Universiteit Leiden
The Netherlands

Bachelor Computer Science

Scalable architecture for event management systems

Patrick Bergman

Supervisors:

Prof. dr. J.N. Kok & Dr. A.-W. de Leeuw

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

16 May 2018

Abstract

For the Live I Live festival in 2015, an event management system was used to monitor the safety of the event. The system had several dashboard views, with busy indicators for podium locations and train stations. Also some social media was monitored and accessible from a central point where the event organisation could monitor the event. The local police could access the dashboard as well.

As the old architecture was rather static with 2 servers and a database, a new architecture is required to accomodate other events. The proposed architecture is "serverless", which basically means that every action, like 'fetching twitter messages with a certain tag' or 'fetching data from a wifi point to get data about the near crowd', is a separate function that can be called whenever needed. The architecture scales automatically when the demand increases. Also, the system owner will only be billed when the functions are used and in rest (or non-event days) the system functions has no costs. This is an advantage compared to the old architecture, since this has montly costs to keep it running.

Acknowledgements

First of all, I thoroughly want to thank Arie-Willem de Leeuw for his support and willingness to guide me with this process for the whole run. Also, I want to thank him for the final assessement of this thesis. Secondly, I want to thank Joost Kok for bringing me in contact with Event Cloud and this thesis subject. Another person I want to thank is Rick van Oosterhout of Event Cloud. He provided the contacts to get more information about the thesis subject and his use-case of Event Cloud. Finally, I want to thank my family for the support and faith in me that, eventually, I will complete this thesis.

Contents

Chapter 1

Introduction

1.1 Event Security with data

Events in The Netherlands are growing year after year. According to 'Respons', in 2015 the amount of events and festivals increased by 4.5% compared to 2014. In total there were 23 milion visitors [3]. The largest festival is the Amsterdam Dance Event (ADE) with 395 thousand visitors in a single weekend across 160 locations in Amsterdam [4]. Successful events boost the local economy and the image of a municipality. This is good for local entrepreneurs, the municipality and the event visitors [2].

For large events, the growing number of visitors and shifting responsibilities from the municipality to the event organization are causing pressure on the event organizations. This leads to increased costs and risks in the area of safety during events. Also the current economic climate (now being on the wane) and the decrease in sponsorships are not in favor of the events. Therefore, it is getting harder to organize large events.

The amount of regulations are increasing to prevent incidents like 'Pukkelpop', 'Love Parade' and the event in 'Haaksebergen'. To increase the safety of an event, it is important to gather a lot of information from several data sources. This data can be collected up front or during the event. A few examples are:

- Weather data
- Public transport data
- Social Media data

This data needs to be analysed by the organization, emergency services and security to ensure a safe event. Especially, combining the different sources can increase the event safety a lot. Currently, there is not a standardized platform or architecture that bundles these data sources and provides insight in the event and especially the behavior of the event visitors.

The main concern of such a platform is that it must be able to run anywhere without major complications. A well architected, scalable framework for such a system is needed to ensure continuity on multiple events across the country.

1.2 Event Cloud

Event Cloud is a system developed by Rick van Oosterhout and his company Prooost and was used for event management and crowd control during Live I life 2015 [1]. The purpose of Event Cloud was to create an overview of communication and crowd density for the event management team to secure the event. A large communication scheme was created to manage the communication during the event. This scheme is displayed in figure 1.1

In order to get a more detailed overview of the crowd during the event, Event Cloud was created. Event Cloud provided different dashboards for the central post:

- Mood reports supplied by security guards on the ground with an app [Appendix A: figure:1]
- Crowd density reports supplied by security guards on the ground with an app [Appendix A: figure:2]
- Tracked data of the visitors through the visitors app [Appendix A: figure:3]
- Tweets from twitter with certain hash tags [Appendix A: figure:4]
- Images from social media with certain hash tags [Appendix A: figure:4]
- Weather forecast [Appendix A: figure:5]
- Crowd density on train stations provided by people on the stations with an app [Appendix A: figure:6]

[1] All information is retrieved from him and co-workers during sessions in the building of the The Hague Security Delta located at Wilhelmina van Pruisenweg 104, 2595 AN Den Haag

