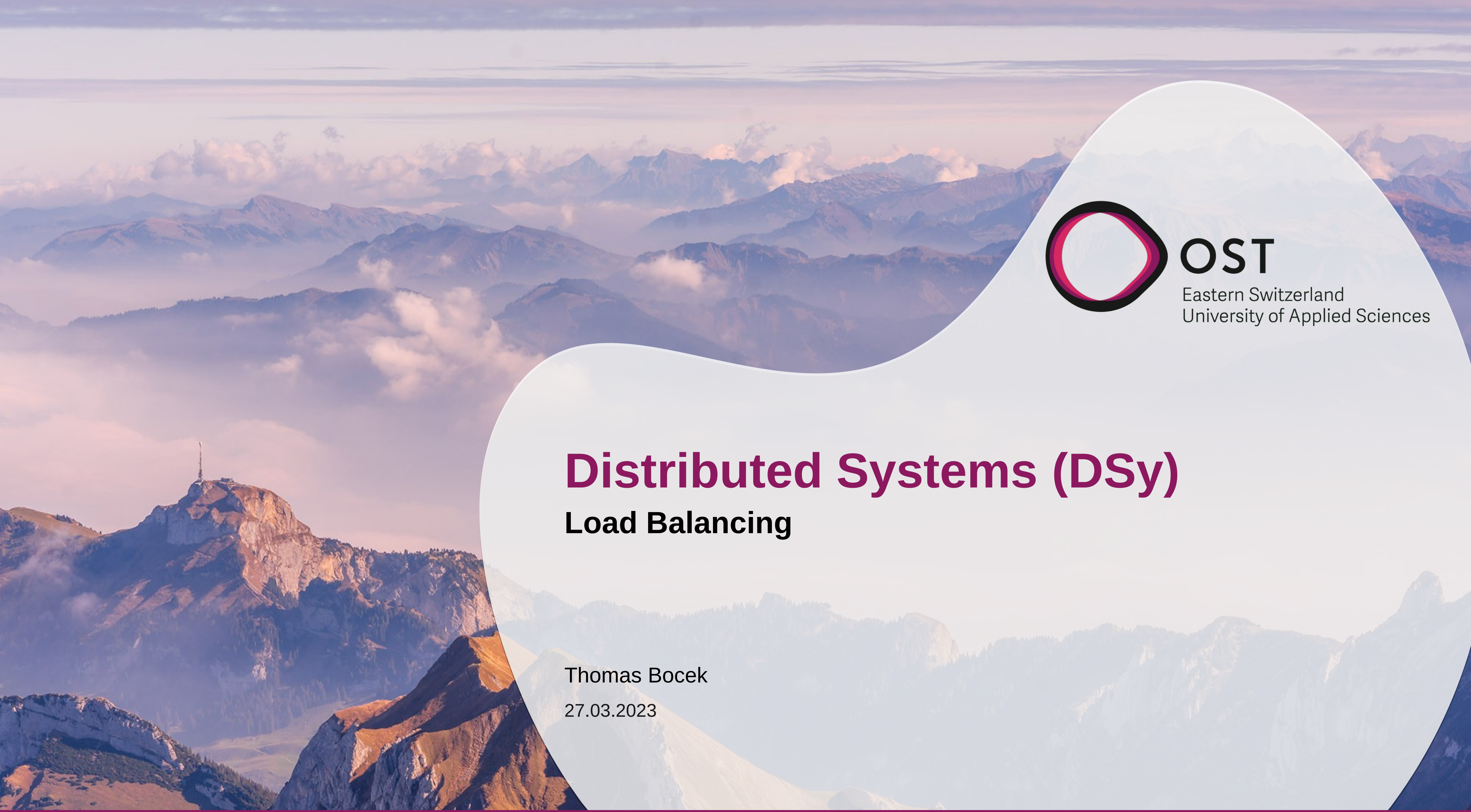


OST
Eastern Switzerland
University of Applied Sciences

Distributed Systems (DSy)

