OST
Eastern Switzerland
University of Applied Sciences

Distributed Systems & Blockchain (DS1)

Admin

Thomas Bocek

11 April 2021

DS1: Plan 2021

Nr	Date	Vorlesung*	Dozent
01	23.02.2021	Admin / Introduction	Bocek
02	02.03.2021	Protocols, Basics	Bocek
03	09.03.2021	Application Protocols	Bocek
04	16.03.2021	Distributed Applications: load balancing	Bocek
05	23.03.2021	Distributed Applications: CORS, gRPC, git	Bocek
06	30.03.2021	Authorization (basic auth, digest auth, session-based, JWT), queues	Bocek
07	-	Eeaster Vacations	Bocek
08	13.04.2021	Containers and VMs	Bocek
09	20.04.2021	Docker, Docker Compose, Connecting everything	-
10	27.04.2021	Real-world examples	TBA
11	04.05.2021	Blockchain, Bitcoin, Ethereum I	Bocek
12	11.05.2021	Blockchain, Bitcoin, Ethereum II	Bocek
13	18.05.2021	Challenge Task Presentations I	you
14	25.05.2021	Challenge Task Presentations II	you
15	01.06.2021	Challenge Task Award Winner Announcement, Q&A	Bocek

* topics may change

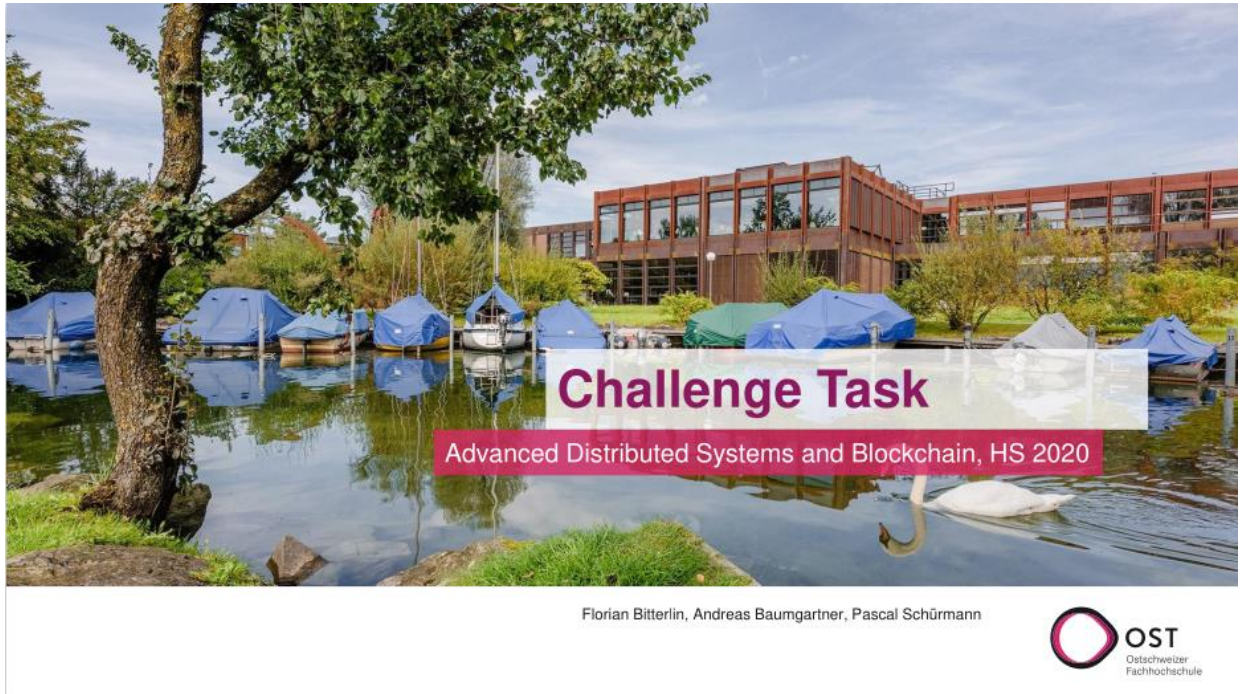
Nr	Date	Vorlesung*	Suggested Milestone
01	23.02.2021	Admin / Introduction	Group forming, idea for services
02	02.03.2021	Protocols, Basics	Group forming, idea for services
03	09.03.2021	Containers and VMs	Information flow, setup dev environment
04	16.03.2021	Distributed Applications: load balancing	Load balancer working
05	23.03.2021	Authorization (basic, digest, session-based, JWT, OAuth)	JWT Authorization
06	30.03.2021	Application Protocols and RPC	Storage backend ready
07	-	Eeaster Vacations	
08	13.04.2021	Distributed Applications: git, rsync	
09	20.04.2021	Docker, Docker Compose, Connecting everything	
10	27.04.2021	Real-world examples	Frontend ready
11	04.05.2021	Blockchain, Bitcoin, Ethereum I	Connecting everything
12	11.05.2021	Blockchain, Bitcoin, Ethereum II	Testing, testing, testing
13	18.05.2021	Challenge Task Presentations I	
14	25.05.2021	Challenge Task Presentations II	
15	01.06.2021	Challenge Task Award Winner Announcement, Q&A	