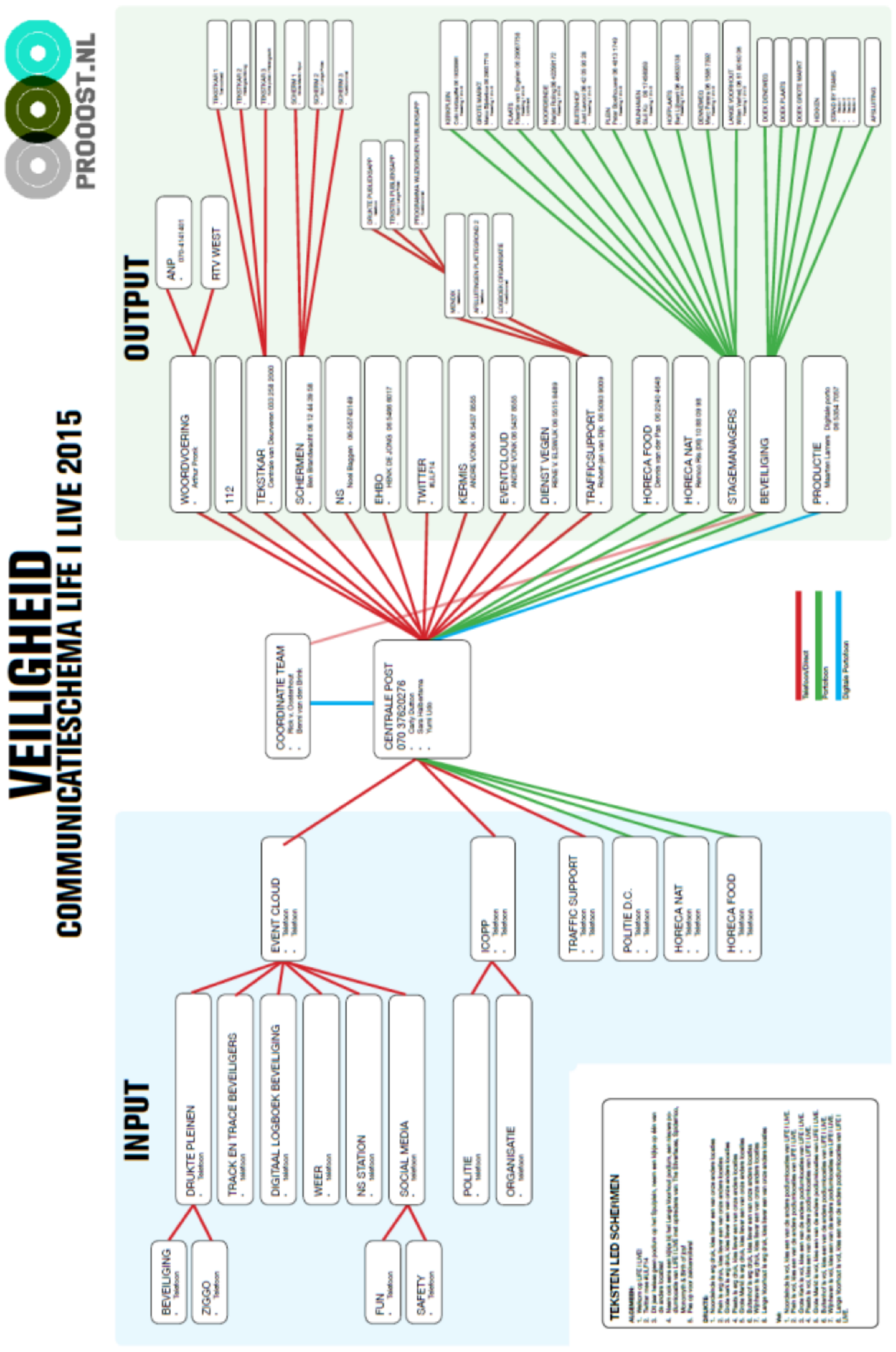


Figure 1.1: Communication schema Live I Live 2015, multiple data sources combining to a single dashboard with multiple output possibilities for crowd control

1.3 Thesis overview

This thesis will start with discussing scalable architectures and assessment criteria for server architectures in chapter 2. Furthermore, the ”old” architecture will be summarized and weight with the assessment criteria. Next, in chapter 3 we discuss new scalable architectures and assess them with the same criteria as defined in chapter 2. Specifically, we consider an upgrade of the ”old” architecture and a new type of serverless architecture. Thereafter, in chapter 4 a new ’scalable’ architecture will be introduced and assessed. Next, in chapter 5 we discuss the two new architectures. Finally, we end with a conclusion in chapter 6.

Chapter 2

Scalable architectures

This chapter *will give an overview of some options that are currently viable and widely used as a server architecture. Next, scalability of architectures and infrastructure will be highlighted. Furthermore, we will enclose assessment criteria to weigh architectures. In this chapter, we will introduce the assessment criteria that are used in the following chapters to assess new architectures. In particular, we focus on the AWS Well Architected Framework paper and the therein criteria.*

2.1 Server architectures in general

When people see websites or web applications, they don't think about how the architecture of the servers is managed. A company or private person has many options for hosting their web applications. In this thesis, we will highlight a couple of popular solutions, such as shared hosting and single servers. Other options are multiple app servers with a load balancer or multiple app servers with multiple database servers with a load balancers. The latter can be combined with a Floating IP and more loadbalancers. A relative new solutions is the so-called serverless framework.

Shared Hosting
Shared hosting is the budget option, that is extremely cheap, but not very good. A server is essentially a computer with a CPU, RAM and disk space (either an HHD, SSD or a hybrid). In a shared hosting environment, the host puts a large number (> 1000) of users on a single server. Since each user may have several sites, and share the same server resources, this can lead to problems. For example, if one website out of a 1000, has some faulty code or simply gets a lot of visitors, it could use 75% of a server's memory for example, which means that the other 999 sites are left with 25% of the total RAM. More details can be found in Ref. [9] A diagram of this type of hosting can be found in appendix B, figure 7.

