?
  - Which one to pick?
  - How can a LB be used for the challenge task?

Load Balancing

- Challenge Task Requirement

  1) Load balancing with scalable service

  2) Failover of a service instance

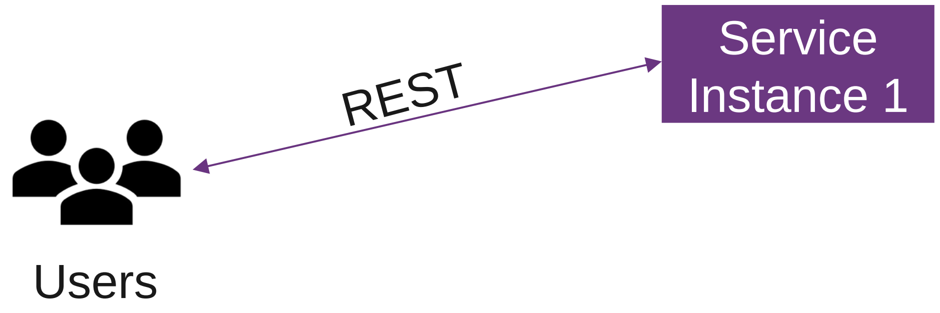


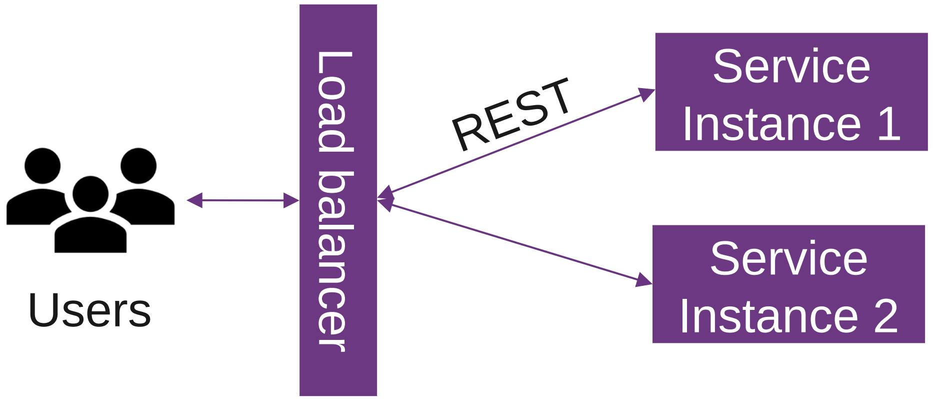
Load Balancing

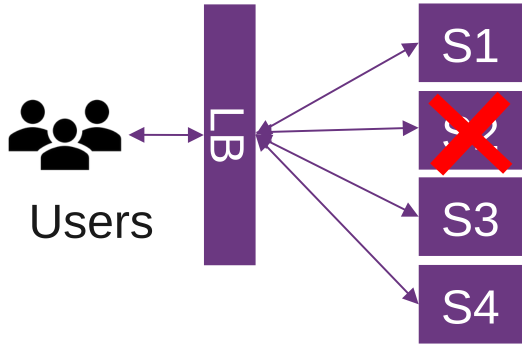
- What is load balancing
  - Distribution of workloads across multiple computing resources
    - Workloads (requests)
    - Computing resources (machines)
  - Distributes client requests or network load efficiently across multiple servers [link]
    - E.g., service get popular, high load on service