Load Balancing

Thomas Bocek

27.03.2023

Learning Goals

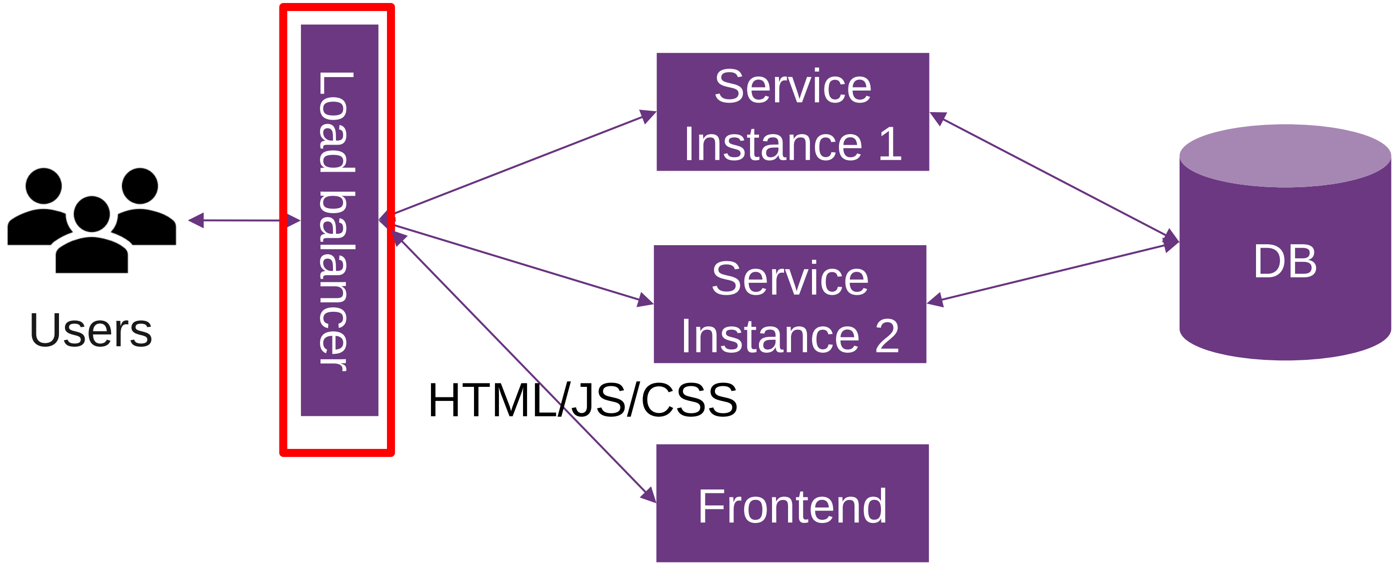
- Lecture 6 (Load Balancing)
  - What types of LB exists?
  - Which one to pick?
  - How can a LB be used for the challenge task?

