Aalto University
School of Electrical
Engineering

Personal coursework

25.1.2019

Santeri Paavolainen

What is a software architect?

What is a software architect?

Personal course work

- **This is the largest single work item on the course**
  - Based on ECTS: 135h of work for the whole course, removing lectures, group assignments and exercise sessions leaves ~ 80h → 6h / week

- **You are expected to design and develop a**
  - Microservice architecture with >2 distinct services
  - At least one serverless or stateless component
  - Integrates three aspects of microservice architectural patterns (see later)

What is a microservice architecture?

- That's the topic of this course
- If you want a crash course, go to Martin Fowler's microservices page

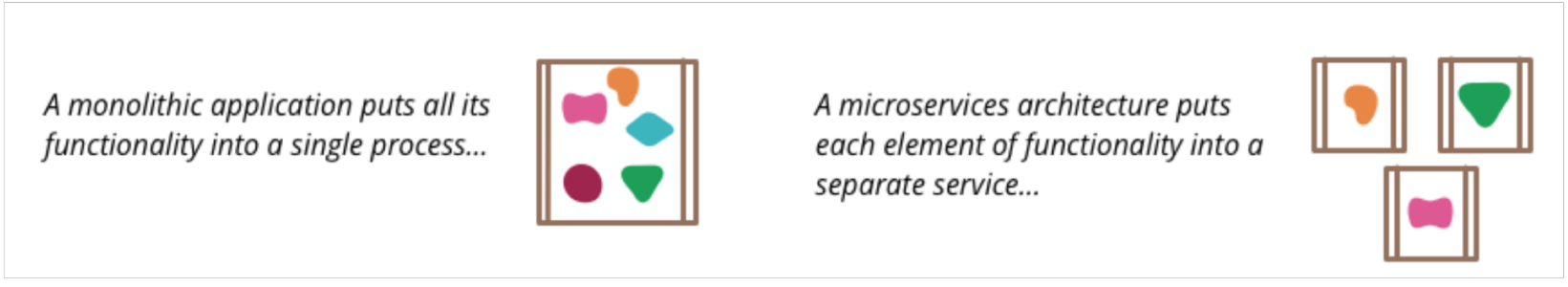

Image source: Martin Fowler

Personal course work

- **It must work at the level of a "demonstrator" piece**
  - Actual functionality can be trivial or mocked
  - However, the three microservice aspects cannot be mocked, and the overall inter-service operations must be functional

- **Focus is on how the service is structured and operated**
  - The "business functionality" is relevant only as much as it is needed for testing, demonstrations etc.
  - Of course, working on a project is probably more meaningful if the functionality makes sense to you

What's in a demonstrator?

- **You should invent yourself a real-world (business) scenario**
  - Pet grooming service reservation?
  - Mobile application backend for group messaging?
  - Some service you'd run for your home automation and monitoring?
- **Why some real-world scenario?**
  - Because generalization is difficult → easier to take a scenario and work on it
  - Easier for **others** to understand what a specific microservice is for
    - *"Service FOO calls service BAR to perform operation XYZ"*
      vs.
      "Front-end calls geolocation service to get user's geographical coordinates for focusing on the correct area on a user-visible map."

What can be mocked and what not?

"mocked" = technojargon for "faking it"

- **Assume one microservice you create is *GeoLocation of an IP address***
  - It is used like: GET /geo?ip=1.2.3.4
    It returns 200 OK and JSON response on success: {"lat": 60.1234, "lon": -5.1234}
- **What <u>needs to be implemented</u> in fully functional manner**
  - REST server
  - /geo endpoint
  - Specification for your service (you can use OpenAPI format, or just words)
- **What can be <u>mocked</u> (e.g. not implemented functionally correctly)**
  - Correct functionality is irrelevant, just validity of response

Unacceptable:

```
def geo(ip):
  return 200, {}
```

Acceptable:

```
def geo(ip):
  if not valid_ip(ip):
    return 400, "input not ip address"
  return 200, {"lat": random.uniform(-90.0, 90.0),
               "lon": random.uniform(-180.0, 180.0)}
```

Why?