Single Server
Hosting a web application is also possible on a single server, which can either be a Virtual Private Server (VPS) or a barebone server. VPS is probably the most popular service and it can be the most well-balanced one as

well. A VPS server is still a shared environment, but the way it is shared is very different compared to shared hosting. First of all, a VPS server is usually limited to 10-20 virtual servers on a single instance. This decreases stress in itself, but the real improvement comes in the form of the hypervisor. A VPS server is literally split into as many parts as there are users. If there are 10 users, 10 GB of RAM and 200 GB of hard drive spave on the server, each user will be able expend 1 GB of RAM and 20 GB of space. Once you hit the RAM limit your site may go down, but the others will remain stable. This simple addition removes almost all of the bad neighbor effects. In certain extreme situations, there can be some issues which affect multiple users through the hypervisor itself, but for most practical purposes this can be ignored. For more detailed information, see Ref. [9]

A bare-bone server is the same as a VPS, but all the resources of the server only belong to the owner and are not shared or partitioned.

Load Balancer with multiple app servers
A load balancer can be added to a server environment to improve performance and reliability by distributing the workload across multiple servers. If in this case one of the servers fails, the other servers will handle the incoming traffic until the failed server becomes healthy again. For more details, see Ref. [8].

This type of hosting will increase the security of hosting a web application for 2 reasons. First, only the load balancer is exposed to the Internet. Secondly, the application servers and database server are in a private network which can only be accessed by the load balancer.

Load Balancer with multiple app and database servers
This configuration is essentially the same as the previous option. The only difference is that there are now multiple database servers which lowers the traffic and load on a single database server. This gives the disadvantage that the database server has the highest load with many requests from the different application servers. Another disadvantage of this architecture is that data must be equal if it is accessed in different databases. Possible solutions for this phenomenon are a master-slave configuration, or a configuration with a single database the accepts insert queries and multiple database servers that accept reading queries. However, this can delay the reading of the just submitted data.

Floating IP with multiple Load Balancers and multiple app and database servers
A Floating IP is a publicly-accessible static IP address that you can assign to one or multiple servers. The instant remapping capability of Floating IPs allows to add redundancy to the entry point, or gateway, to servers. This is a key component in a high-available server infrastructure, which is a setup with no single points of failure. A complete HA setup requires redundancy at every layer of your infrastructure, including your application and database servers [7].

Serverless architecture

Whith this architecturing the concept of building and running applications does not involve server management. Applications, bundled as one or more functions, are uploaded to a platform to be executed, scaled and billed on the exact demand. This results in a finer-grained deployment model [10].

Hosting and running code still use servers in serverless computing. Moreover, operation engineers are stull required. However, time spend on server provisioning, maintenance, updates, scaling, and capacity planning is greatly reduced for consumers of serverless computing. Instead, the serverless platform handles all these tasks and capabilities. They are completely abstracted from the developers and IT/operations teams. As a result, developers can focus on writing their applications' business logic. The focus of operation engineers can now be directed to business critical tasks. [10].

An overview of the above mentioned types of architectures are schematically displayed in appendix B.

2.2 Scalability

Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth [11]. A system can be seen scalable if it manages to handle increasing amount of processes being executed, while automatically increasing the hardware to handle all processes. That also implicates that the system can automatically scale down, when the number of processes decreases. There are a number of ways to provide scalability to a system:

- Administrative scalability: *"The ability for an increasing number of organizations or users to easily share a single distributed system [13]."*
- Functional scalability: *"The ability to enhance the system by adding new functionality at minimal effort [13]."*
- Geographic scalability: *"The ability to maintain performance, usefulness, or usability regardless of expansion from concentration in a local area to a more distributed geographic pattern [13]."*
- Load scalability: *"The ability for a distributed system to easily expand and contract its resource pool to accommodate heavier or lighter loads or number of inputs. Alternatively, the ease with which a system or component can be modified, added, or removed, to accommodate changing load [13]."*
- Generation scalability: *"The ability of a system to scale up by using new generations of components [13]."*
- Horizontal and vertical scalability: *"Either the ability to have more servers handling multiple systems, or spreading responsibilities (like databases and cache), or a combination of both [13]."*

2.3 Assessment criteria

To fairly compare different server architectures a set of criteria must be established. Since every framework, architectures, infrastructure or hosting is different and there are multiple providers and services, this can be quite tedious. You have larger ones like Amazon Web Services, Microsoft Azure, Google Cloud, Digital Ocean or smaller ones like the Leiden based company TransIP.

In the search of criteria that can be used for assessing server architectures, we use a the criteria that are set by Amazon in the so-called "AWS Well-Architected Framework" [6]. In this paper, Amazon has defined how a well-architected framework must be designed and assessed. Since they are one of the largest providers of server space and services to host web applications and more, they are most likely to have a lot of experience architecting the web. With clients as Netflix, Expedia and Philips they certainly qualify as a reliable source for architecting large and small frameworks. Thus, in this thesis will use the "AWS Well-Architected Framework" paper as a guideline for assessing architectures.

2.4 AWS Well-Architected Framework and defined assessment criteria

AWS is a cloud service relaesed by Amazon.com Inc (Amazon). in 2006. Amazon released an Application Programming Interface (API) for highly scalable, reliable, low-latency storage called S3, which is basically online storage [1]. Today, AWS has over 70 services providing large computing capacity that is quicker and cheaper than a client company building an actual physical server farm [5].