→ 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

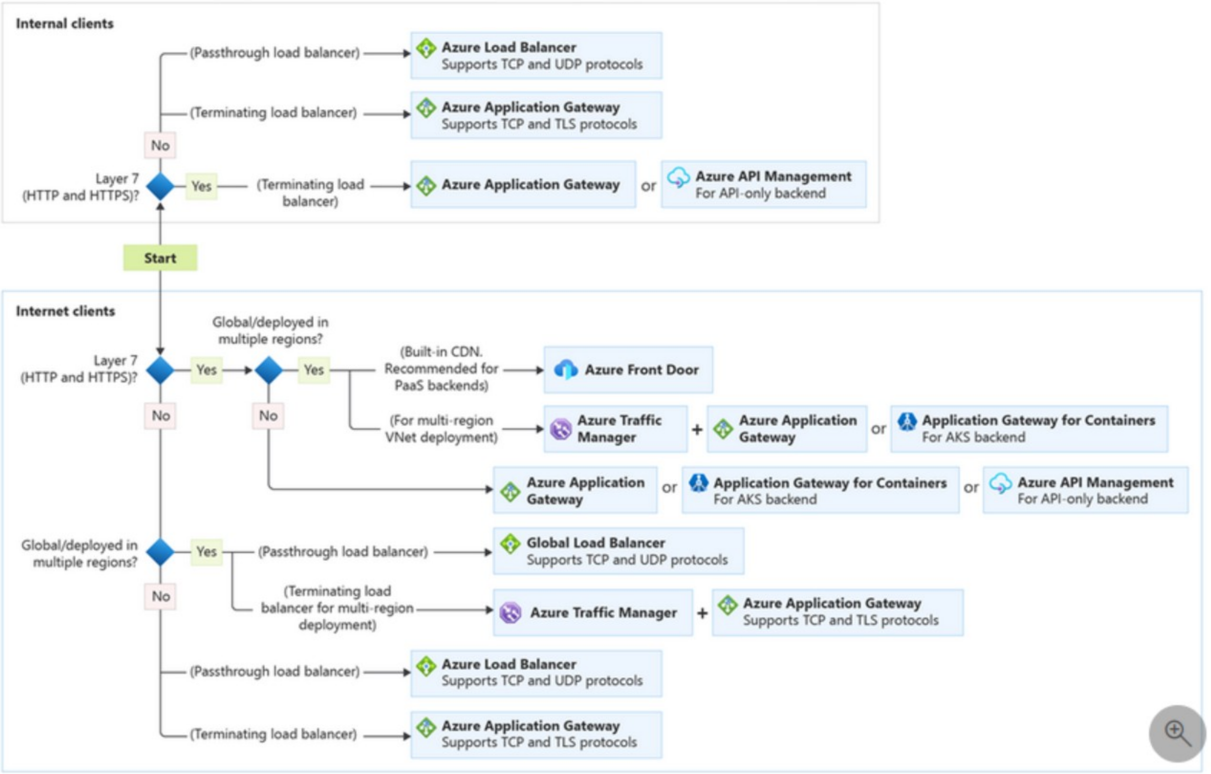
- Hardware load balancer
  - HW-LB use proprietary software, which often uses specialized processors
    - Less generic, more performance
    - Some use open-source SW, e.g., HAProxy
  - E.g., loadbalancer.org, F5, Cisco, Kemp
  - Only if you control your datacenter

- Software load balancer
  - L2/L3: MetalLB (Kubernetes bare-metal, standard solution for type: LoadBalancer on bare-metal/kind clusters without a cloud LB), Seesaw (not widely used)
  - L4: HAProxy (desc), Nginx, Gobetween, Traefik
  - L7: Envoy (C++), HAProxy (C), Nginx (C), caddy (golang), Gobetween (golang), Eureka (Java) – services register at Eureka, Traefik (golang)
- SW vs. SW / SW vs. HW
  - strong opinions, funny opinions, 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, make sure benchmarks are reproducible: benchmark

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:


Internal clients
(Passthrough load balancer)
Azure Load Balancer
Supports TCP and UDP protocols
(Terminating load balancer)
Azure Application Gateway
Supports TCP and TLS protocols
No
Layer 7
(HTTP and HTTPS)?
Yes
(Terminating load balancer)
Azure Application Gateway
or
Azure API Management
For API-only backend
Start
Internet clients
Layer 7
(HTTP and HTTPS)?
Yes
Global/deployed in multiple regions?
Yes
(Built-in CDN. Recommended for PaaS backends)
Azure Front Door
No
No
(For multi-region VNet deployment)
Azure Traffic Manager
+
Azure Application Gateway
or
Application Gateway for Containers
For AKS backend
Azure Application Gateway
or
Application Gateway for Containers
For AKS backend
or
Azure API Management
For API-only backend
Global/deployed in multiple regions?
Yes
(Passthrough load balancer)
Global Load Balancer
Supports TCP and UDP protocols
No
(Terminating load balancer for multi-region deployment)
Azure Traffic Manager
+
Azure Application Gateway
Supports TCP and TLS protocols
(Passthrough load balancer)
Azure Load Balancer
Supports TCP and UDP protocols
(Terminating load balancer)
Azure Application Gateway
Supports TCP and TLS protocols

https://docs.microsoft.com/en-us/azure/architecture/guide/technology-choices/load-balancing-overview

