Aalto University
School of Electrical
Engineering

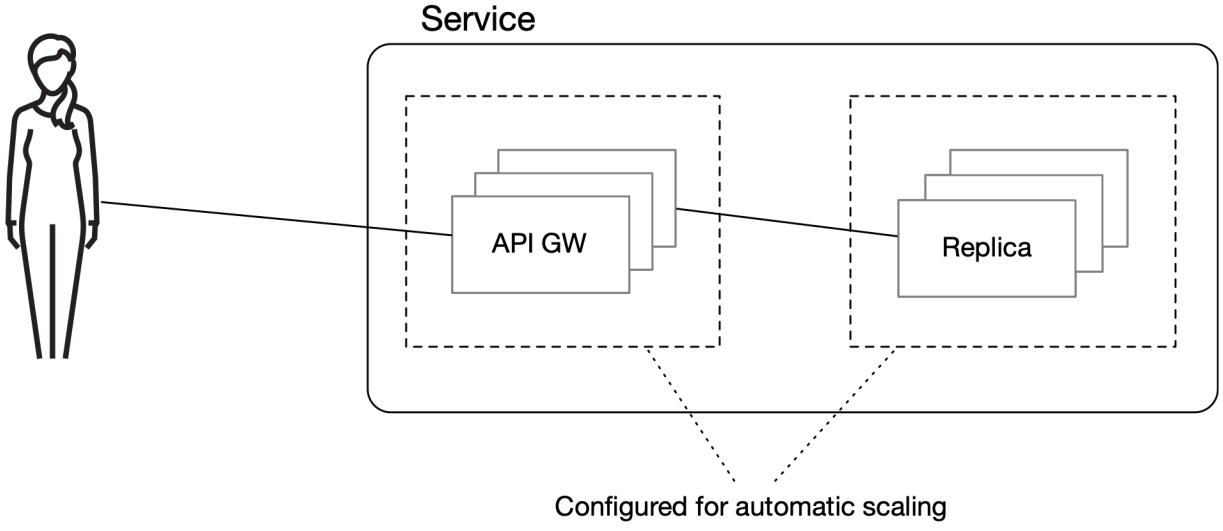
Serverless computing

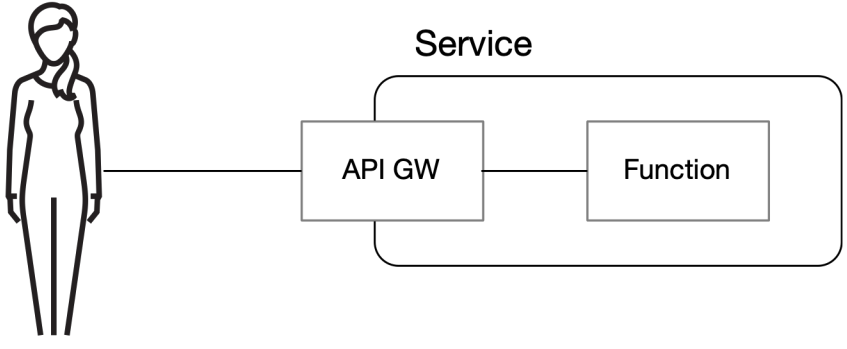
21.3.2019

Santeri Paavolainen

Serverless

- **“Serverless” (or Function-as-a-Service, FaaS)**
  - There is always some hardware somewhere (servers)
  - Operates at a function or a single service level (one or more “endpoints”)
  - ”Someone else” is responsible for
    - *Providing hardware*
    - *Scaling up and down as needed*
    - *Handling log collection*


Service
API GW
Replica
Configured for automatic scaling


Service
API GW
Function

Why serverless?

- **Simplicity**
  - Removes lot of operational concerns related to resource management
- **Scalability**
  - Always a goal for potentially Internet-scale services
  - Helps guarantee customer satisfaction even under unexpected loads
- **Costs**
  - While not definite, anecdotes abound of massive cost savings (wrt/ VMs and containers)

Why not serverless?

