Cloud Computing and Services

**PREPARED FOR**

Erik van der Schriek

Niko Kuijpers

Gertjan Schouten

**PREPARED BY**

Maarten Blömer

Nick Krijgsman

Faruk Aydin

Jursley Gonzalez

Vincent Andersen

# Table of Contents

[**Table of Contents**](#_z6lhuq9emniv) **1**

[**Case Study**](#_k6jx82pctwds) **2**

[Case goal:](#_bpxkra1gvikb) 2

[Case problems:](#_tz830mgo6nzk) 2

[Case research questions:](#_3vn6lbsuph60) 2

[Brainstorming:](#_llbf4q7qnu37) 3

[**Independent study**](#_zc6h8lveooxw) **4**

[What is serverless architecture? (Vincent)](#_rix3b9o32hwo) 4

[Definition of serverless](#_6qo2kmwy5wmy) 4

[FaaS](#_xosfr4vusdku) 4

[Event-driven and scalability](#_1asscwfqhfab) 5

[Overall pros and cons](#_yeg1xxr5tc7) 5

[Pros](#_s3iwc51vbqa2) 5

[Cons](#_o1aum83czfd4) 5

[What functionality regarding cloud computing does the provider give and what do they offer in terms of serverless architecture?   
(Faruk en Nick)](#_fgxu9dbgfta9) 6

[Microsoft Azure (Faruk)](#_97ho52d59gzk) 6

[Pros](#_5r0nidergfcl) 6

[Cons](#_udn998km6f3a) 6

[Serverless](#_cbz05khzijb6) 7

[Security](#_ae66ivpxkpiw) 7

[Cost](#_79vu4wsnc5dy) 8

[Amazon AWS (Nick)](#_wl6nktm1y31l) 9

[Which guideline points of the 12-factor app are implemented and which points do we want to add to our application? (Maarten)](#_i5dfjhklpj2a) 11

[**Matrix**](#_cg2taz46zkvz) **13**

[**Sources**](#_qwibiw7iqp2l) **15**

# Case Study

## Case goal:

* Provide your team with a basic understanding of what cloud computing is and how it relates to earlier approaches of hardware virtualization.
* Gain understanding what the different scaling principles mean.
* Know the possibilities of services in modern cloud computing platforms and make a judgement which are of interest for this specific case.
* Learn what serverless cloud computing is and how it relates to the micro-service architectural pattern (differences, pros, cons?).
* Be aware of resource limitations and have some idea how to keep resource usage under control.

*You can now make an evaluation which cloud services can be meaningful to integrate in your system and how it fits into your architecture. You make a decision based on capabilities, pros,  
cons and the resource limitations that you have (e.g. maximum credits on the cloud platform).*

## Case problems:

1. Can cloud services like serverless be an option to replace our complete or part of our microservice architecture?
2. What are possible guidelines of best practices to keep our future cloud applications and services maintainable?
3. Some architects suggest 12-factor app guidelines should be followed. Should we?

Your task is to advise on a **suitable**, **reliable** and **secure** cloud platform, with a **high community acceptance** and **low cost** that can offer the necessary cloud services to **create and deploy a REST service** to validate IBANs with **fast response times** and **minimal resource usage**.   
You should motivate why the suggested services provided the right functionality, keep cost low and that your solution is able to handle the expected load. Furthermore, you should **include an estimation** of the **yearly costs** of the suggested solution.

## Case research questions:

1. What is serverless architecture?
2. What functionality regarding cloud computing does the provider give and what do they offer in terms of serverless architecture?
3. Which guideline points of the 12-factor app are implemented and which points do we want to add to our application?