Load Balancing

- Challenge Task Requirement

  1) Load balancing with scalable service

  2) Failover of a service instance

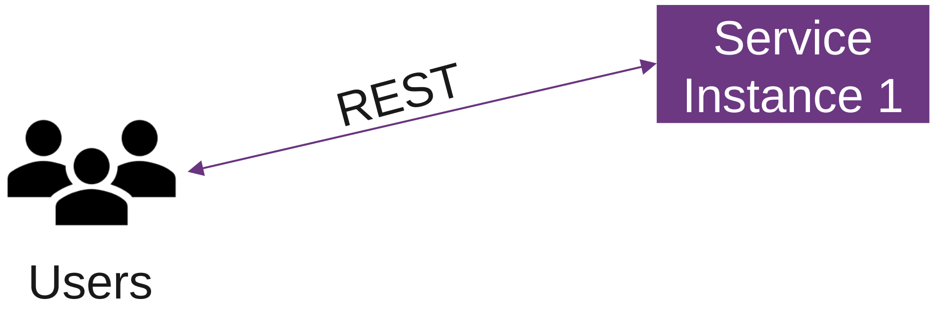


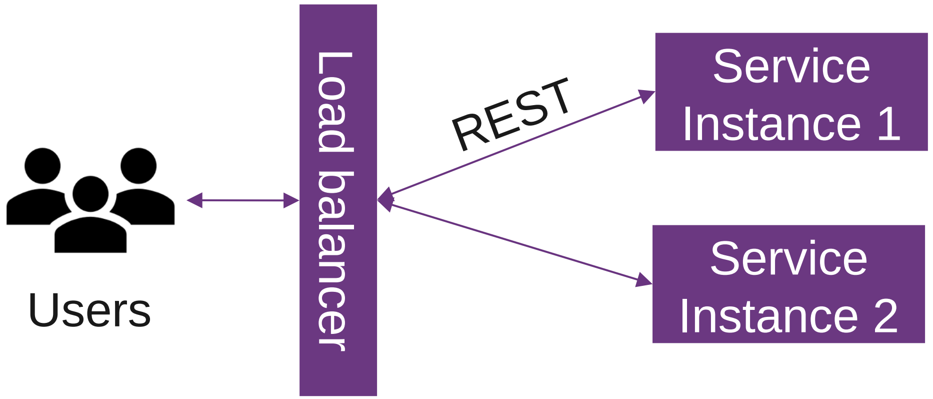
Load Balancing

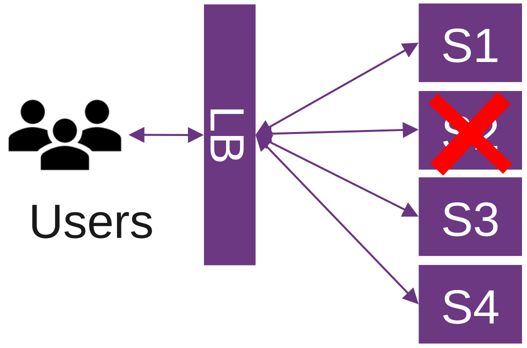
- What is load balancing
  - Distribution of workloads across multiple computing resources
    - Workloads (requests)
    - Computing resources (machines)
  - Distributes client requests or network load efficiently across multiple servers [link]
    - E.g., service get popular, high load on service

→ 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
  - Only if you control your datacenter

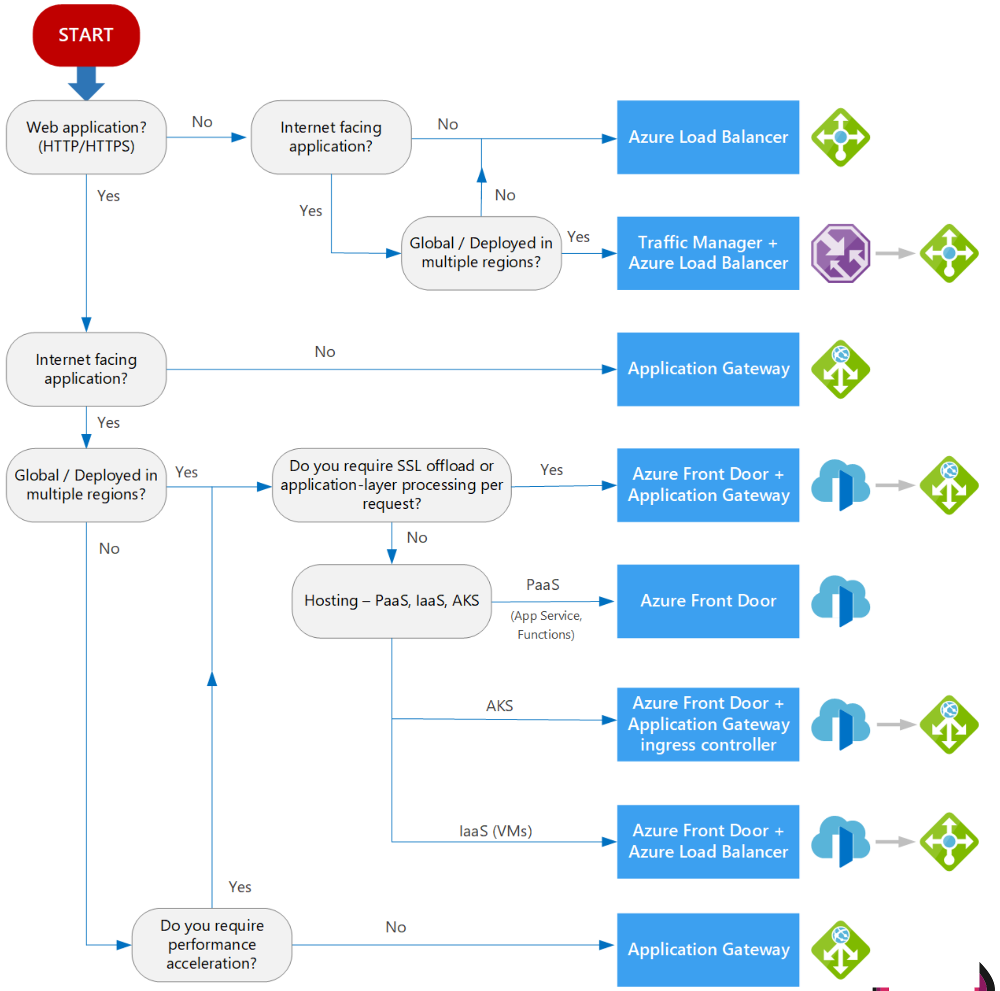
https://www.loadbalancer.org/products/hardware/