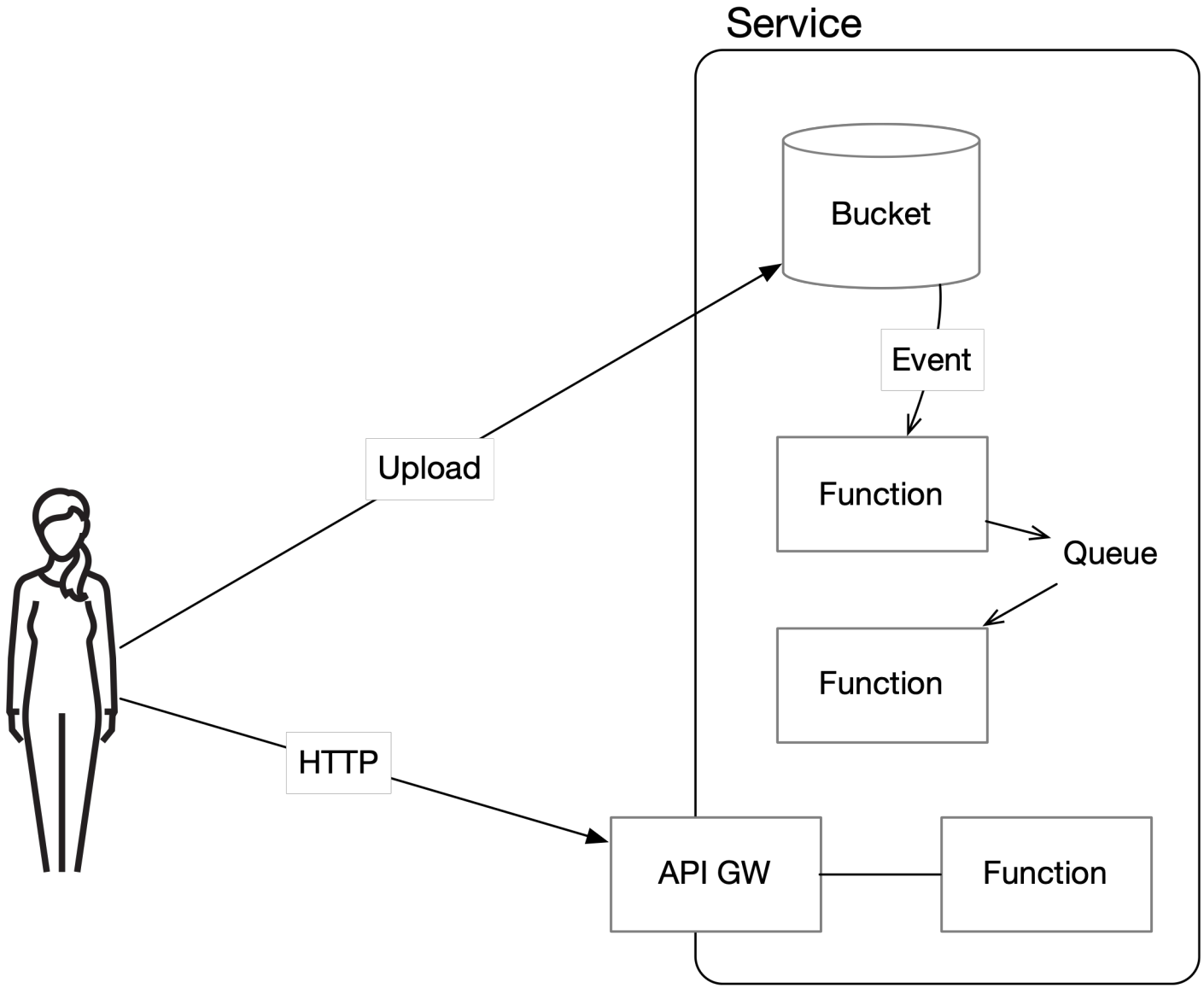
- **Tooling is now better**
  - It was abysmal previously, but has it gotten good enough? Stable?
- **Testing and development**
  - Getting better, but still friction that doesn't exist with container/instance based services
- **Unbounded costs**
  - Probably more of a psychological thing, but theoretically a DDoS or programming bug can blast your budget
- **Deployment coordination**
  - Errr, are you deploying a service, a function, or what?
- **Summary: Lots of friction between existing "ways of working", and tooling not yet fully formed (best practices still in flux?)**