Basically, the AWS Well-Architected Framework described by Amazon is based on 5 pillars: security, reliability, performance efficiency, cost optimization, and operational excellence [6]. The documented framework has a general set of design principles to facilitate a good design in the cloud: [6]

- "Stop guessing your capacity needs"
- "Test systems at production scale"
- "Automate to make architectural experimentation easier"
- "Allow for evolutionary architectures"
- "Data-Driven architectures"
- "Improve through game days"

These principles are confided in the five pillars where the AWS Well-Architected Framework is based on, Operational Excellence, Security, Reliability, Performance efficiency and Cost optimization. All these pillars have several questions, which we will use to assess the different alternative 'new' frameworks. These questions can be found in the whitepaper from Amazon Web Services [6].

2.4.1 Summary

This thesis will not answer all questions in detail, since it is a comparison using the "AWS Well-Architected Framework" as a guideline. However, we will answer all questions with the knowledge we have about the "old" architecture and make a similar assessment to the new proposed architectures. To use these criteria for other hosting providers then AWS, we need slightly more generalised criteria then mentioned in the previous sections. Therefore, the above mentioned questions are slightly altered before using them to assess the new architectures.

Chapter 3

The current "old" architecture

In this chapter *we will describe an overview of the "old" architecture used by Event Cloud in 2015. Moreover, we will use the guidelines and questions as provided in the "AWS Well-Architected Framework" to assess the architecture.*

3.1 Overview

First, we will describe and review an existing architecture for events. We will consider an architecture called Event Cloud. This system is specifically developed for the Life I Live festival in 2015. This event was visited by around 200.000 visitors and contained 8 podiums in the city center of The Hague.
Because of the free access, alcohol being served and a broad range of target audience, this event was classified as a high risk event. Therefore, the organizer - PROOOST - thought of a system to manage the crowd and provide them fun at the same time.

The main focus points of the system were: crowd density, social media monitoring and usage, public transport crowd density, weather forecasts and logging all communication and events. Eventually, the input of the systems consists of 2 apps for smart-phones, an app for the visitors and an app used by security guards. Furthermore, an event dashboard was build on QlickView.
Moreover, the real-time location of the security guard was determined via a security app. Additionally, the guard was asked to provide information about the crowd density and the current ambiance of his current location. If the guarded felt the urge to provide extra information, he could add some textual content as well.
The visitor app had a different purpose. The app had 4 options: a program overview, personal routing and locations, find your friends and find your music. The program overview contained a detailed schedule of the event, e.g., who performs on which stage. This section was retrieved from the system and could be altered by the organizer to better manage the crowd. The second option allowed visitors to create their route by selecting the performers they want to see. The third option was meant to find your friends through social media and location based access, so the visitor and their friend (who also enabled the option) could locate each other. The

last option was a combination of the event program and a visitors Spotify playlist. The combination was used to create a personalized program for the visitor.

All this information had to be processed, stored, secured and distributed. Due to time constraints, the developer decided to combine existing technologies. Although a detailed description of the architecture is not available, we can provide a rough sketch of the main properties of the architecture.

3.1.1 Architecture

As mentioned in the previous part of this chapter, there were 2 apps communicating with the back-end of the application and there was a dashboard for the organizers on which they monitored social media, weather, public transport and crowd density. All the data was stored and processed on servers in the KPN Cloud with the help of a PostgreSQL database. With the data from this database, the dashboard was generated by an application called QlikView. The 2 applications retrieved their data from app servers hosted on the KPN cloud. If an app needed to store data, this was also stored in the PostgreSQL database.
Because the developers did not know how many people were going to use the app, they made sure the servers were automatically scaled based on usage.

3.2 Assessment

In this section the ″old″ framework is assessed using the pilars of the ″AWS Well-Architected Framework″. The subsection will try to give answers to the questions that are in chapter 2.

3.2.1 Operational Excellence

System reliability is the most important operational priority, since the systems must always be available during an event. Although the design of the system was a guess at forehand, this was ensured by the hosting provider KPN. However, the systems were not tested on large workloads. The designer/builder of the system was monitoring the systems operational health during the event. There were no other monitoring services active to monitor the operations and the system was not designed to evolve.

3.2.2 Security

To review the security of the system, we have to consider identity and access management, detective controls, infrastructure protection, data protection and incident response.

Since the previous system was a trail and needed to be quickly setup, there was no protocol for identity and access management. However, the access to the system was limited to a single company and one person. Moreover, only one person could make changes to the architecture whenever necessary. According to the AWS Well Architected Framework, it is better to define roles and responsibilities for system users to control access. Also, there should be protocols to limit automated access to resources like applications, scripts and/or third-party tools or services.

Another security item are the detective controls. Although in Event Cloud the logs were captured, they were generated on user events and not as a basic setting. It is advisable to capture logs for all events and analyse them, preferable live.

Infrastructure was important to secure the protection of the servers and services. Since this is managed by KPN and their KPN cloud services, this responsibility lies at the company providing the app servers, the PostgreSQL service and the QlikView service.

Data protection was enabled by using SSL certificates to encrypt the traffic between the servers and the security and visitors apps. There were no policies for encrypting and protecting the data and there is no clear management of keys to the data, though.