- 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:



START

Web application? (HTTP/HTTPS)

No

Internet facing application?

No

Azure Load Balancer

No

Yes

Yes

Global / Deployed in multiple regions?

Yes

Traffic Manager + Azure Load Balancer

Internet facing application?

No

Application Gateway

Yes

Global / Deployed in multiple regions?

Yes

Do you require SSL offload or application-layer processing per request?

Yes

Azure Front Door + Application Gateway

No

No

Hosting – PaaS, IaaS, AKS

PaaS

(App Service, Functions)

Azure Front Door

AKS

Azure Front Door + Application Gateway ingress controller

IaaS (VMs)

Azure Front Door + Azure Load Balancer

Yes

Do you require performance acceleration?

No

Application Gateway

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

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
    - Your ISP, you if you do recursive DNS
    - But 1.1.1.1, 4.4.4.4, 8.8.8.8
  - 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
- Reduced Downtime, Scalable, Redundancy
  - Client can decide what to do
  - Negative caching impact!
  - Used in bitcoin: dig dnsseed.emzy.de

```
$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
bootstrap       A               188.40.119.115
bootstrap       A               152.96.80.48
$INCLUDE "/etc/opendkim/keys/mail.txt“
$INCLUDE "/etc/bind/dmarc.txt"
```

```
--- bootstrap.tomp2p.net 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 bootstrap.tomp2p.net
PING bootstrap.tomp2p.net (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
--- bootstrap.tomp2p.net 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 bootstrap.tomp2p.net
PING bootstrap.tomp2p.net (152.96.80.48) 56(84) bytes of data.
64 bytes from dsl.hsr.ch (152.96.80.48): icmp_seq=1 ttl=53 time=23.1 ms
```