Technical limitations

- **These vary from service to service**
- **Limitations**
  - Maximum runtime (seconds, e.g. request processing timeout)
  - Memory limits
  - No persistent local storage (even containers have more)
- **Latency**
  - Can be higher
  - “Cold functions”

Event model

- **Serverless uses an event model**
  - For HTTP, receives a request event
- **Many other event sources**
  - Data streaming
  - Messages from queues
  - IaaS internal events (like bucket upload complete)
  - Chimes (e.g. cron triggers)


Service
Bucket
Event
Function
Queue
Function
Upload
HTTP
API GW
Function

Services and tools

- **AWS Lambda probably best known**
  - Technically not the first, but close enough …
  - All IaaS vendors have FaaS offerings now
- **Kubeless, OpenFaaS, …**
- **Development frameworks (tons and tons)**
  - *Caveat Emptor*: Littered with abandoned and superseded projects!
  - Serverless Framework, Chalice, AWS SAM, …
  - Not really necessary on a fundamental level (it's just code! upload it!)

Serverless vs. microservices

- **Is a serverless ”thingie” a microservice**
- **Or is it a component in a microservice?**

- **Depends …**
  - If serverless function uses SQL or Redis cluster for state?
  - If providing frontend services (filtering, authentication etc.) for another serverless? Or a containerized backend?

- **Consider service boundaries and coupling**
  - Clear boundary and closely coupled?
  - Unclear boundary and loose coupling?

Some patterns

- **API gateways**
- **Separate state management**
  - Another microservice
    - *Delegate state storage and management elsewhere*
  - Some stateful component specifically for this purpose
  - FaaS as a component in otherwise stateful service

Kubeless example

- **Setup steps**
- **”Hello world” as a function**
  - Not using any framework on purpose (thus specific to Kubeless)

- **Ingress**
  - Add HTTP ingress controller (https://kubeless.io/docs/http-triggers/)

Unikernels

Aalto University
School of Electrical
Engineering

Current serverless implementation

- **Serverless functions actually run in containers**
  - Management and allocation done by FaaS

- **Can we do without containers?**

Unikernels

- **Pretty much cutting edge**
  - But look at AWS Nitro hardware architecture …

- **Idea: Use a separate virtual machine to handle each event**
  - Application packaged directly as a virtual machine image
  - Runs as a kernel in the VM

- **See http://unikernel.org/projects/**

Why and why not?

Pros

- Minimizes overhead
- Simplifies resource management
- Increased isolation
  - *Now across events!*
- It's cutting edge

Cons

- Most definitely cutting edge research area
  - *Lack of tooling*
  - *Lack of management tools*
  - *Maturity (not) of tools*
  - *No established practices*
- Unclear whether will bring any benefits
- VM boot delay
- Runtime initialization delay

Process (roughly)

1. **Write code**
2. **Compile**
3. **Package into binary**
4. **Deploy binary**

- **Need to abide by the calling convention of the runtime environment**
  - How are event data passed in? How does data go out?

- **Runtime environment**
  - Solo5
  - KVM/QEMU
- **Library selection**
  - MirageOS (OCaml!)
  - IncludeOS (C++)
  - LING (Erlang)

Summary

- **“Serverless” is an abstraction**
  - Functionally does not offer anything significantly new
  - But: can be easier to use and deploy; can be cheaper
- **The field of “serverless” still maturing**
  - Best practices not always established; not always fully supported by tooling
  - Tooling and frameworks churn
- **Just like using containers, requires some setup for local testing**
- **Unikernels cutting edge, but no clear benefits yet**