Software-based load balancing

- L7 vs L4 Load Balancing
  - L7 more resource-intensive than L4, L7 terminates TLS and HTTP (encryption overhead)
- DNS Load Balancing (Layer 7), round-robin DNS
  - Easy to setup, client-side selection, dig lb.bocek.ch
  - Drawback: negative caching impact
  - Used in Bitcoin Core: dig dnsseed.emzy.de
- DNS Load Balancing (Layer 7): split horizon DNS
  - Different DNS information, depending on source of DNS request
- 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"
```

```
--- 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

- 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

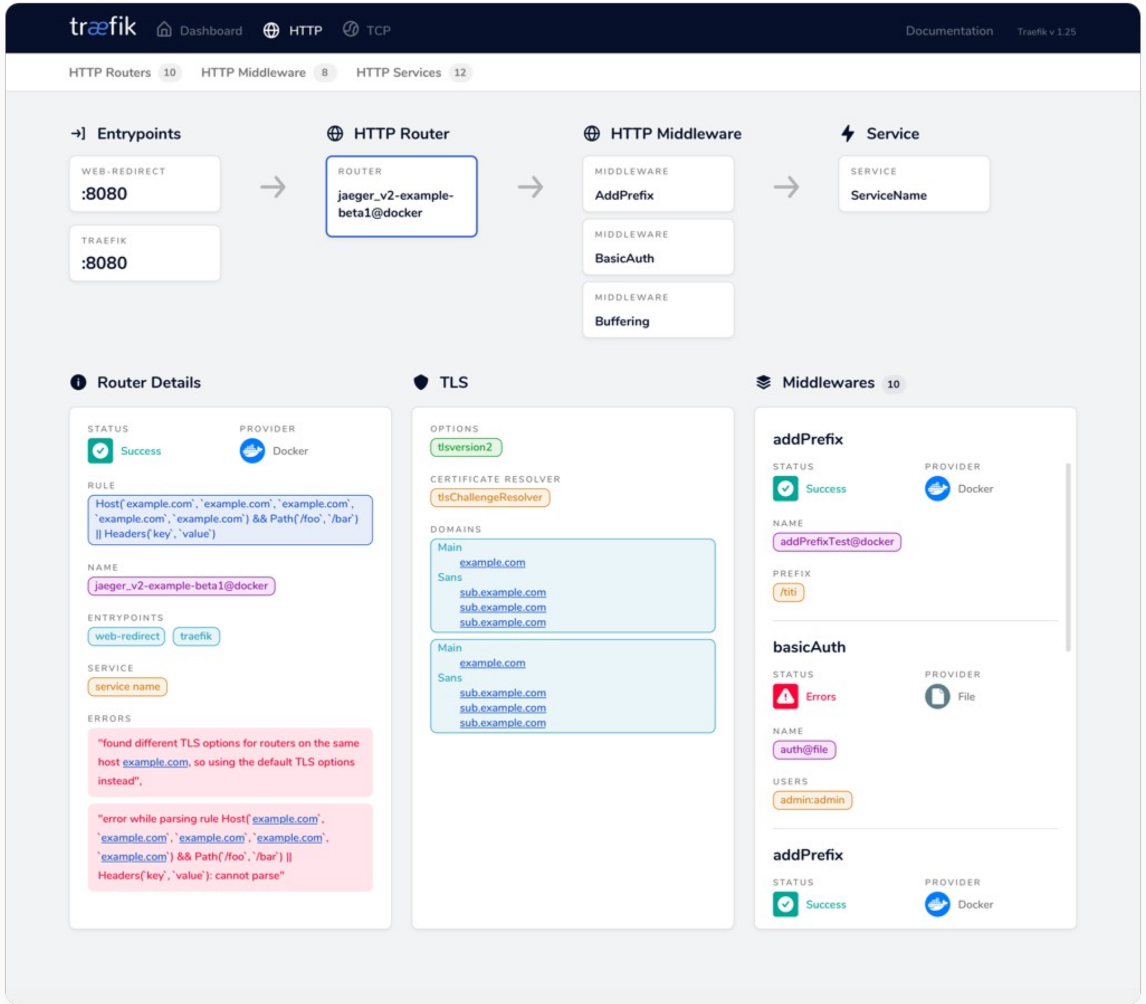
- Passive: only check with request
- Inline within service
- Different behavior:
  - Nginx: passive, caches request, so if an upstream fails, it uses another.
  - Caddy: passive, does not cache, but marks upstream as failed for the next request.