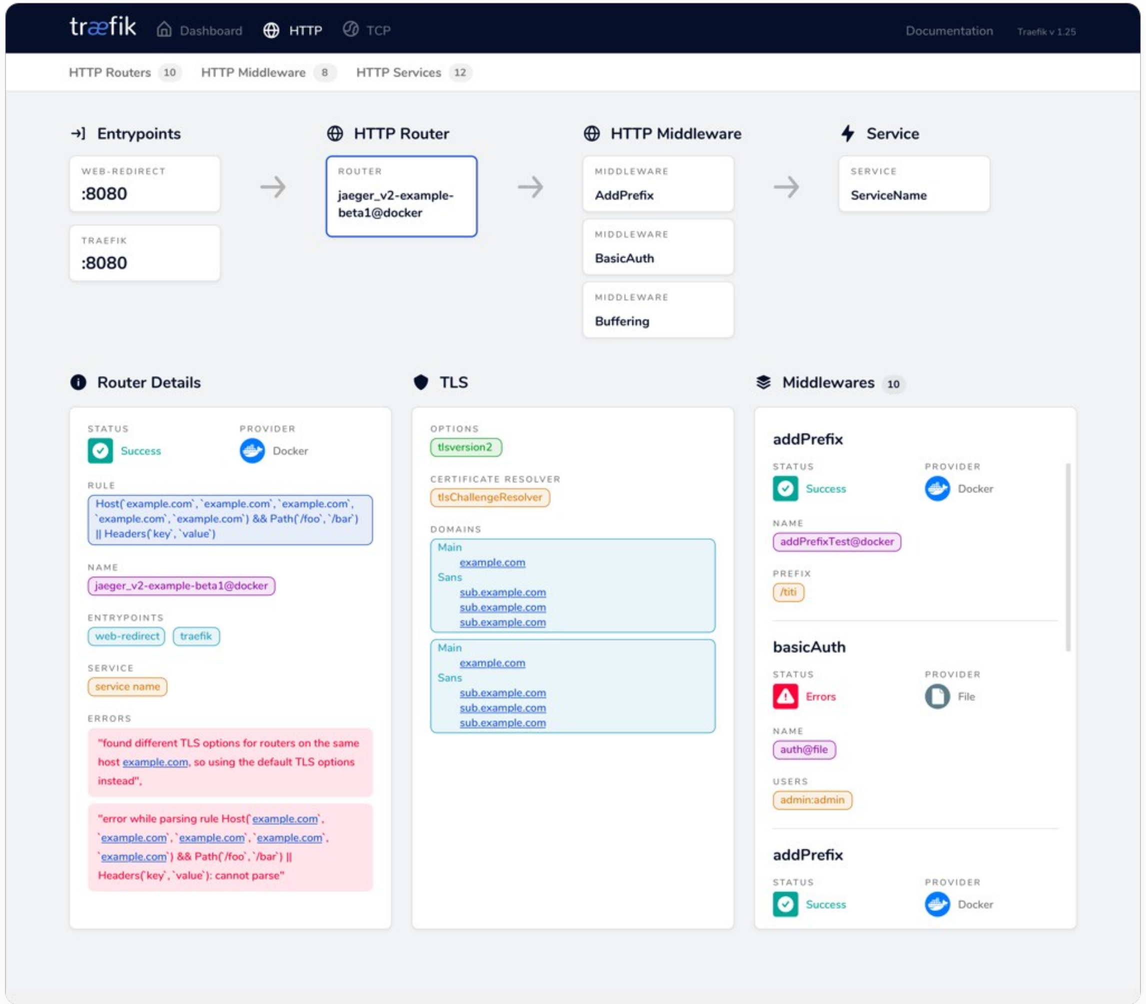
Load balancing L4/L7

- Load Balancing Algorithms
  - Round robin – loop sequentially
  - Weighted round robin – some server are more powerful
    - You can put weighted in from of everything
  - Least connections – fewest current connections to clients
  - Least time – combination of fastest response time and fewest active connections
  - Least pending requests – fewest number of active sessions
  - Agent-based – service reports on it load
  - Hash – distributes requests based on a key you define (e.g., source) – can be static / sticky
  - Random – flip a coin

- Easiest: round-robin
  - 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
- Health checks: tell your load balancer if you are running low on resources
  - Inline within service
  - OOB – out of band (API to check health), e.g., necessary with DB, as connection may be OK, but database not
- 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



Entrypoints

WEB-REDIRECT :8080

TRAEFIK :8080

HTTP Router

ROUTER jaeger_v2-example-beta1@docker

HTTP Middleware

MIDDLEWARE AddPrefix

MIDDLEWARE BasicAuth

MIDDLEWARE Buffering

Service

SERVICE ServiceName

Router Details

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"

