FS20

DISTRIBUTED APPLICATIONS

Load balancing, git

T. Bocek
thomas.bocek@hsr.ch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Agenda

- **Distributed Systems in the News**
- **DNSSEC – from last lecture**
- **Load balancing**
- **Git**

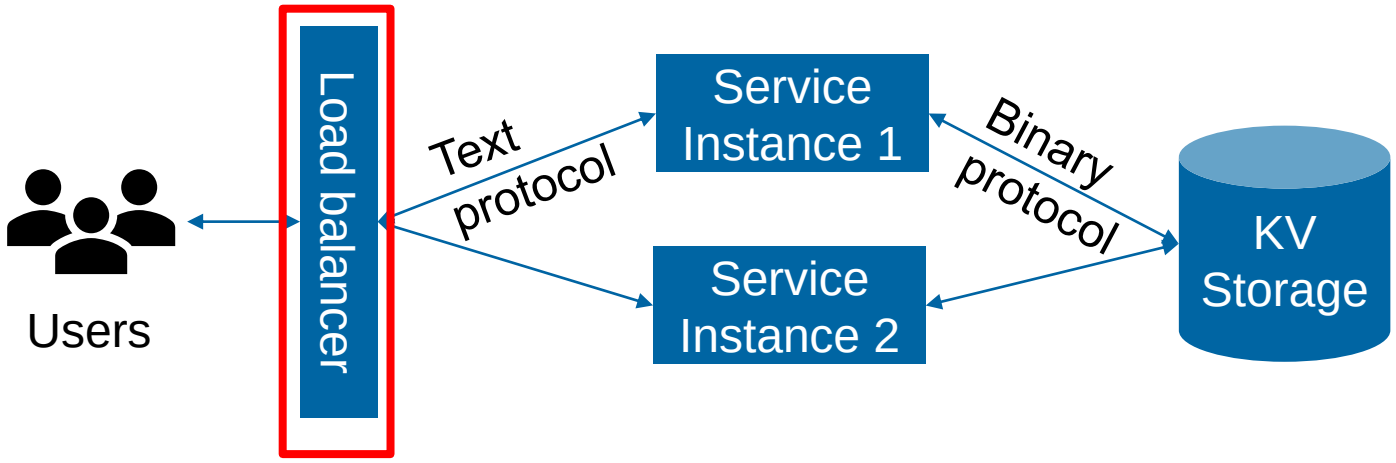
Distributed Systems - In the News

- **02.03.2020: Simple Systems Have Less Downtime**
  - Person responsible for the system leaves
    - Another takes over without much learning
  - “analytics dashboard built with Tableau is likely to have more qualified people to fix it than one built with a patchwork of custom scripts and APIs”
  - Troubleshooting takes less time
  - More alternative solutions
    - “…A Salesforce process that uses a mishmash of automation and third-party tools to score, filter, classify, and assign new sales leads. If that fails then there is no obvious replacement.”
  - Principles for Simple Systems
    - Features don’t justify complexity
    - Complex ideas lead to complex implementations
    - Modifications before additions
- **06.03.2020: Cloud Storage for $2 / TB / Mo**
  - Lecture 2: $5 / 1TB / month
  - 99.9999% with 3x overhead
  - 10 out of 30 scheme
  - Calculation online
  - Compared to cloud, shortcuts:
    - traditional model: ultra high reliability
    - Sia: counting milliseconds, not microseconds
    - Sia: designed to be optimal as a decentralized system

Last Lecture

- **DNSSEC, DoT, DoH**

Challenge Task FS 2020



- **Todays lecture is about load balancing**
  - Some students don't know git
  - Upcoming: many students use docker

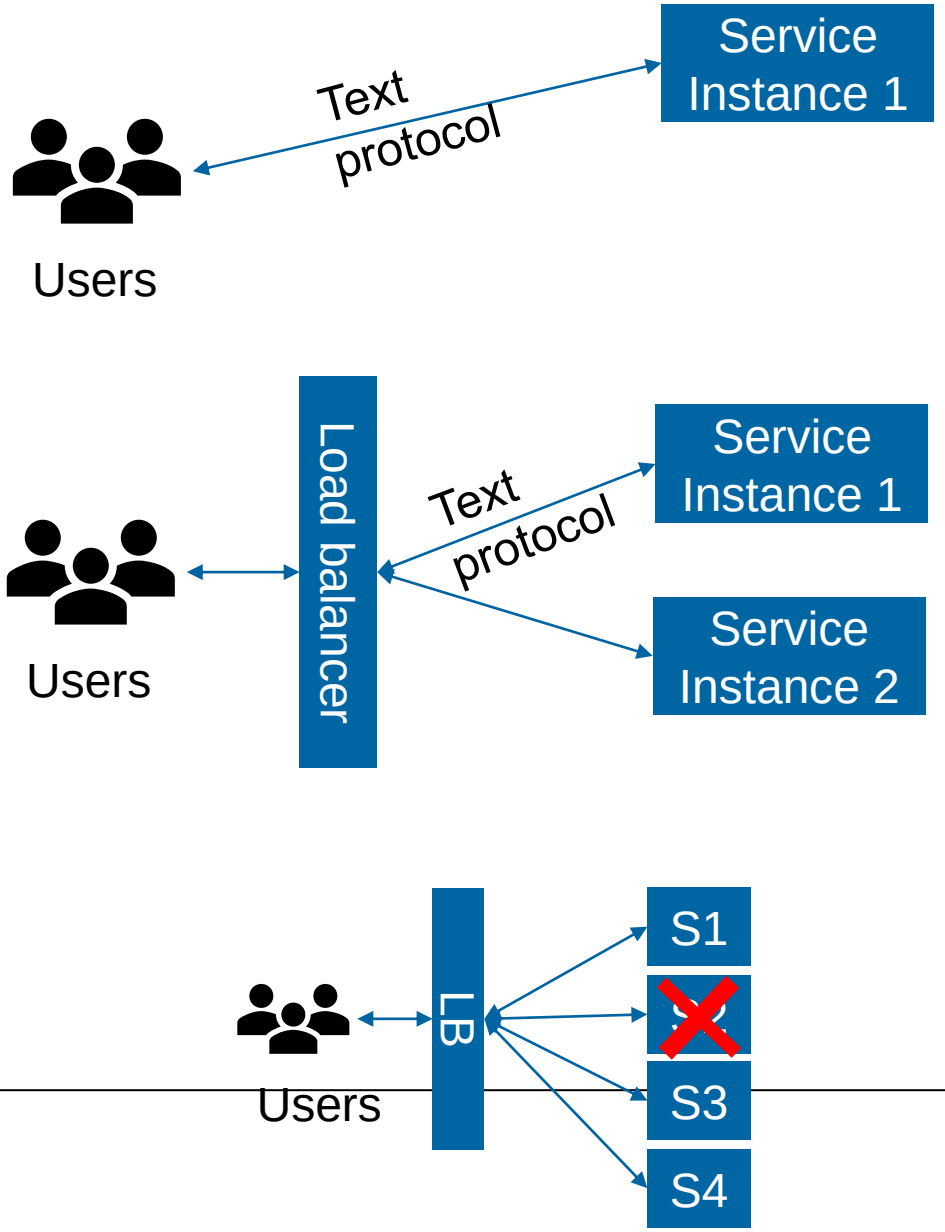
Verteilte SW-Systeme (VSS)