Regarding incident management, this was not taken into account. Although they made sure the servers were scalable to handle larger amounts of data.

3.2.3 Reliability

The network is setup as a partially connected mesh framework [1]. The main focus of the initial project was to setup a network quickly and only use connections whenever needed, not planning for service limits or a plan on network topology.

Change management is accounted for by using applications servers that will scale on demand. Although this process must be triggered manually, it is made possible to scale up the servers when necessary.

Failure management is an important subject related to reliability. Since the physical management of the servers lies at KPN, it is not necessary to manage that. However, a plan to recover the architecture should be included in case of a hardware or software failure. Although this is not the case, KPN does have backup systems that will automatically take over failing hardware.

3.2.4 Performance efficiency

Performance efficiency is categorized in the selection, reviewing and monitoring of the used architecture as well as the tradeoffs that are made to improve performance.

[1]`https://en.wikipedia.org/wiki/Network_topology#/media/File:NetworkTopology-Mesh.svg`

The best way to determine a good or best performing architecture is to use a data-driven approach. For the version of Event Cloud that is under review. There were no clear choices made for a best performing architecture, compute solution, storage solution, database solution or network solution. The choices for this architecture were primarily made by the partner provider KPN, since there was insufficient time to over think those decisions.
There were no specific monitoring systems in place and therefore KPN performed the monitoring by themselves.

3.2.5 Cost optimization

Since KPN was a partner of the project, there were no other considerations for a service provider. At forehand, the number of visitors was an educated guess based on previous editions of the event. Therefore, it was possible to make a good estimation of the costs. Since the system was part of the event, there was no pricing strategy applied.

Summarizing: The costs were known before the event started. There was no pricing policy, because the customer is the builder.

3.2.6 Summary

For a quick set-up of a not yet created system, this was a really good effort. The system operated during the event and KPN made sure the infrastructure was solid and accessible. However, this architecture is not build to be scalable to multiple events or to be re-used for new editions of the same event. Another miss is the monitoring of the systems with logs and other services to analyse all the data.

Chapter 4

Assessing a ″new″ architecture

***In this chapter** we will discuss a new type of architecture called a 'serverless'-architecture. This architecture does not relay on 'old-school' static servers, but on functions that will be used when needed.*

4.1 A new scalable ″serverless″ architecture

As discussed in the previous chapter, the old architecture has the disadvantages that it is static and missing extensive logging. Here, we propose to work with a ″serverless″ architecture. The system has the advantage that it is more flexible, less sensitive to errors and the data storage is better accessible. Better, it will be more error proof. Also data storage will be better accessible, while having more databases give the advantage that in the case of a failing database, there are automatically multiple copies available.

4.1.1 Overview

The name 'serverless architecture' does not mean there are no servers, just like wireless internet, which also does not mean there are no wires [12]. As a technology, you can actually spend coding all day, instead of managing all side tasks, like setting up servers [12]. This means that the focus is on developing the system/application. There is no need for server management, scaling the serverpark or paying for unused resources.

The system consists of multiple services that are working with the coded functions on the serverless framework. Examples of these services are databases, queues, SMS messaging and email delivery [12]. The code can access all services when needed.

A serverless application has 5 benefits over more static architectures. First, it has zero administration. We can deploy code without provisioning anything beforehand. Second, there are no instances or a fleet of servers. There event isn't an operating system that needs to be managed. A third benefit is the auto-scaling. The service provider manages the scaling challenges, and there is no need to fire alerts or write scripts for scaling. The fourth advantage is the possibility to pay-per-use. When the functions are not used, there are no costs involved. This can give cost-savings up to 90% compared to a regular server or VPS. The last benefit is the increased velocity. There will be less time between the idea and deploying to production. This is due to the fact there is less provisioning up front and management after deployment is a breeze. For more information about the benefits for serverless architectures, see Ref. [12].

A schematic overview of a serverless architecture is displayed in figure 12.

4.1.2 Assessment

In this section the "serverless" framework is assessed using the pillars of the "AWS Well-Architected Framework". The subsection will try to give answers to the questions as mentioned in chapter 2.

Operational Excellence

To ensure operability, the whole system must be developed with shared design standards and isolated tests, e.g., each serverless function must be tested before it's deployed. Besides testing each individual function separately, the whole system must have an integration test and a load test to simulate the high demand during an event. By defining an expected behavior and identifying success, workload and operations metrics, operational health can be monitored. Operational events and the corresponding responses must be predefined in a operational script. After an event, all operational events must be reviewed to evolve the event management system as a whole.

Security

Root accounts are similar to the admin accounts of one's personal computer, you can do about everything with these credentials. In practice, it is best to avoid using root credentials. Furthermore, each user or group with roles must have the minimum amount of privileges that are necessary for accomplishing their tasks. When credentials are created, use static credentials for automatic access, this is preferable and they should be stored securely. To make the system more secure, logs must be captured and analysed. There are many great solutions for even more type of systems and languages. Another security measurement is to control network traffic e.g. the use of firewalls and security groups. But not only the application and system must be monitored, also the OS must be part of the security monitoring by ensuring file integrity.

Besides securing the system and including software with monitoring, data safety is another important aspect. Data that will be stored must be classified by a data classification scheme[1]. It would be even more secure to encrypt the data. Encrypting data in transit is possible by using an HTTPS connection on all event between servers. However, in our case most of the data is stored behind multiple security layers.