Should I do X or Y?

- **Synchronous vs. asynchronous vs. message-passing**
  - It's your architecture and your service, you have to make a reasoned choice
- **REST vs. gRPC vs. Thrift vs. XYZ?**
  - Same thing – some scenarios may work better on some, but in the end, it is your decision (hint: you won't get penalized for sticking to REST)
- **nginx vs. AWS ELB? X vs. Y?**
  - I really do not care on technology choice – just on how it is used

“Aspects”?

- **Course covers a wide variety of design and operational aspects of microservices**
  - HA, failovers, tracing, logging, service authentication, discovery, …
- **Infeasible to implement all of these in the course work**
- **Each student must pick three separate aspects and implement these within the scope of their work**
  - These must be functionally “complete” (no mock-ups)
  - These must be demonstratable, e.g. can be shown to work and cover the problem they are meant to solve

Aspects

Logging	Service AAA	Role-based user AAA	Discovery
Service degradation	Monitoring	Tracing	Continuous deployment
Chaos engineering	Backups and disaster recovery	Caching	Secret management
Dynamic configuration	Geographical distribution	Automated scaling	Versioning
High availability	A/B testing	…	(suggest)

Selecting aspects

- **There is no firm deadline on selecting**
  - Apart from the actual course work deadline
- **You can pick one now, all three, and change your mind later**
- **… or just choose them after something like a month or so (you've still got about 2 months until course work DL)**
  - In the meantime, you can work on the basic microservice structure and functionality first
  - Which you have to define yourself, too
- **Feel free to suggest similar aspects too**

- **You may do more than 3 aspects in your project — grading is based on the best 3**

What if aspect X for geolocation …?

- **Logging?**
  - ```
    def geo(ip):
      logger.trace("geo: {}", ip)
      if …:
        logger.error("geo: {} is not a valid ip", ip)
        …
      logger.info("geo: mapped {} to ({}, {})", ip, lat, lon)
      …
    ```
  - Include logging sidecars (potentially)
  - Implement logging server, logging analysis etc. as separate services
  - Collect logs from all of your services (not just one)
  - …

Why generating log entries is not enough?
    ```

What if aspect X for geolocation …?

- **Metrics?**
  - ```
    metrics = open("…/metrics.dat", "a")
    def geo(ip):
      print(time.now(), "start", "geo", ip, file=metrics)
      …
      print(time.now(), "end", "geo", ip, file=metrics)
      return 200, …
    ```
  - Would this be acceptable? If so, why? If not, why?
- **Resiliency?**
  - ```
    response = requests.get("http://geo.local/geo?ip={}".format(ip))
    if response.code != 200:
      … handle error …
    …
    ```
  - Ok? Why? Why not?

Personal coursework grading

- **Weight on final grade: 50%**
- **0-3 pts for general evaluation of the architecture**
  - Separable? Logically consistent? Division of responsibility?
- **0-3 pts for maintainability**
  - Can you give the code to someone else? Would they understand how it works and be able to work on it?
- **3 topic areas x 0-3 points each**
  - Implementation, demonstratability, coverage in implementation
- **0-3 pts for milestone demonstrations**
  - 1 point for demonstrating progress in milestone exercise sessions
- **Rejected on unattributed copying, deductions on excessive code re-use**
- **Total: 18 pts (x 50%)**

What's logical consistency? Etc.?

- **Again, this is something that is part of the course**

- **This – and many other – are not black-and-white things**
  - Software architectures never are
  - If you want to be a software architect, **you must learn to provide rationale for your decisions**

- **So think it this way**
  - If you don't know a shred on the topic, you are unlikely to achieve any logical consistency (etc.)
  - If you know some of the topic, then you'll make mistakes, but you'll get points
  - If you can provide rationale for you mistakes – flawed in detail, perhaps – that will help
  - Finally, you are not expected to have 10 years of software architect experience