Nr	Date Lecture	Date Exercise	Topics	Slides	Milestones
01	18.02.2020	-	Admin, Introduction	VSS-FS20-01-Admin_v1.pdf, VSS-FS20-01-Introduction_v2.pdf, alpine-fs20_v3.ova (VirtualBox Image - root/hsr12345 (47MB))	Group forming idea for services
02	25.02.2020	25/26.02.2020	Protocols, Basics, Scalability	VSS-FS20-02-Proto_v4.pdf	Setup dev environment
03	03.03.2020	03/04.03.2020	Application Protocols	VSS-FS20-03-App-proto_v2.pdf	Information flow
04	10.03.2020	10/11.03.2020	Distributed Applications		Load balancer
05	17.03.2020	17/18.03.2020	RPC and Queues		Authentication

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
    - 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 of load balancing: Hardware, Cloud-based, Software load balancer

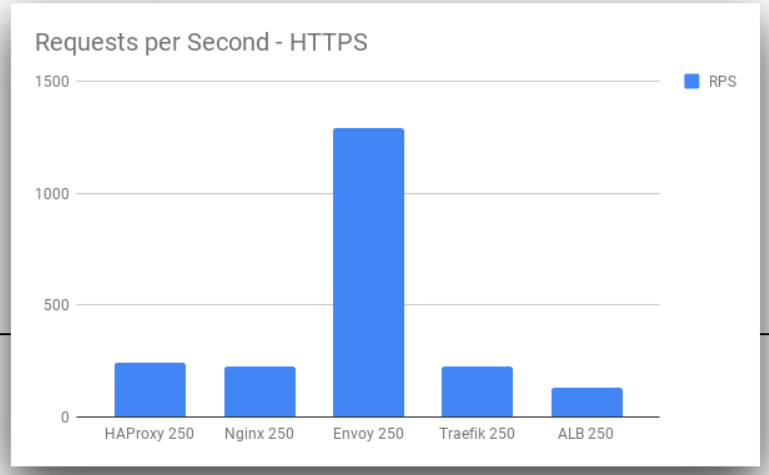
- **Hardware load balancer**
  - Application delivery controller (ADC)
    - Also a load balancer, remove load from web servers
  - HW-LB use proprietary software, which often uses specialized processors
    - Less generic, more performance
    - Some use open-source SW, e.g., Haproxy
  - E.g., Barracuda, F5, Cisco
  - Only if you control your datacenter

- **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

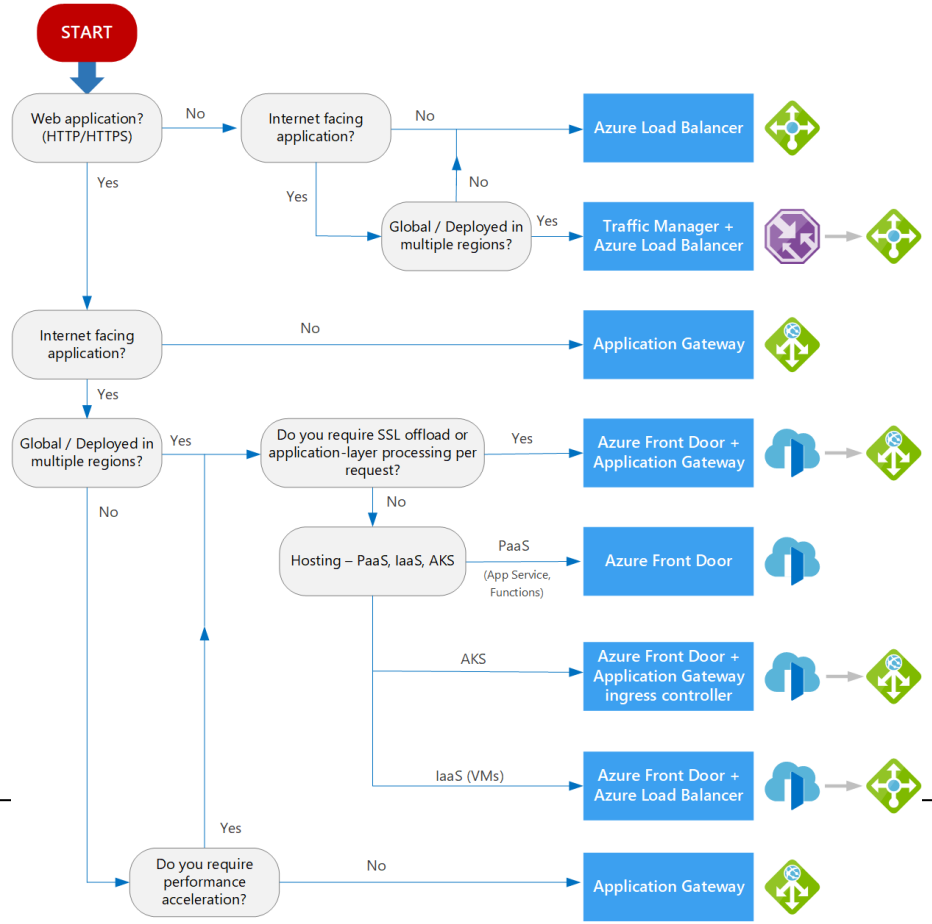
- **SW vs. SW / SW vs. HW**
  - strong opinions, funny opinions, other opinion, but: “We encourage users to benchmark Envoy in their own environments with a configuration similar to what they plan on using in production [source]”

Types Load balancing

- **Cloud-based load balancer**
  - Once you are in the cloud, cloud services offer load balancing
    - DIY – not a good idea, 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, L7), Azure (L4, L7)