## Brainstorming:

# Independent study

## What is serverless architecture? (Vincent)

### Definition of serverless

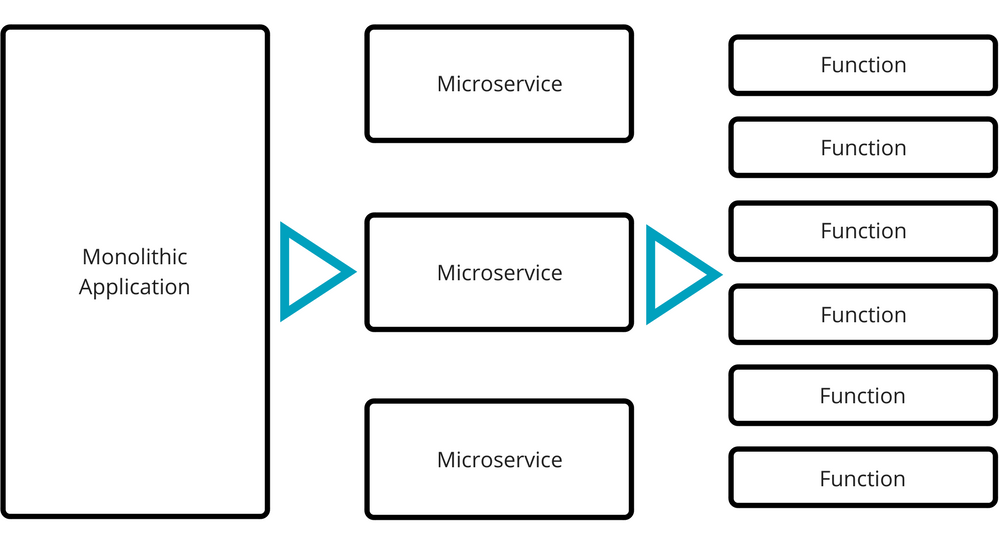
Serverless architecture in general means that you as the developer are not responsible for managing the servers and instead outsource it to the multiple different cloud providers. This brings the advantage that developers can simply focus on developing business logic.

The definition of serverless can come in different forms which encapsulate the aforementioned idea of serverless architecture but may implement it in different ways. Some names are:

1. Backend as a Service (BaaS)
2. Functions as a Service (FaaS)
3. Platform as a Service (PaaS)
4. Software as a Service (SaaS)
5. Infrastructure as a Service (SaaS)

### FaaS

FaaS is the most popular choice and the one that is the most reminiscent of a microservice architecture so we’ll be focussing more on this implementation in this document

When building something in the FaaS form, developers assemble code as much as possible in building blocks called functions. This way, the application will be split up into small functionality based elements that can be scaled and updated separately. This also makes each component much more manageable. While this is the part that looks like a microservice architecture, FaaS splits it up even further.  


*Example 1.0:  
The difference in implementation size between microservices and functions.*

### 

### Event-driven and scalability

The functions on the server side are only computed when activated which makes them event-driven. This means that serverless is more or less stateless. It does make them ideal for scaling applications as each of the functions run in seperate containers which allows you to scale them endlessly however you want. FaaS also handles inconsistent traffic very well as you per request instead of an auto scaling option which may not be feasible when the startup-time takes too long for it to miss parts of the high traffic.

Due to the free-tiers given out by some providers you may not even have to pay anything if your traffic is low.

## 

## Overall pros and cons

#### Pros

* Cheaper than the traditional cloud with pay per request
* Highly flexible and scalable
* Lower maintenance costs
* More focus on business logic for developers

#### Cons

* Locked in to a provider of your choice
* Steep learning curve
* Unsuitable for long term tasks

## What functionality regarding cloud computing does the provider give and what do they offer in terms of serverless architecture? (Faruk en Nick)

### Microsoft Azure (Faruk)

#### Pros

* High Availability

Azure offers a service level agreement, or SLA, of 99.95% (4.38 hours of downtime per year)

* Scalability

Azure has the ability to auto scale according to the demands of the application. Beside that, it’s also possible to easily scale up/down within a few clicks.

* Cost Effective

Azure’s pay-as-you-go pricing helps small/midsize businesses by only letting them pay for what they need. The cloud environment allows businesses to launch both customer applications and internal apps in the cloud, which saves on IT infrastructure costs while reducing the hardware and maintenance burdens on in-house IT management.

* Security

Azure offers a security control system based on the DADSC approach (Detect, Assess Diagnose, Stabilise, and Close). Azure offers strong protection against data loss. Not only is the platform protected, the end user is also covered with Azure. This multi-level of protection is essential.

#### Cons

* It can be hard to start with. There are some functions that need more expertise.
* It needs good management. This also requires more expertise. (someone familiar with Azure)

#### Serverless

The ability to build serverless apps within Azure is available. The azure functions can be used without any problems. It’s known to reduce some time on tasks that are non-core to the business.

#### Security

Products and services

* Security center

Integrate security management and advanced threat protection for various hybrid cloud workloads

