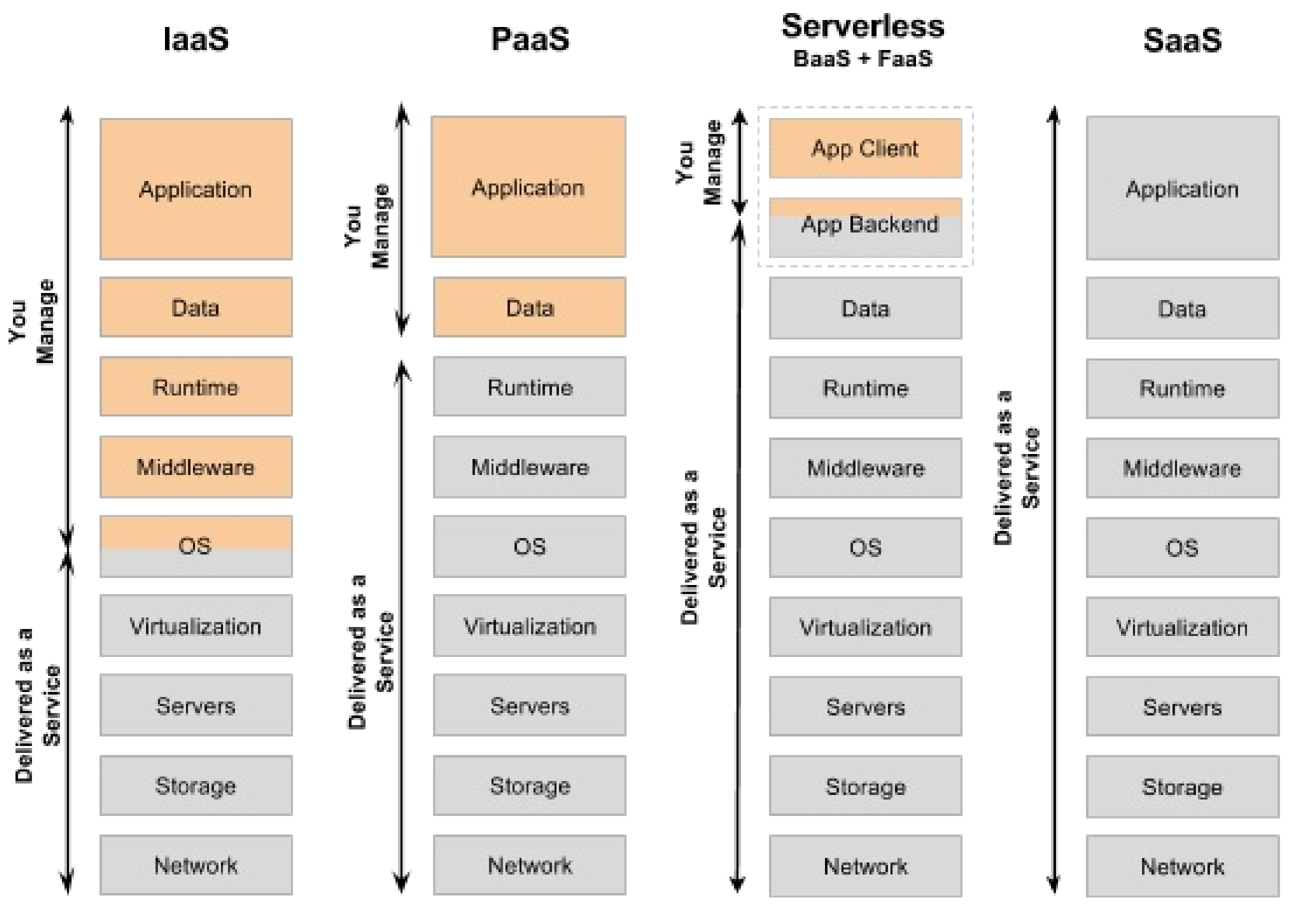
Research plan: Migrating a web application to FaaS

Aleksi Pekkala (alvianpe@student.jyu.fi)

21.11.2017

Motivation

Function as a Service (FaaS or serverless) is defined as a cloud-native platform for short-running, stateless computation and event-driven applications which scales up and down instantly and automatically and charges for actual usage at a millisecond granularity. Unlike SaaS or PaaS that are always running, but scale on-demand, serverless workloads run on-demand, and consequently, scale on-demand.



[(https://specify.io/concepts/serverless-baas-faas)](https://specify.io/concepts/serverless-baas-faas)

The two main benefits of FaaS (as I see them) are gains in developer productivity (less time spent on operations/infrastructure vs. features) and decreased hosting costs. Also "green computing", reduced operational costs, time-to-market and experimentation.