- Benchmark, benchmarks,
- **Azure: what to use:**
  - Choices, choices, choices…



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.
- **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

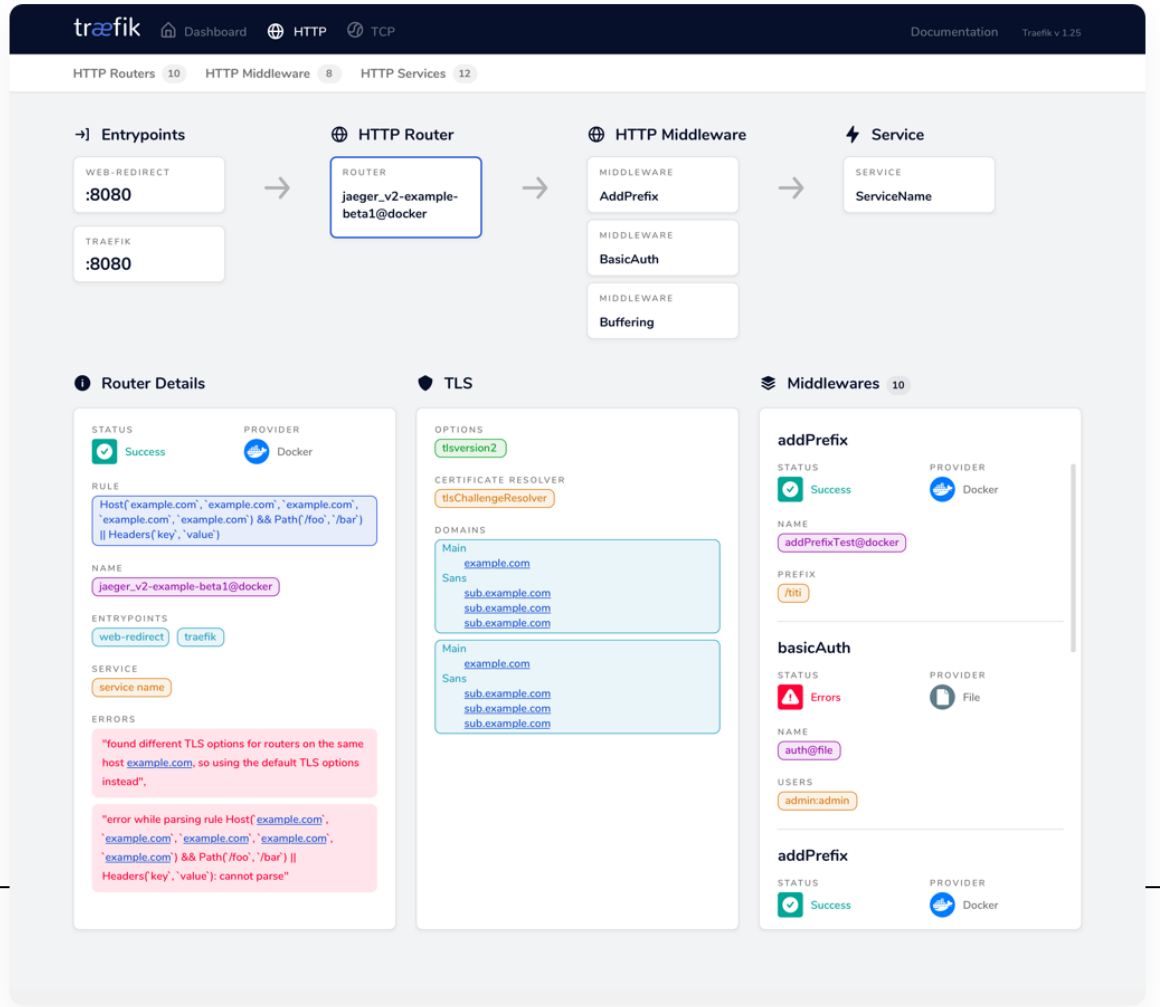
- **Load Balancing Algorithms**
  - Round robin – loop sequentially
  - Weighted round robin – some server are more powerful
    - You can put weighted in from of everything
  - Least connections – fewest current connections to clients
  - Least time – combination of fastest response time and fewest 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**

URL: b.socrative.com/student/

Room: HSR

Traefik

- **Open Source, software-based load balancer: https://github.com/containous/traefik**
  - “The Cloud Native Edge Router”
  - L4/L7 load balancer
  - Golang, single binary
  - Authentication
- **Dashboard**
- **Many examples for v1, this lecture: v2 (docs)**
- **Official traefik docker image**
  - For alpine: download latest binary
  - wget https://github.com/containous/traefik/releases/download/v2.1.6/traefik_v2.1.6_linux_amd64.tar.gz
  - unzip



Traefik

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

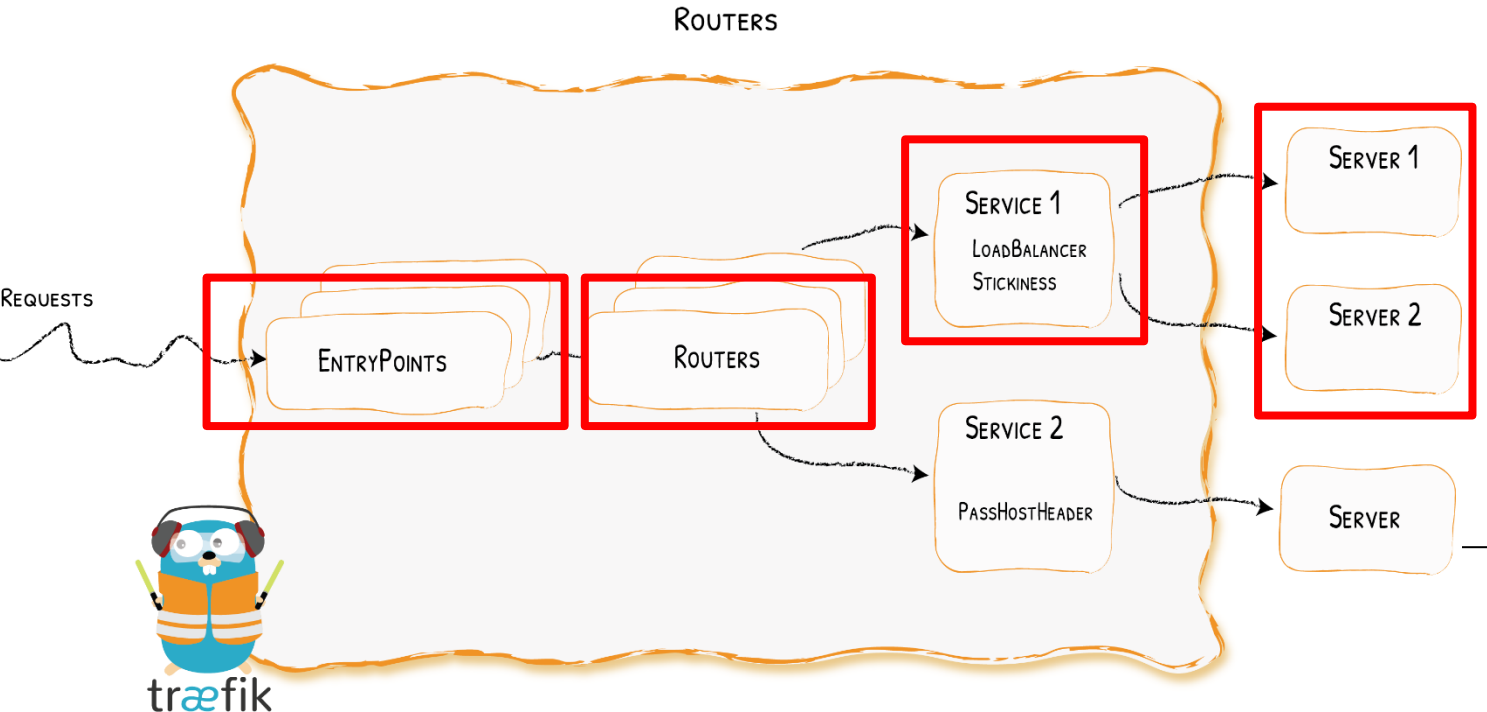
```
[entryPoints.web]
address = ":80“

[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"
```