* Application Gateway

Build secure, scalable, highly available web front ends in Azure

* Azure Active Directory

Sync on-premises folders and enable single sign-on

* Azure DDoS protection

Protects applications from DDoS (Distributed Denial-of-Service) attacks

* Key Vault

Securing and managing keys and other secrets

* Azure Information Protection

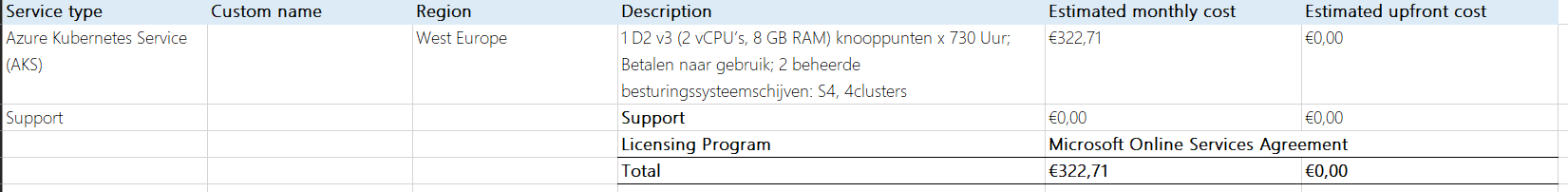
Protects sensitive information wherever and whenever you work

#### 

#### Cost

It’s really easy to create a scenario on Azure to know how much it will cost. I’ve created a scenario.

Scenario (see Description in image):



### 

### Amazon AWS (Nick)

**Pros:**

* High availability. Amazon offers a service level agreement, in which they will give discounts if their monthly uptime percentages are not above 99.99%. Between 99.99% and 99% you receive a 10% discount, 99% to 95% a 30% discount and if the uptime is lower than 95% you get refunded fully.
* Security. Amazon offers a wide variety of security services. Each with their own use case. Some categories of these services: Identity & access management, Detection, Infrastructure protection, Data protection, Incident response and Compliance & data privacy.
* Scaling. Amazon offers a service called AWS Auto Scaling. This service monitors your applications and automatically adjusts capacity to maintain steady, predictable performance at the lowest possible cost. Using AWS Auto Scaling, it’s easy to set up application scaling for multiple resources across multiple services in minutes. The service provides a simple, powerful user interface that lets you build scaling plans for resources like EC2 instances.

**Cons:**

* Billing is difficult to understand.
* EC2 limits; the resources are limited per region.

**Serverless**

Amazon offers a service called AWS Lambda. It is a serverless computing service. It works by uploading your code as a ZIP file or container image, and Lambda automatically and precisely allocates compute execution power and runs your code based on the incoming request or event, for any scale of traffic.

**Pricing**

Pricing is comparable to the services offered by Azure.



## Which guideline points of the 12-factor app are implemented and which points do we want to add to our application? (Maarten)

1. **Codebase**

The code must always be tracked in a version control system. In our case all our code is tracked on GitHub. Each microservice has its own repo.

1. **Dependencies**

A 12 factor app never relies on implicit existence of system-wide packages. All dependencies should be declared in a dependency declaration manifest such that no dependencies can leak in.

1. **Config**

The config is what is likely to vary between deploys, this includes things like resource handles to the database, and credentials to external services. Config varies substantially across deploys, code does not.

1. **Backing services**

A backing service in any service the app consumes over the network as part of its normal operation. Things like this are datastores, messaging/queueing systems. In our case we have MySQL running a user database and RabbitMQ running the messaging system.

1. **Build, release, run**

There should be strict separation between the build, release and run stages. The build stage takes the code and transforms it into an executable bundle. The release stage takes the build and combines with the config and finally the run stage runs the app in the execution environment.

1. **Processes**

Twelve-factor processes are stateless and share-nothing. It does not assume that anything cached in memory or on disk will be available on a future request or job. All local memory should be wiped between deploys.

1. **Port Binding**

The twelve-factor app is completely self-contained and does not rely on runtime injection of a webserver into the execution environment to create a web-facing service. The web app exports HTTP as a service by binding to a port and listening to requests coming in on that port.

1. **Concurrency**

In the twelve-factor app, processes are a first class citizen. Processes in the twelve-factor app take strong cues from the unix process model for running service daemons. HTTP requests may be handled by a web process, and long-running background tasks handled by a worker process. Async/evented model found in tools such as node.js. Horizontally partionable nature of the twelve-factor app process means that adding more concurrency is a simple and reliable operation. These apps should focus on information streams.

1. **Disposability**

Apps should be able to start and stop at a moment’s notice. This can easily be done via the hosting site to take down the web applications/services to perform maintenance.

1. **Dev/prod parity**

Keep development, staging and production as similar as possible. The app should be designed for continuous deployment.

1. **Logs**

A twelve-factor app never concerns itself with routing or storage of its output stream. It should not attempt to write to or manage log files. Each process should write the logs to stdout, into the terminal. During deployment it should be easy to find what happened in the past.

1. **Admin processes**

Admin tasks should be able to run against the release build for easy debugging and updates. Running commands in the console to target problems in the release build. Most importantly they need to use the same code base and config files.