[1]https://www.cmu.edu/iso/governance/guidelines/data-classification.html

Reliability

To keep the systems running, the account limits should be below a certain threshold value. Therefore, a service that automatically monitors these limits and notifies the owner of an account that is almost crossing the service limits is a must. As can be seen in figure 12, the network topology is structured and changes in demand will not effect the system performance. New servers will be activated when the demand increases. The systems must be monitored with a tool like Amazon CloudWatch, which will send notifications when significant events occur.

We need an automatic process that automatically tests and deploys code changes, while backing-up the old code and data. By default, the backups and data must be stored by default on different locations.

The advantage of a serverless system is that when one of the components fails, new systems will take over the role of the broken components. This also happens in case a disaster , e.g., fires or overheated physical servers.

Performance efficiency

After choosing the architecture, the next step is to set a benchmark for the scripting language. Hereby, it is important to take into account the preferences of the provider of the architecture. Since the system is going to store a lot of data, a scalable storage and database solution is required. Amazon S3 buckets or Digital Ocean Spaces meet these requirements and are therefore possible options for a scalable database solution with many database servers.

This system will be primarily used in The Netherlands. Thus, to minimise the amount of network delay, the servers must be located in or near The Netherlands. After each event, the performance must be reviewed to ensure the best solutions are still used. There are many third party services available to monitor the performance of CPU, disk, database and network performances.

Cost optimization

To optimize the costs needed to run a serverless architecture, one must analyse services to reduce the overall costs. The total costs strongly depends on the type of events. Namely, large events gather a lot of data, while smaller events have a smaller dataset en thus require less computing power. To reduce the overall costs, a serverless architecture will only be invoiced when the serverless framework is used. This that in rest no costs will be charged. An important factor that also needs to be considered is data transfer, especially when storing or retrieving data files. In most cases, this is a cost that is overseen. During events, the functionality and costs should be monitored together. Then, decisions can be made if the costs exceed the budget that is available. Such that when the costs will exceed the expected costs, actions can be made to reduce the usage.

Thus, one should only add additional services if they are really necessary and not too expensive.

Chapter 5

Discussion

Due to a lack of available evaluation methods, we have only used the AWS Well-Architected Framework for evaluating a scalable framework. Since the used evaluation method is really extensive, we have only considered the most promising type of architecture regarding scalability and costs, e.g., a serverless framework.

The most important reason for choosing a serverless-framework is the short time that is needed to create a working product. However, a developer must be acquainted with this type of architecture to make a quick start. We think this type of architecture meets all necessary requirements. It provides a way to quickly create isolated, relevant and testable functions where the logic and UI are separated. Also only a single developer is needed to create functions, in one of the supported languages, and there is no need for server knowledge (DevOps).

Hopefully, in the near future more evaluation methods will become available.

As an alternative, it is also possible to start with a single server. This is easier to begin with, but scaling can become an issue in case you want an architecture as displayed in figure 11.

Chapter 6

Conclusions

In this thesis, we have investigated scalable architectures for event security systems. First, we evaluated the Event Cloud system that was used during the Life I Live 2015 edition. Thereafter, an upgrade for the aforementioned system is described and assessed. Finally, we introduced a completely new architecture which is 'serverless'. This serverless architecture was also assessed with the same criteria as the old architecture. A serverless architecture has multiple advantages. First, the costs are low because of the incidental usage of the system (only when an event is happening). Second, the single functions are also an upside of such an architecture. They are testable in an isolated environment and when it fails, you know exactly which function it is. Lastly, the biggest advantage of this architecture is the automatic scalability. When a function is required very often, more instances for the specific function will be booted up to handle the requests. Since you only pay for what you use, all costs are accountable. A large disadvantage is the greater start-up time of a function. Since a new instance has the be booted up, the start time of a function can be up to 2 seconds. After that, the instance will be available for some time and when needed, the function can be executed directly. For an event management system, that requires a continues stream of data from different resources, this is no problem.

The only thing that is left, is building a small system that must be tested in real life. Since this thesis is only an exploitative research on scalable event management architecture, it is too early to determine if the proposed architecture will be the most effective one. Since the system does not has to be operational each day, it shows potential. However, the developer(s) still that will build the system must get acquainted with the technology.

Bibliography

[1] Amazon.com Inc., *Amazon Web Services Launches*. Seattle, 14th of March 2006. (http://phx.corporate-ir.net/phoenix.zhtml?c=176060&p=irol-newsArticle&ID=830816)

[2] Eventbranche.nl, *Nederlands onderzoek: meer eventbudget, groei events zet door!*. 19th of december 2016. (http://www.eventbranche.nl/nieuws/nederlands-onderzoek-meer-eventbudget-groei-events-zet-door-11958.html)

[3] Respons, *Festivalsector trekt recordaantal bezoeken in 2015*. 25th of july 2016. (http://www.respons.nl/nieuws/festivalsector-trekt-recordaantal-bezoeken)

[4] Het Parool, *Amsterdam Dance Event 2017 breekt bezoekersrecord*. 23th of october 2017. (https://www.parool.nl/stadsgids/amsterdam-dance-event-2017-breekt-bezoekersrecord a4501855/)