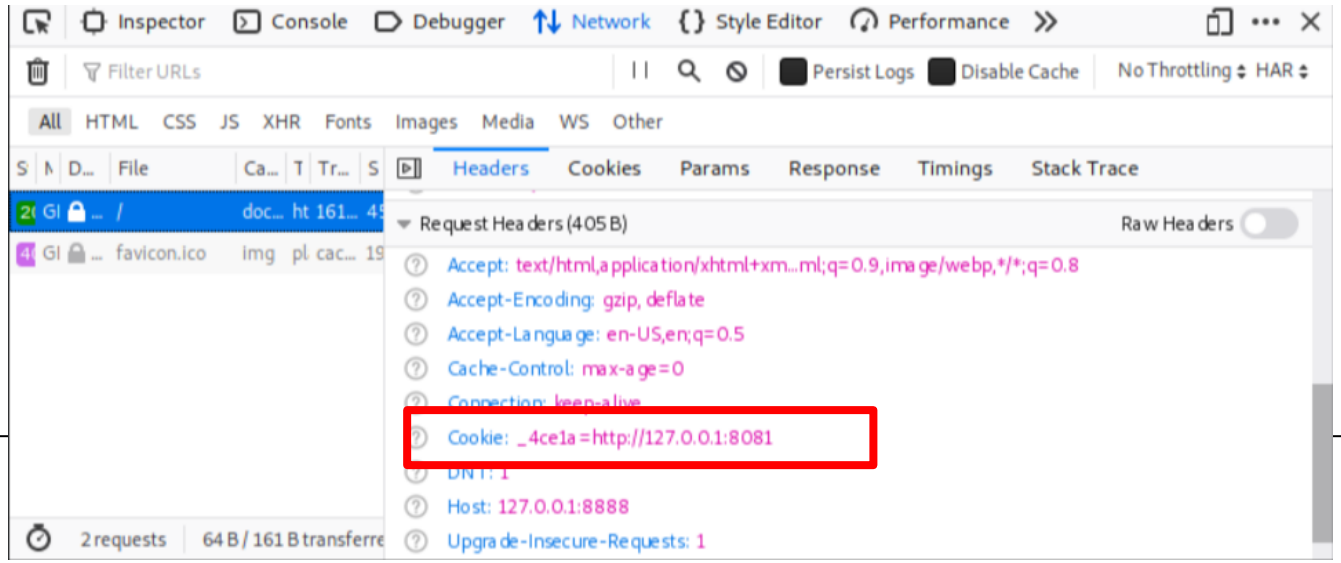
Service

- **As a start, stateful service**
  - Key value storage, in-memory (golang)
  - Specify port, we run it multiple times
  - GOOS=linux go build server-stateful.go
  - copy to alpine
- **Start one service, open 127.0.0.1:8888**
  - What Load Balancing Algorithms is used?
- **Stickiness with cookies**
- **Let's add a health check**
- **Weighted round robin**
  - load balance between services and not between servers (example)

```
router.GET("/health", func(w http.ResponseWriter, r *http.Request, _ httprouter.Params) {
  w.WriteHeader(http.StatusOK)
})
```

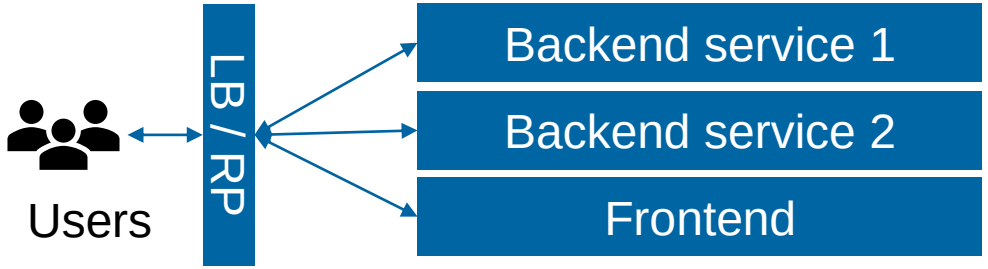
```
[http.services.coinservice.loadBalancer.healthCheck]
path = "/health“
interval = "3s"
timeout = "1s"
```