# 

# Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| **Evaluation Matrix** | **Weight** | **Microsoft** | **Amazon** |
| **Availability** | 0.2 | **2** (10) | **2** (10) |
| **Scalability** | 0.2 | **1.6** (8) | **1.6** (8) |
| **Cost** | 0.2 | **1.8** (9) | **1.6** (8) |
| **Security** | 0.2 | **1.8** (9) | **1.8** (9) |
| **Complexity / Learning Curve** | 0.2 | **1.4** (7) | **1.4** (7) |
| Total score: | 1 | 8.6 (43) | 8.4 (42) |

**Advice**

Based on the [Matrix](#_ms14khqko54c) alone which scores the following frameworks as such:

|  |  |  |
| --- | --- | --- |
| **Platform** | Microsoft | **Amazon** |
| **Score** | 8.6 | **8.4** |

In terms of performance and cost, cloud services like services are an excellent way of handling the situation as seen in the case study, especially FaaS. Using FaaS serverless architecture will aid in handling inconsistent traffic and peak load times while keeping the cost as low as possible based on the request.  
One could argue that with 4 billion pin transactions the free-tier won’t be worth much as it will roughly average about 333 million requests a month(without factoring in low and peak times). So a calculation should be made using the azure calculator and the pricing of a server in order to find out which is more cost effective.   
On top of that, In the study case, the company must be aware that their transaction services are bound to the cloud provider and should be ready to comply with any changes regarding data that will be made within that company. Since the company will be handling IBAN transaction information they should think about the protection of data in the cloud and which provider to trust their data to.

Costs for the computing power of the 4 billion transactions. We are not sure on the time it takes and how much cpu it needs to complete the transaction. So we are taking some unegigated guesses. It takes a transaction 100ms to complete and needs 256Mb in resources. By entering this information in the Azure price calculator for functions, it returns a monthly cost of $193,53. Which leads to a yearly amount of $2322,36.

It is recommended to follow the 12 factor app guidelines. Our application currently follows most of these factors; to be more specific it follows the guidelines for: codebase, dependencies, config, backing services, build/release/run, processes, port binding, concurrency, disposability, logs, and admin processes**.** In addition to those guidelines we would like to focus a bit more on the dev/prod parity by improving our codebase on GitHub. There are still some repos that need to be structured in the correct fashion. Most of these twelve factors are standard practice when it comes to the microservice architecture and are overlooked during the building process. These are also interwoven with multiple best practices when using certain frameworks within the multi-repo microservice architecture.

# Sources

|  |  |
| --- | --- |
| **Student** | **Source** |
| Vincent | <https://martinfowler.com/articles/serverless.html> |
| Vincent | <https://www.altexsoft.com/blog/cloud/pros-and-cons-of-serverless-architecture/?utm_source=DZone&utm_medium=referral> |
| Vincent | <https://www.densify.com/articles/autoscaling> |
| Vincent | <https://blog.symphonia.io/posts/2017-11-16_learning-lambda-part-9#:~:text=Lambda%20will%20horizontally%20scale%20precisely,it%20to%20a%20massive%20extent.&text=However%20if%20hundreds%20of%20concurrent,automatically%20to%20support%20that%20load>. |
| Vincent | <https://docs.microsoft.com/en-us/azure/azure-functions/> |
| Nick | <https://www.simform.com/compute-pricing-comparison-aws-azure-googlecloud/> |
| Nick | <https://aws.amazon.com/> |
| Nick | <https://sados.com/blog/aws-benefits-and-drawbacks/> |
| Nick | <https://www.master-it.nl/p/blog/microsoft-azure-vs-aws-welk-cloudserviceplatform-kies-jij> |
| Maarten | <https://12factor.net/> |
| Faruk | <https://searchcloudcomputing.techtarget.com/definition/Windows-Azure> |
| Faruk | <https://docs.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-overview> |
| Faruk | <https://www.tutorialspoint.com/microsoft_azure/microsoft_azure_scalability.htm> |
| Faruk | <https://azure.microsoft.com/nl-nl/> |
| Faruk | <https://azure.microsoft.com/en-us/pricing/calculator/> |
| Faruk | <https://mymatetech.net/how-secure-is-azure-276fbce61155> |
| Faruk | <https://cloudbusiness.com/how-secure-is-azure/#:~:text=Automatic%20encryption.,Xbox%20and%20Microsoft's%20Office%20365>. |
| Faruk | <https://azure.microsoft.com/nl-nl/overview/security/> |
| Faruk | <https://azure.microsoft.com/nl-nl/global-infrastructure/availability-zones/#overview> |
| Faruk | <https://cogitogroup.net/pros-and-cons-of-azure/> |