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, uri_hash, cookie (e.g., caddy, nginx)

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
  - HTTP/3 support
- Dashboard
- Official traefik docker image
- Example



WEB-REDIRECT :8080

TRAEFIK :8080

ROUTER jaeger_v2-example-beta1@docker

MIDDLEWARE AddPrefix

MIDDLEWARE BasicAuth

MIDDLEWARE Buffering

SERVICE ServiceName

STATUS Success

PROVIDER Docker

RULE Host(`example.com`, `example.com`, `example.com`, `example.com`, `example.com`) && Path(`/foo`, `/bar`) || Headers(`key`, `value`)

NAME jaeger_v2-example-beta1@docker

ENTRYPOINTS web-redirect traefik

SERVICE service name

ERRORS

"found different TLS options for routers on the same host example.com, so using the default TLS options instead",

"error while parsing rule Host(`example.com`, `example.com`, `example.com`, `example.com`, `example.com`) && Path(`/foo`, `/bar`) || Headers(`key`, `value`): cannot parse"

OPTIONS tlsversion2

CERTIFICATE RESOLVER tlsChallengeResolver

DOMAINS

Main
example.com
Sans
sub.example.com
sub.example.com
sub.example.com

Main
example.com
Sans
sub.example.com
sub.example.com
sub.example.com

addPrefix

STATUS Success

PROVIDER Docker

NAME addPrefixTest@docker

PREFIX /titi

basicAuth

STATUS Errors

PROVIDER File

NAME auth@file

USERS admin:admin

addPrefix

STATUS Success

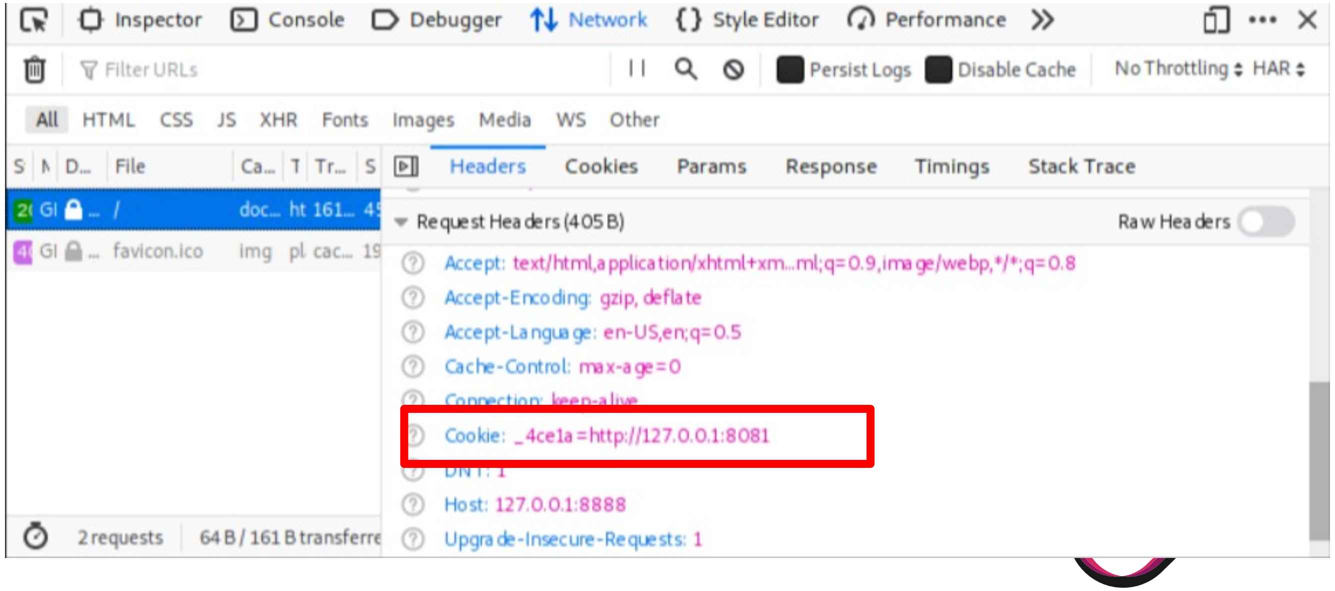
PROVIDER Docker

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]
```



Request Headers (405 B)