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

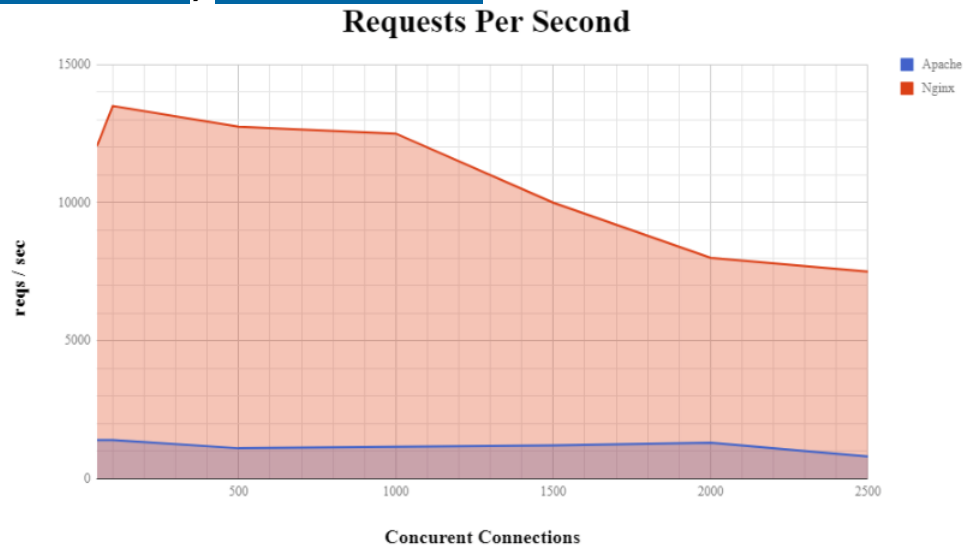


NGINX

- **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 logging in commercial version, no active health checks, no sticky sessions (not usable in prod env)
- **12.12.2019 Russian police raid NGINX Moscow office**
  - Copyright dispute with Rambler while the author was working for that company



- **Performance tuning – some ideas**
- **Benchmarks, benchmarks**

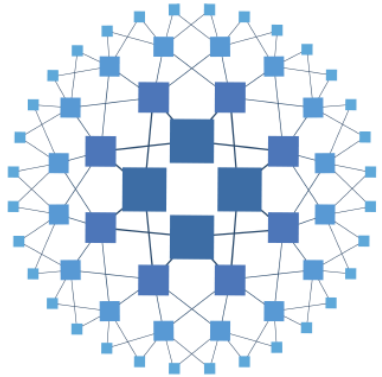


NGINX

- **apk add nginx**
- **Start: /etc/init.d/nginx start**
- **Add configuration**
- **Health check**
  - Inband/passive (active - commercial)
- **Default: round robin**
  - Least connected (least_conn)
  - Sticky (ip_hash), cookie (commercial)
  - Weighted balancing (weight=1)

```
#/etc/nginx/conf.d/default.conf
upstream coinservice {
        #least_conn;
        server 127.0.0.1:8080 weight=1;
        server 127.0.0.1:8081;
}
server {
        listen 80 default_server;
        listen [::]:80 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:

HAPROXY

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

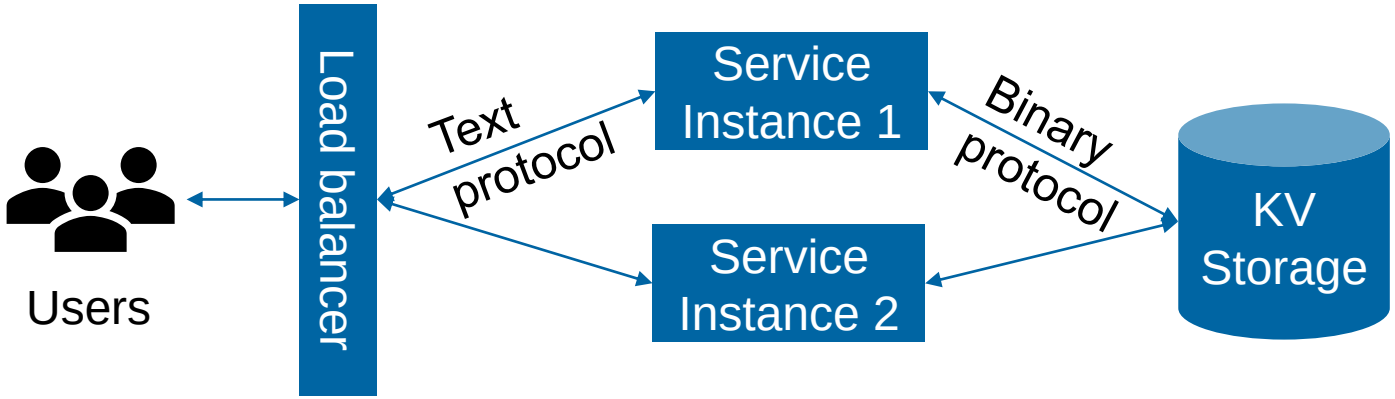
```
#/etc/haproxy/haproxy.cfg
defaults
    retries 3
    timeout client 30s
    timeout connect 4s
    timeout server 30s