[5] Amazon.com Inc., *What is cloud computing?* Retrieved 9th of May 2017. (https://aws.amazon.com/what-is-cloud-computing/)

[6] Amazon.com Inc., *AWS Well-Architected Framework* November 2017. (https://do.awsstatic.com/whitepapers/architecture/AWS_Well-Architected_Framework.pdf)

[7] Melissa Anderson, Digital Ocean *How To Use Floating IPs on DigitalOcean* 18th of April 2018. (https://www.digitalocean.com/community/tutorials/how-to-use-floating-ips-on-digitalocean)

[8] Mitchell Anicas, Digital Ocean *5 Common Server Setups For Your Web Application* 25th of May 2014. (https://www.digitalocean.com/community/tutorials/5-common-server-setups-for-your-web-application)

[9] John Stevens, *Different types of web hosting* 1st of January 2016. (https://hostingfacts.com/different-types-of-web-hosting/)

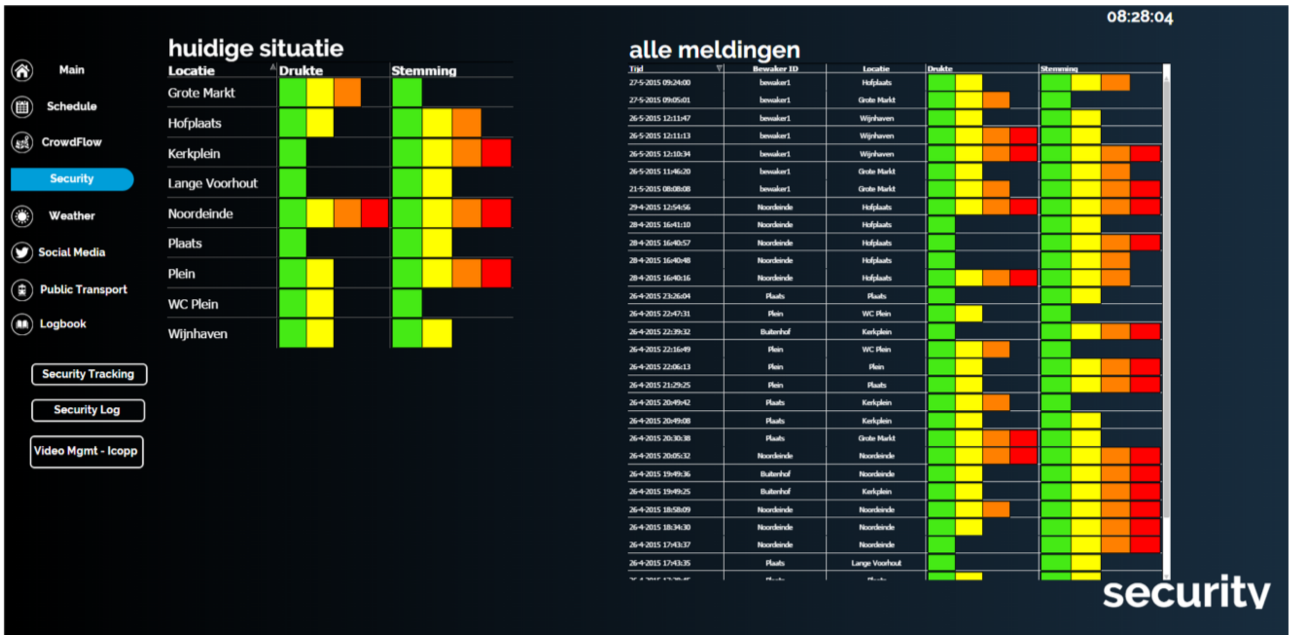
[10] Ken Owens, et al. *CNCF Serverless Whitepaper v1.0* March 2018 (https://github.com/cncf/wg-serverless/tree/master/whitepaper)

[11] André B. Bondi *Characteristics of scalability and their impact on performance* 2000 Proceedings of the second international workshop on Software and performance – WOSP '00. p. 195.

[12] The Serverless Framework *Why Serverless?* June 2018 (https://serverless.com/learn/)

[13] Wikipedia *Scalability* June 2018 (https://en.wikipedia.org/wiki/Scalability)

Appendix A: Dashboard views of Life I Live 2015



huidige situatie

Locatie	Drukte	Stemming
Grote Markt	🟩🟨🟧	🟩
Hofplaats	🟩🟨	🟩🟨🟧
Kerkplein	🟩	🟩🟨🟧🟥
Lange Voorhout	🟩	🟩🟨
Noordeinde	🟩🟨🟧🟥	🟩🟨🟧🟥
Plaats	🟩	🟩🟨
Plein	🟩🟨	🟩🟨🟧🟥
WC Plein	🟩🟨	🟩
Wijnhaven	🟩🟨	🟩🟨