```
Accept: text/html,application/xhtml+xm...ml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Cache-Control: max-age=0
Connection: keep-alive
Cookie: _4ce1a=http://127.0.0.1:8081
DNT: 1
Host: 127.0.0.1:8888
Upgrade-Insecure-Requests: 1
```

Caddy

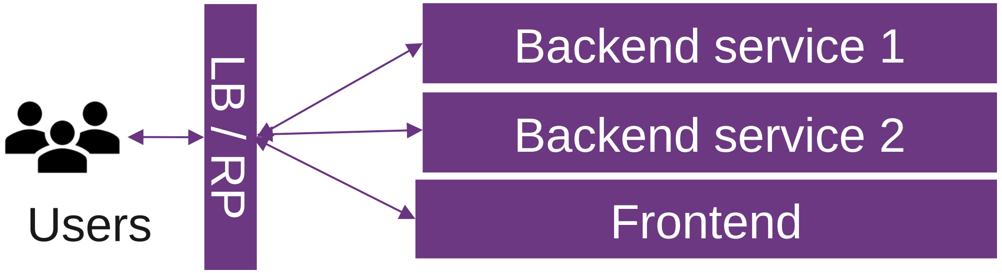
- Configuration: dynamic
  - Static: Caddyfile
- One-liners:
  - Quick, local file server: caddy file-server
  - Reverse proxy: caddy reverse-proxy --from example.com --to localhost:9000

```
:80 {
  reverse_proxy /* {
    to example-lb-go-service-1:8080
  }
}
```

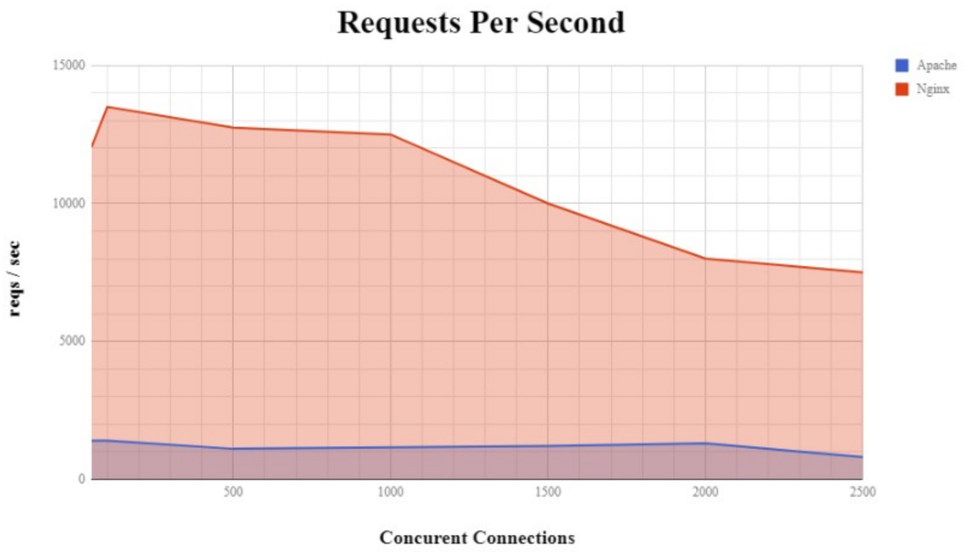
- Open Source, software-based load balancer: https://github.com/caddyserver/caddy
  - “Caddy 2 is a powerful, enterprise-ready, open source web server with automatic HTTPS written in Go”
  - L7 load balancer
  - Reverse proxy
  - Static file server
  - HTTP/1.1, HTTP/2, HTTP/3
  - Caddy on docker hub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

NGINX

- Free + commercial version
  - Fast webserver, ~35% market share
  - Acquired by F5 Networks (slide 5) 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
- Forks: Freenginx (2024, by core dev, dispute with F5), Angie (by former nginx devs, drop-in replacement)



- Benchmarks, benchmarks



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)
- Configure and run: /etc/init.d/haproxy start
  - Algorithms: roundrobin, leastconn, source
  - Sticky session: cookie
  - check → health checks (passive / active)

HAPROXY

- Primary/secondary: backup server only receives traffic when all primary servers fail
- Dynamic backend discovery via server-template + DNS resolvers
- Built-in stats dashboard
- Rate limiting and connection throttling built-in