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
```

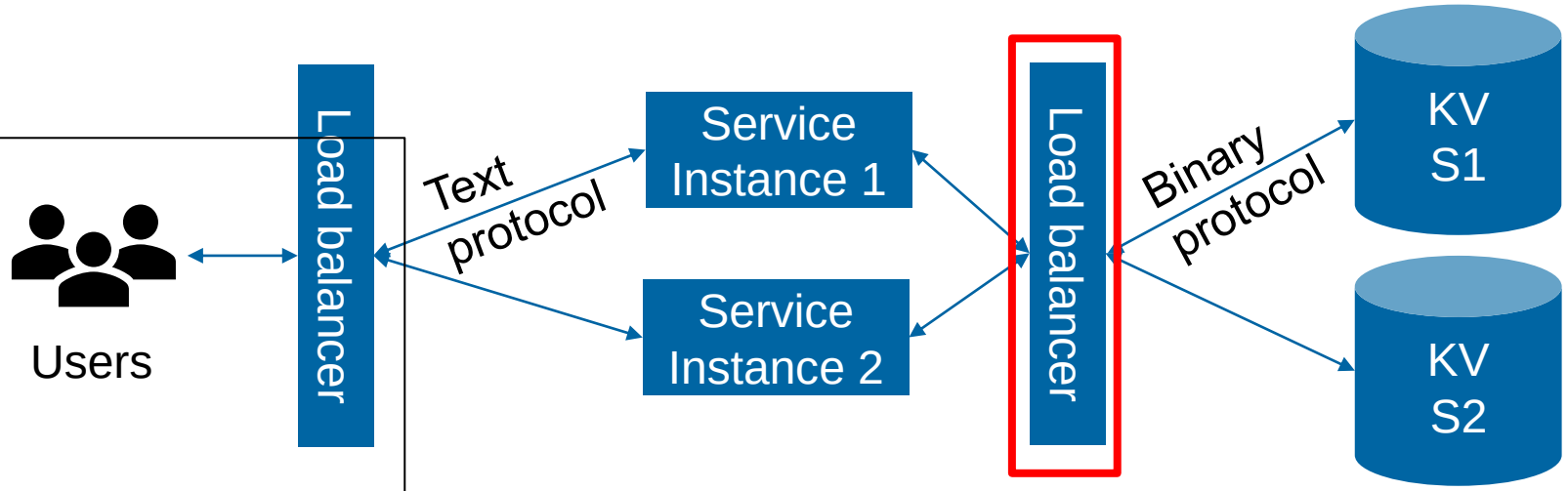
gRPC Load Balancing

- **Focus on HTTP(S) with Traefik, HAproxy, NGNIX**
- **Four choices**
  - Proxy
  - P2P
  - Look-aside
  - L4



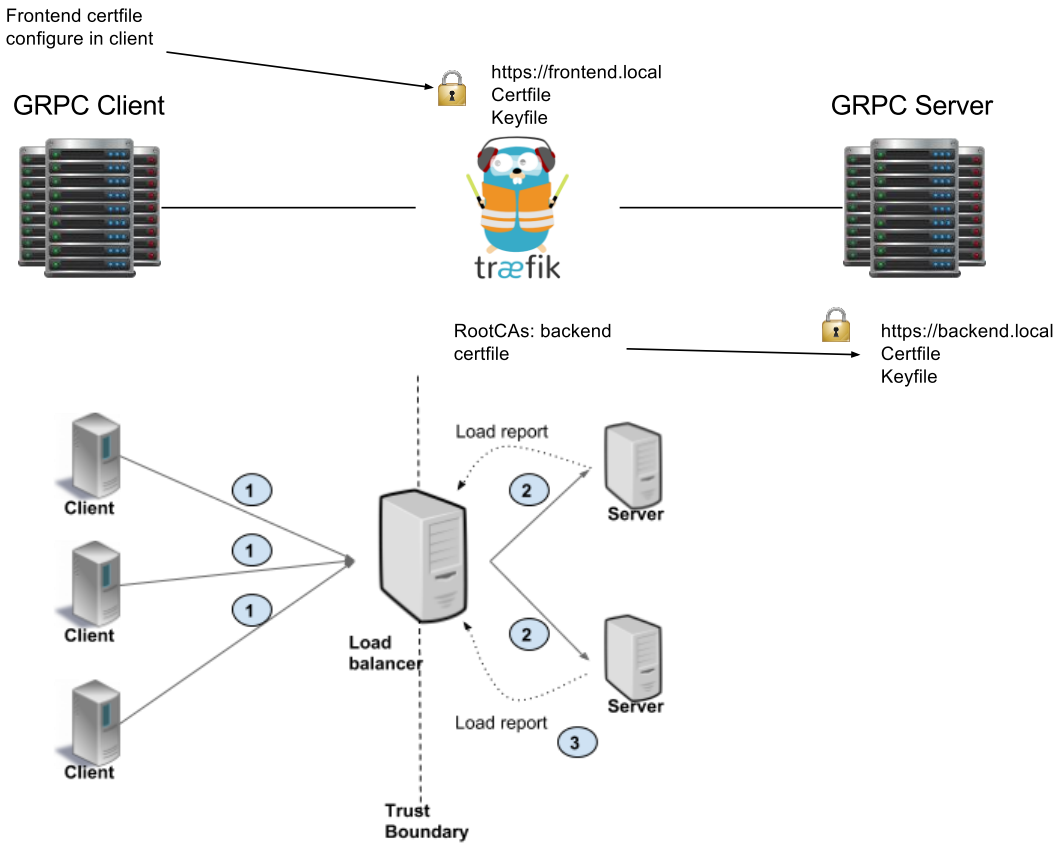

```
#/etc/haproxy/haproxy.cfg
…
frontend rpc
    bind                :70
    mode                tcp
    default_backend     storage