TLS

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

Middlewares 10

addPrefix

STATUS Success

PROVIDER Docker

NAME addPrefixTest@docker

PREFIX /titi

basicAuth

STATUS Errors

PROVIDER File

NAME auth@file

USERS admin:admin

addPrefix

STATUS Success

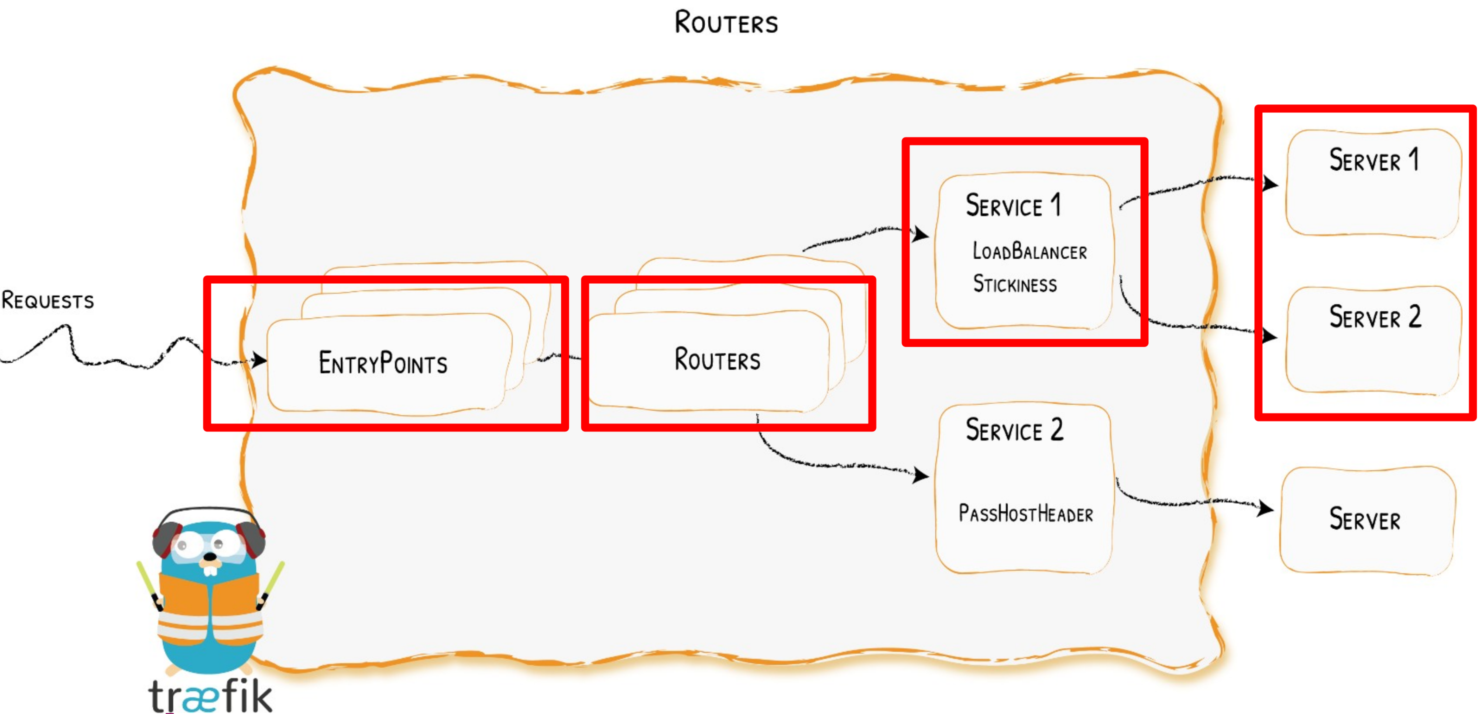
PROVIDER Docker

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"
```

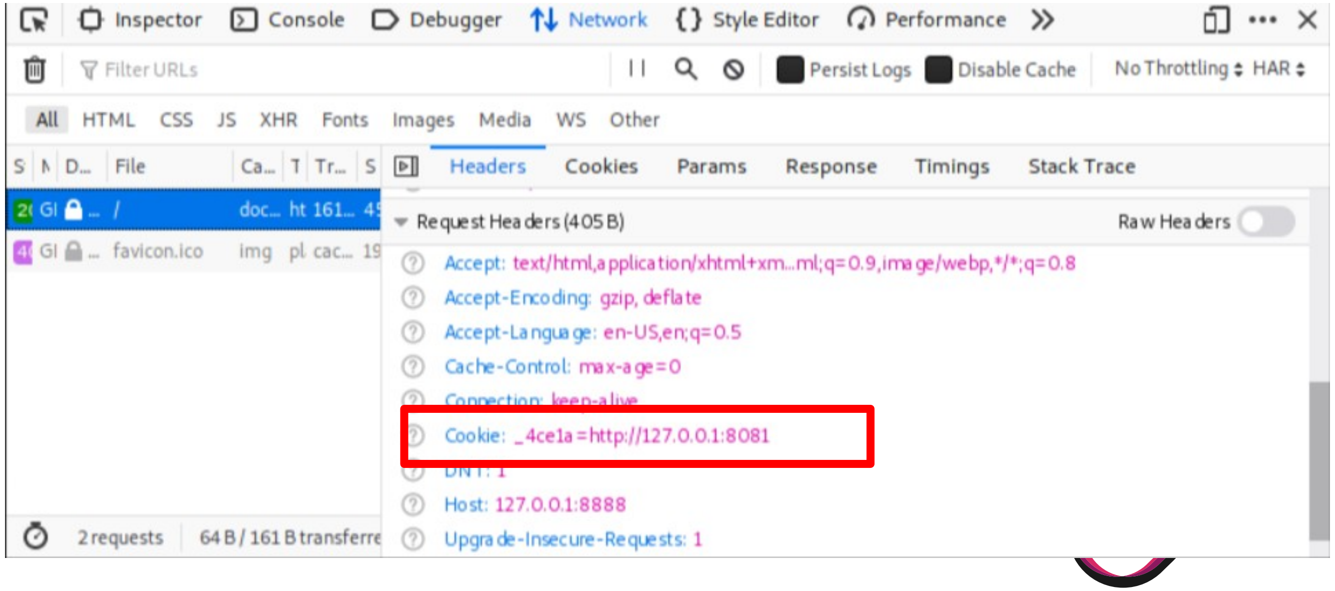


Service

- 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

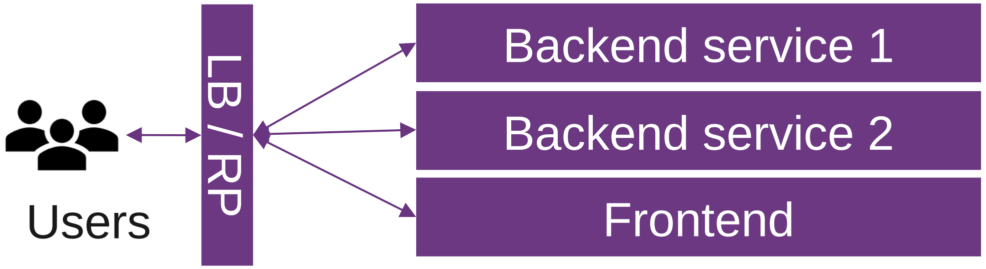
- Configuration: dynamic
  - Static: Caddyfile
- One-liners:
  - Quick, local file server: caddy file-server