* topics may change

OST

Admin

- Some groups are already finished
  - Early hand-in is appreciated!
  - Hand-in:
    - Repo source code, or give me access to gitlab/github
    - Powerpoint or PDF
      - Goal / motivation / idea
      - Software design / architecture
      - Implementation / libraries, frameworks
      - Requirements fulfilled
      - Demonstration
    - Example: HS20, group 6
    - If you send code via email, send to thomas.ost@bocek.ch (OST mail filter)
- Regular hand-in
  - 17.05.2021, 23:59 (CET)



OST

Load balancing: HTTP Keep Alive

- Traefik does not seem to loadbalance
  - Nginx does
  - “Problem” – traefik respects the HTTP keep alive option,
    - Default 0s in the go example app, also in node
- Keep Alive: A header directive to indicate that the HTTP connection will be reused
  - Helpful to reuse the same connection
- After turning off, round-robin load balancing

```
server := &http.Server{Addr: ":8080", Handler: http.HandlerFunc(dbQuery)}
server.SetKeepAlivesEnabled( v: false)
server.ListenAndServe()
```

```
var express = require('express');
var app = express();
var server = app.listen(5001);

server.keepAliveTimeout = 30000;
```

Deployment: Health Checks

- Problem: in your docker-compose.yml you define services: db and service
  - Service needs the db up and running, how can hello-world check that?
    1. Quick and dirty: wait loop in the app
    2. Less quick and dirty: before you start your app, execute wait4port – waits until port becomes ready (assuming if a port is available, service is fully available)
    3. Better: healthchecks in docker-compose.yml

```
err = db.Ping()
now := time.Now()
for err != nil &&
now.Add(time.Duration(30)*time.Second).After(time.Now()) {
        // check the connection
        err = db.Ping()
        time.Sleep(time.Second)
}
```

```
db:
    container_name: flatfeestack-db
    image: postgres:13-alpine
     volumes:
      - ./.db:/var/lib/postgresql/data
    ports:
      - 5432:5432
    #https://docs.docker.com/engine/reference/builder/#healthcheck
    healthcheck:
      test: [ "CMD-SHELL", "pg_isready -U postgres" ]
      interval: 1s
      timeout: 5s
      retries: 15


backend:
    container_name: flatfeestack-backend
    build: ./backend
    ports:
      - 9082:9082
    environment:
      - PORT=9082
    depends_on:
      db:
        condition: service_healthy
```

Deployment: Distroless

- Alpine is lightweight, but can we go more lightweight?
  - "Distroless" Docker Images
    - Based on the Debian Distribution
- In production, CVE scanner is often used when deploying image
  - CVE: Common Vulnerabilities and Exposures
    - E.g., CVE-2021-30480, media coverage
      - 08.04.2021: Zoom zero-day discovery makes calls safer, hackers $200,000 richer
    - E.g. Clair scanner, test local (postgres:alpine / postgres:latest)
- Alpine image was affected by CVE-2020-28928
  - Result: if any CVE is found, deployment is stopped
  - Result: spend half day to investigate issues, analyze impact, fix all images.

- Distroless

  + Less code ~ less complexity ~ less security issues

  - Less convenience
    - E.g., test connectivity with curl/wget: no
    - E.g., add packages with apk add xyz: no
    - E.g., run chown/chmod: no
- Fastauth now uses distroless – static
  - Statically compile your binaries

```
build:
    go build -ldflags "-linkmode external -extldflags -static"
test:
```

Deployment: Distroless

- Dockerfile
- Assets in Distroless static
  - Docker pull gcr.io/distroless/static
  - Docker save gcr.io/distroless/static
    - SSL certificate and timezone information
    - FROM scratch
      - No certs, no tz
- Base vs Alpine
  - gcr.io/distroless/base
  - alpine:3.13

```
FROM gcr.io/distroless/static
WORKDIR /home/nonroot
COPY --from=builder /app/login.html /app/banner.txt /app/fastauth /app/rmdb.sql /app/init.sql ./
USER nonroot
ENTRYPOINT ["/home/nonroot/fastauth"]
```

- 6MB vs 16.6MB – Why distroless
  - Less attack surface, many convenience binaries missing
    - No shell access
    - Performance: CPU vs RAM
      - E.g., strlen: optimized for speed in assembler
  - Building image? Alpine may be slower, but image is smaller (download faster, but its cached), statically compile: glibc/musl