backend storage
    mode        tcp
    balance     source
    server      s1 127.0.0.1:7070 check
    server      s2 127.0.0.1:7071 check
```



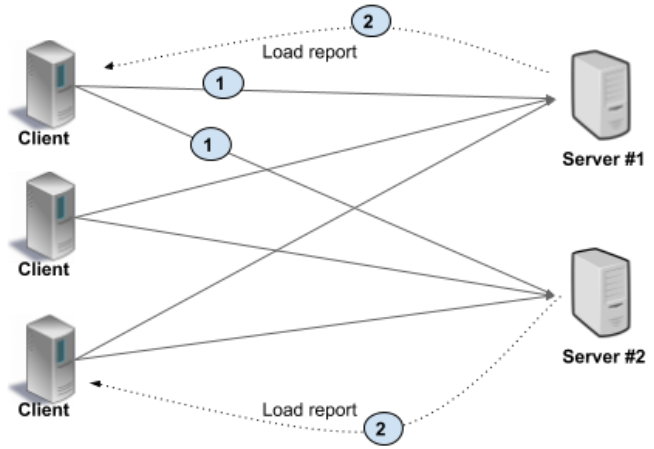
gRPC Load Balancing

- **Proxy gRPC with [Traefik]**



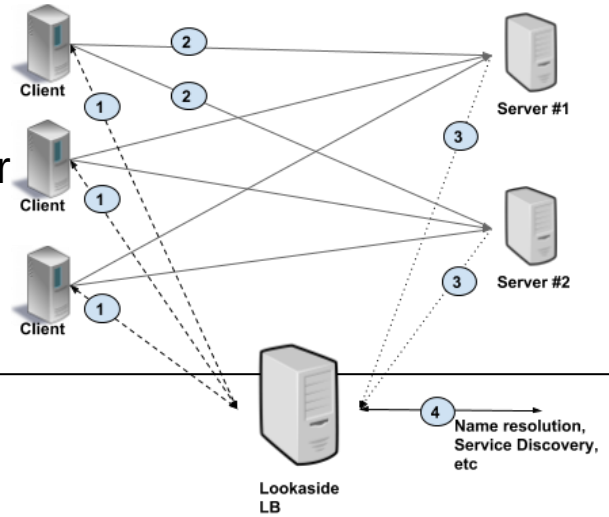
https://grpc.io/blog/grpc-load-balancing/

- **P2P/ client side**
  - Complex client, keeps track of server load and health, implements load balancing algo



- **Look-aside**
  - Mix of P2P/proxy
  - Ask proxy for server

→ connect directly



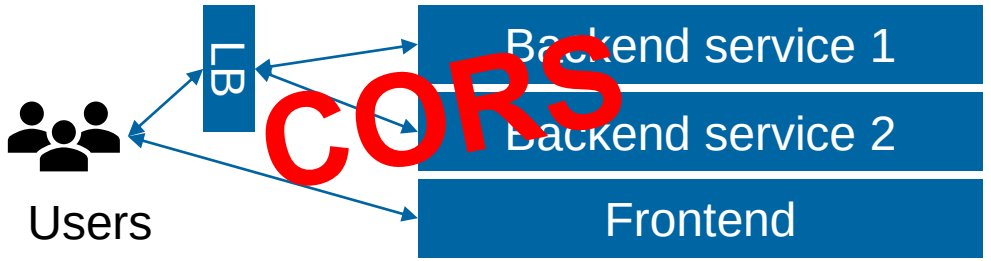
CORS

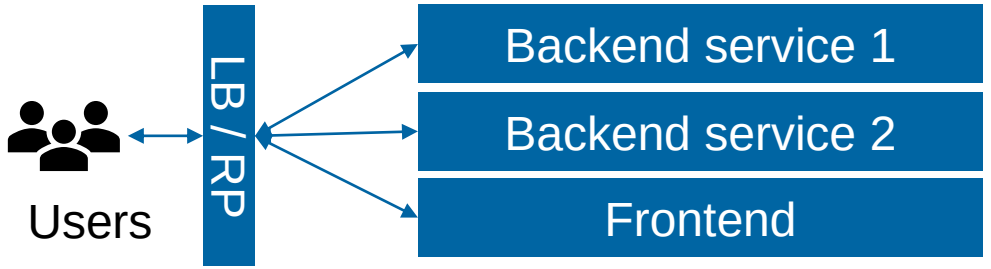
- **CORS = Cross-Origin Resource Sharing**
  - Some students ran into CORS issue
  - Mechanism to instruct browsers that runs a resource from origin A to run resources from origin B
  - For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts (among others)
- **Solution**
  1. 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

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

  → For dev: Access-Control-Allow-Origin: *
  - w.Header().Set("Access-Control-Allow-Origin", "*")
- **Reverse proxy**





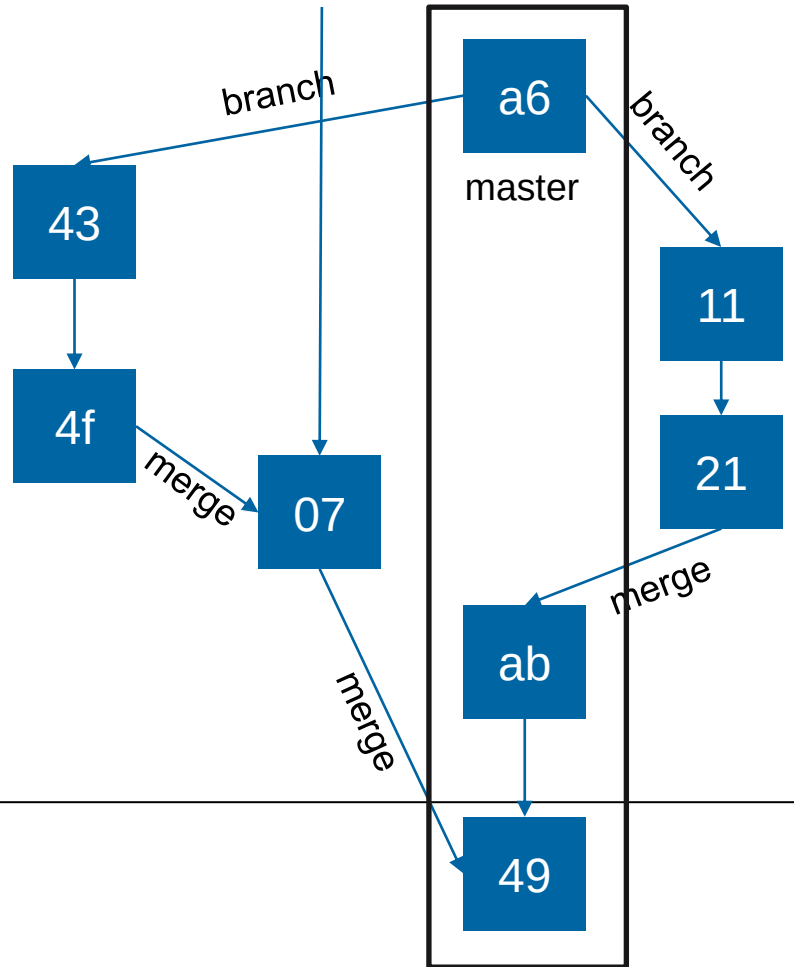
URL: b.socrative.com/student/

Room: HSR

Distributed Version Control

- **What is a version control system**
  - Management of changes to documents/source code
  - Revisions
    - Compared, Restored, Source code: merged
- **Why version control**
  - Control over changes
  - Teams work on source code, features, bugs, different deployment
- **Why distributed version control system**
  - Simplest method: locks
  - Distributed revision control - complete codebase, including its full history, is mirrored on every developer's computer

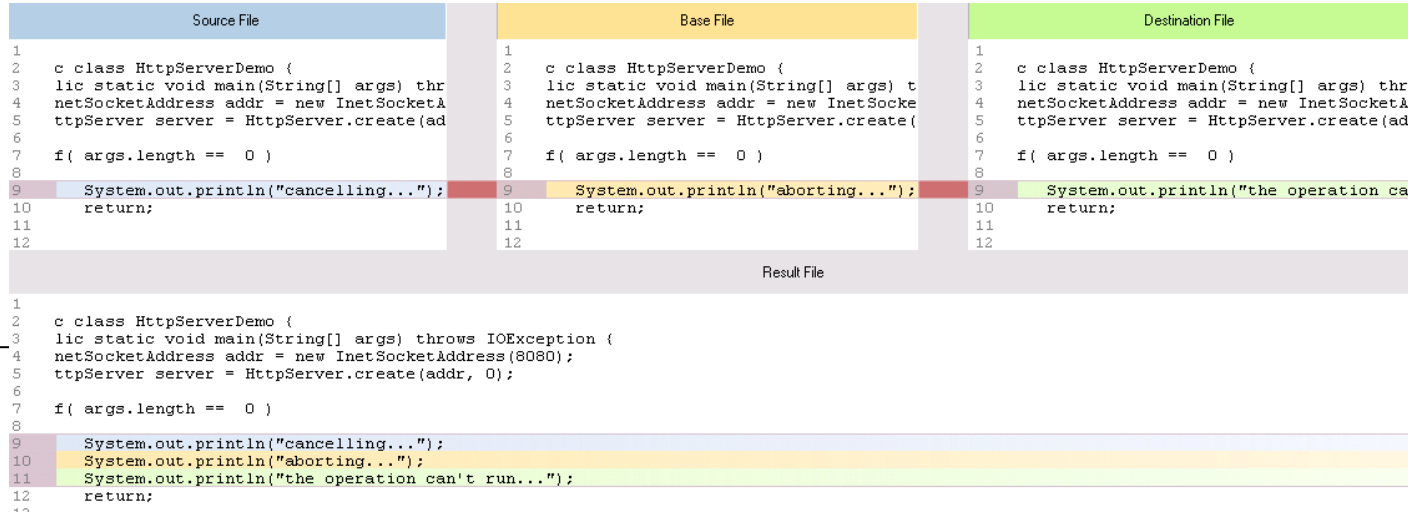
- No single master copy, working copies
- P2P, every peer has full repo. Commit to your local repo → push/pull to interact with others



Distributed Version Control

- **Git – distributed version control system for source code**
- **Created in 2005 by Linus Torvalds, creator of Linux**
  - Before 2005 Linux used BitKeeper, a proprietary VCS
  - Some hackers stared to reverse engineer the protocol of BitKeeper → they withdrew the free license. → git, mercurial
  - Good article about history of git
- **Distributed development**
  - Local copy of the full development history