  - Reverse proxy: caddy reverse-proxy --from example.com --to localhost:9000

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

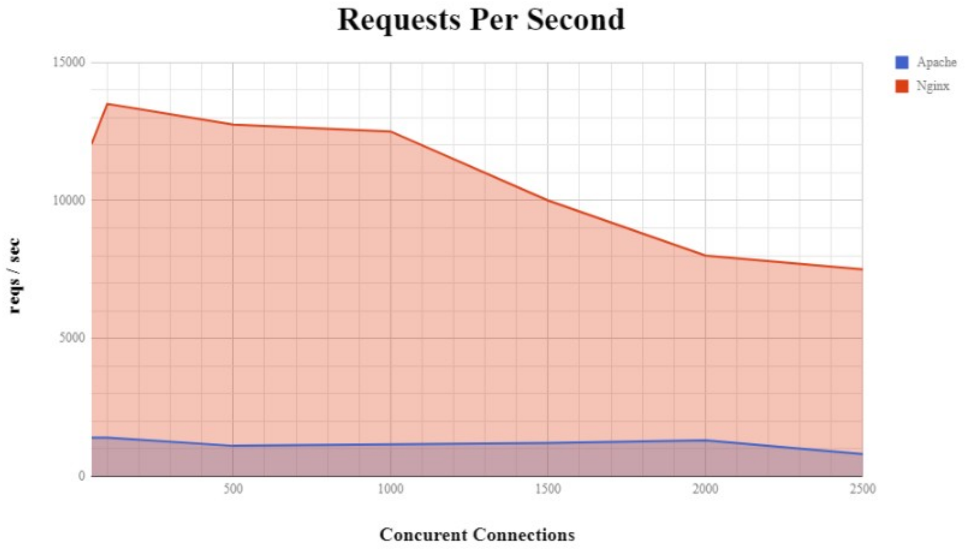
- Open Source, software-based load balancer: https://github.com/caddyserver/caddy
  - “Caddy 2 is a powerful, enterprise-ready, open source web server with automatic HTTPS written in Go”
  - L7 load balancer
  - Reverse proxy
  - Static file server
  - HTTP/1.1, HTTP/2, and experimental HTTP/3
  - Caddy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

- 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

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