> I don't have to manage a virtual machine, operating system, patch management, scaling service, load balancing, availability, fault tolerance, provisioning, anti-virus, anti-malware, vulnerability scanning, continuous monitoring, access control, rightsizing, server tuning, intrusion detection, hardware affinity, OS dependencies [...] and I only pay for what I use (Groat & Lu, 2017 (https://www.slideshare.net/AmazonWebServices/serverless-design-patterns-for-rethinking-traditional-enterprise-application-approaches-aws-public-sector-summit-2017))

A serverless platform has a number of unique characteristics and restrictions, including statelessness, lifespan and resource limits on individual functions and an event-driven execution model. Considering these features it's relevant to ask how do we compose the building blocks of serverless (individual functions or lambdas) into larger systems? Do the same rules and patterns apply as in more traditional platforms?

> Will there be patterns for building serverless solutions? How do we combine low granularity basic building blocks of serverless into bigger solutions? How are we going to decompose apps into functions so that they optimize resource usage? For example how do we identify CPU-bound parts of applications built to run in serverless services? Can we use well-defined patterns for composing functions and external APIs? What should be done on the server vs. client (e.g., are thicker clients more appropriate here)? Are there lessons learned that can be applied from OOP design patterns, Enterprise Integration Patterns, etc.? Baldini et al., 2017 (https://arxiv.org/abs/1706.03178)

> Some of these patterns will be in application architecture. For instance how big can FaaS functions get before they get unwieldy? Assuming we can atomically deploy a group of FaaS functions what are good ways of creating such groupings - do they map closely to how we'd currently clump logic into microservices or does the difference in architecture push us in a different direction? (Fowler, 2016 (https://martinfowler.com/articles/serverless.html#TheEmergenceOfPatterns))

Research questions

The above-mentioned benefits and characteristics in mind I thought it'd be worthwhile to focus on the topic of migrating an existing web app into FaaS. My **research questions** include

1. Why should a web app be migrated to FaaS?
2. What kind of patterns are there for building serverless web application backends?
3. Do the existing patterns have gaps or missing parts, and if so, can we come up with improvements or alternative solutions?
4. How does migrating a web app to FaaS effect its quality?

The first two questions are addressed in the theoretical part of the thesis. Question 1 concerns the motivation behind the thesis and introduces FaaS migration as an important and relevant business problem. Question 2 is addressed by surveying existing literature for patterns thought suitable specifically for the target class of applications.

The third question forms the constructive part of the thesis. I'm planning to implement a subset of the target app in a serverless style, utilizing the surveyed patterns and keeping a log of all the tricky parts. In case the patterns prove unsuitable for the given problem, I'll try to come up with an alternative solution. My guess at this point is that the general patterns described in literature provide an approximate fit but don't necessarily account for all the edge cases and distinct requirements of a real-world application; proposing solutions for these unaccounted-for cases would be the main product of my thesis. This might take the form of either proposing a new pattern or extending existing ones.

The last question consists of comparing the migrated portions of the app to the original version. Since the app currently has only a handful of active users I'm planning to simulate and estimate performance and hosting costs. In addition to this hard data, I'll evaluate differences between quality attributes like maintainability, extendability and testability. I'll also try to address ease of development (or developer experience), since this seems to be one of the most commonly mentioned pain points when it comes to serverless.

What is not included in the topic

- Other kinds of common serverless use cases e.g. batch/stream processing, data pipelines, scientific computing workflows
- Theory of system migration or refactoring in general (although I'll probably have to make some references to the topic)
- Inner workings of serverless runtimes (again, might have to mention briefly)

The app to migrate

https://tacit.space/ is a modern(ish) web application for taking, organizing and sharing notes. It consists of about 30k lines of Javascript and Typescript and it includes many of your typical web app features: REST API, single page app UI, complicated role-based authorization, some (soft) real-time features, email and SMS notifications, credit card billing, separate admin tools, dependencies to external services etc. The app is divided into a frontend and a backend, where the latter is further divided into a handful of distinct services.

Outline

A tentative outline:

1. Introduction
    - motivation, research questions, outline
2. FaaS/Serverless
    - general description of the platform, its implications and use cases
    - brief comparison of different vendors, evaluating the best fit for target app
    - benefits and drawbacks, focusing on the target class of applications
    - what kind of things have to be taken into consideration when implementing web application backends on a serverless platform?
3. The tacit.space app
    - describing the target app and the typical requirements of this class of applications

    - this could be incorporated in the introduction chapter, depending on the length

4. Serverless architectural/design patterns
    - how to compose individual functions into larger systems, focusing on web application backends
    - surveying emerging FaaS-specific patterns
    - analyzing the feasibility of existing non-FaaS patterns

5. The migration process (implementation)
    - explaining which parts of the app are migrated and why; options include authentication, notifications, REST, push/pull, a particularly hairy distributed transaction, interacting with external services, ...
    - implementing selected features of the tacit.space app utilizing patterns described in section 4, and proposing alternative or extended patterns

6. Evaluation
    - comparing the result with the original app
    - do the benefits/drawbacks outlined in section 2 apply?
    - addressing briefly the overall ease and amount of work involved in the migration process

7. Conclusion

Materials

Summaries of relevant research papers and other sources can be found here (https://github.com/epiphone/gradu/blob/master/refs.md).