  - Merkle trees for integrity - not possible to change the old versions without noticing

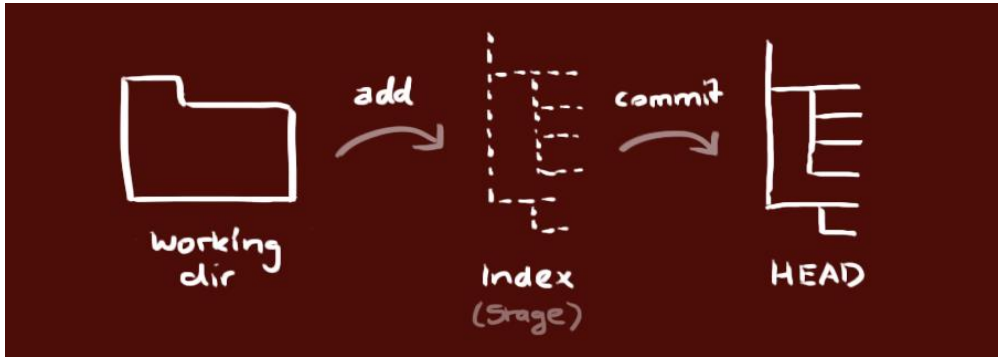
- **Git is dominant**
  - Git 87.2%
  - Subversion 16.1%
  - MS Team Foundation Server 10.9%
  - Zip files: 7.9%
- **Merge: 3-way merge**
  - 2-way merge with diff (apply diffs sequentially)
  - Small independent changes: typically ok
  - Same line changes: merge conflicts, e.g.,



Distributed Version Control

- **Simple commands (create repo with github)**
  - mkdir test-git; cd test-git
  - git init
    - Creates .git with meta information
  - create file…
  - git add file.txt
  - git commit
    - Everything still local
  - Connect with remote git repo
    - Github (selfhost with gitea, gogs), you need to have account, and created the repo
  - git remote add origin git@github.com:tbocek/test-git.git
  - git push -u origin master
- **Now you are ready, check github website**

- **Clone a repository**
  - git clone git@github.com:tbocek/test-git.git
  - Or https also for readonly
- **Workflow**
  - Local



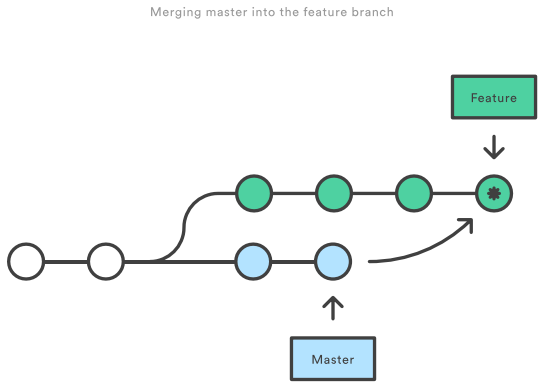
  - Remote: pull/push

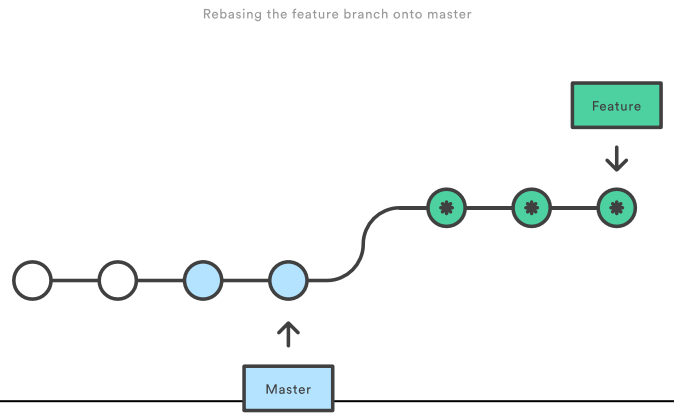
Distributed Version Control

- **Branching**
  - git checkout -b feature
  - git checkout **branch-name**
    - Switch between branches
  - git branch -a
    - Show branches
  - git branch -d **branch-name**
    - Remove local branch
  - git push -d origin feature
    - Remove remote branch
  - git push origin feature
    - Push it remote, before, everything was local
- **Merging**
  - git merge feature
- **Get changes from remote**
  - git pull / git pull --rebase
- **Go back**
  - git log
  - git reset –hard cc5507b071
  - git revert cc5507b071..HEAD
    - Revert last commits
- **Tags (release in github specific)**
  - git tag tagname cc5507b071
  - git checktout tagname
  - git show tagname
  - git push origin tagname
    - To sync it with remote

Distributed Version Control

- **Merge conflict**
  - Edit file and fix it
  - git add *
  - git commit
- **Rebase**
  - Merge preserves history whereas rebase rewrites it
  - Rebasing is better to streamline a complex history





- **Squash**
  - git checkout master
  - git merge –squash feature
  - git commit
- **Cherry picking**
  - git cherry-pick cc5507b071

URL: b.socrative.com/student/

Room: HSR

Questions?

Dr. Thomas Bocek
thomas.bocek@hsr.ch