```
#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
}
```

CORS

- CORS = Cross-Origin Resource Sharing
  - For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts (among others)
  - Mechanism to instruct browsers that runs a resource from origin A to run resources from origin B
- Solution
  - Use reverse proxy with builtin webserver, e.g., nginx, or user reverse proxy with external webserver.

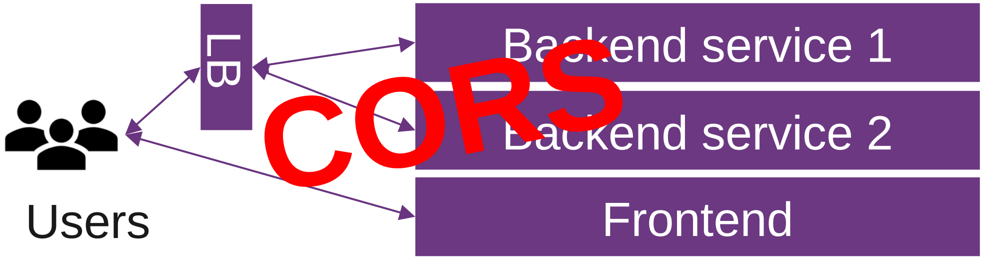
→ The client only sees the same origin for the API and the frontend assets

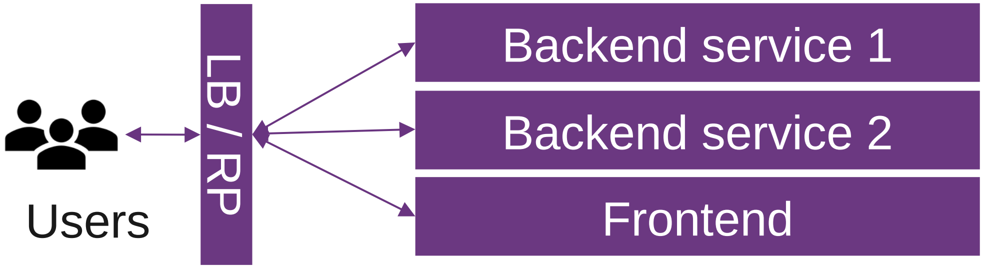
- Access-Control-Allow-Origin: https://foo.example

→ For dev: Access-Control-Allow-Origin: *

- w.Header().Set("Access-Control-Allow-Origin", "*")

- Reverse proxy


LB
Backend service 1
Backend service 2
Frontend
Users
CORS


LB / RP
Backend service 1
Backend service 2
Frontend
Users