alle meldingen

Tijd	Bewaker ID	Locatie	Drukte	Stemming
27-5-2015 09:24:00	bewaker1	Hofplaats	🟩🟨	🟩🟨🟧
27-5-2015 09:05:01	bewaker1	Grote Markt	🟩🟨🟧	🟩
26-5-2015 12:11:47	bewaker1	Wijnhaven	🟩🟨	🟩🟨
26-5-2015 12:11:13	bewaker1	Wijnhaven	🟩🟨🟧🟥	🟩🟨
26-5-2015 12:10:34	bewaker1	Wijnhaven	🟩🟨🟧🟥	🟩🟨🟧🟥
26-5-2015 11:46:20	bewaker1	Grote Markt	🟩🟨	🟩🟨🟧
21-5-2015 08:08:08	bewaker1	Grote Markt	🟩🟨🟧	🟩🟨🟧🟥
29-4-2015 12:54:56	Noordeinde	Hofplaats	🟩🟨🟧🟥	🟩🟨🟧🟥
28-4-2015 16:41:10	Noordeinde	Hofplaats	🟩	🟩🟨
28-4-2015 16:40:57	Noordeinde	Hofplaats	🟩	🟩🟨🟧🟥
28-4-2015 16:40:48	Noordeinde	Hofplaats	🟩	🟩🟨🟧
28-4-2015 16:40:16	Noordeinde	Hofplaats	🟩🟨🟧🟥	🟩🟨🟧
26-4-2015 23:26:04	Plaats	Plaats	🟩	🟩🟨
26-4-2015 22:47:31	Plein	WC Plein	🟩🟨	🟩
26-4-2015 22:39:32	Buitenhof	Kerkplein	🟩	🟩🟨🟧🟥
26-4-2015 22:16:49	Plein	WC Plein	🟩🟨🟧	🟩
26-4-2015 22:06:13	Plein	Plein	🟩🟨	🟩🟨🟧🟥
26-4-2015 21:29:25	Plein	Plaats	🟩🟨	🟩🟨🟧🟥
26-4-2015 20:49:42	Plaats	Kerkplein	🟩🟨🟧	🟩
26-4-2015 20:49:08	Plaats	Kerkplein	🟩🟨	🟩🟨
26-4-2015 20:30:38	Plaats	Grote Markt	🟩🟨🟧🟥	🟩🟨
26-4-2015 20:05:32	Noordeinde	Noordeinde	🟩🟨🟧🟥	🟩🟨🟧🟥
26-4-2015 19:49:36	Buitenhof	Noordeinde	🟩🟨	🟩🟨🟧🟥
26-4-2015 19:49:25	Buitenhof	Kerkplein	🟩🟨	🟩🟨🟧🟥
26-4-2015 18:58:09	Noordeinde	Noordeinde	🟩🟨🟧	🟩🟨🟧🟥
26-4-2015 18:34:30	Noordeinde	Noordeinde	🟩🟨	🟩🟨🟧🟥
26-4-2015 17:43:37	Noordeinde	Noordeinde	🟩	🟩🟨🟧🟥
26-4-2015 17:43:35	Plaats	Lange Voorhout	🟩	🟩🟨

Figure 1: List of reports by security guards

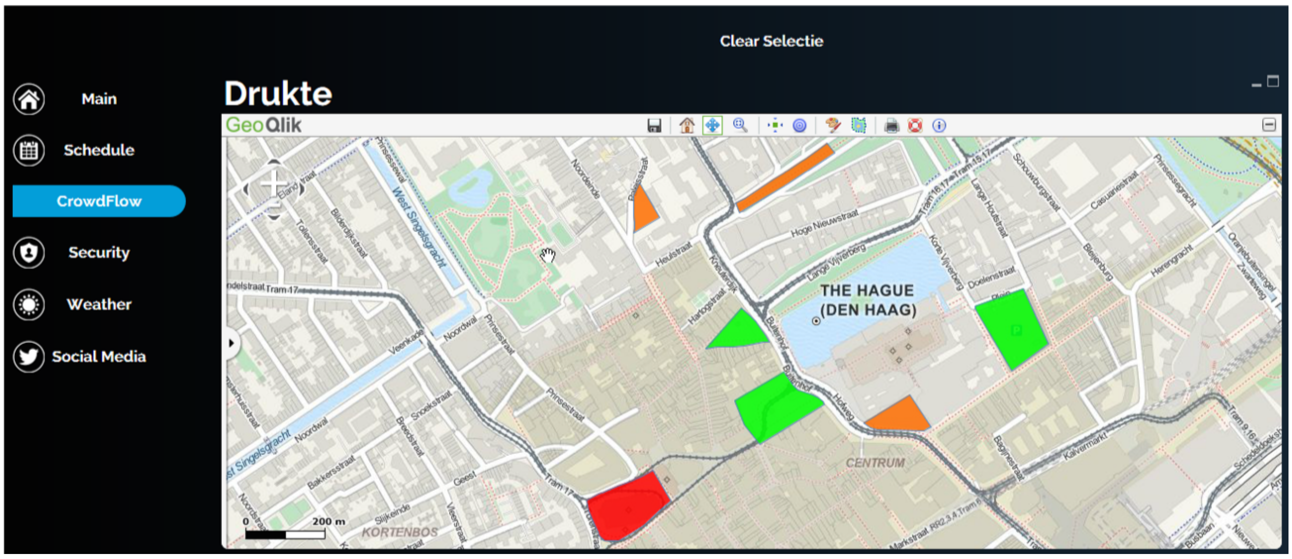
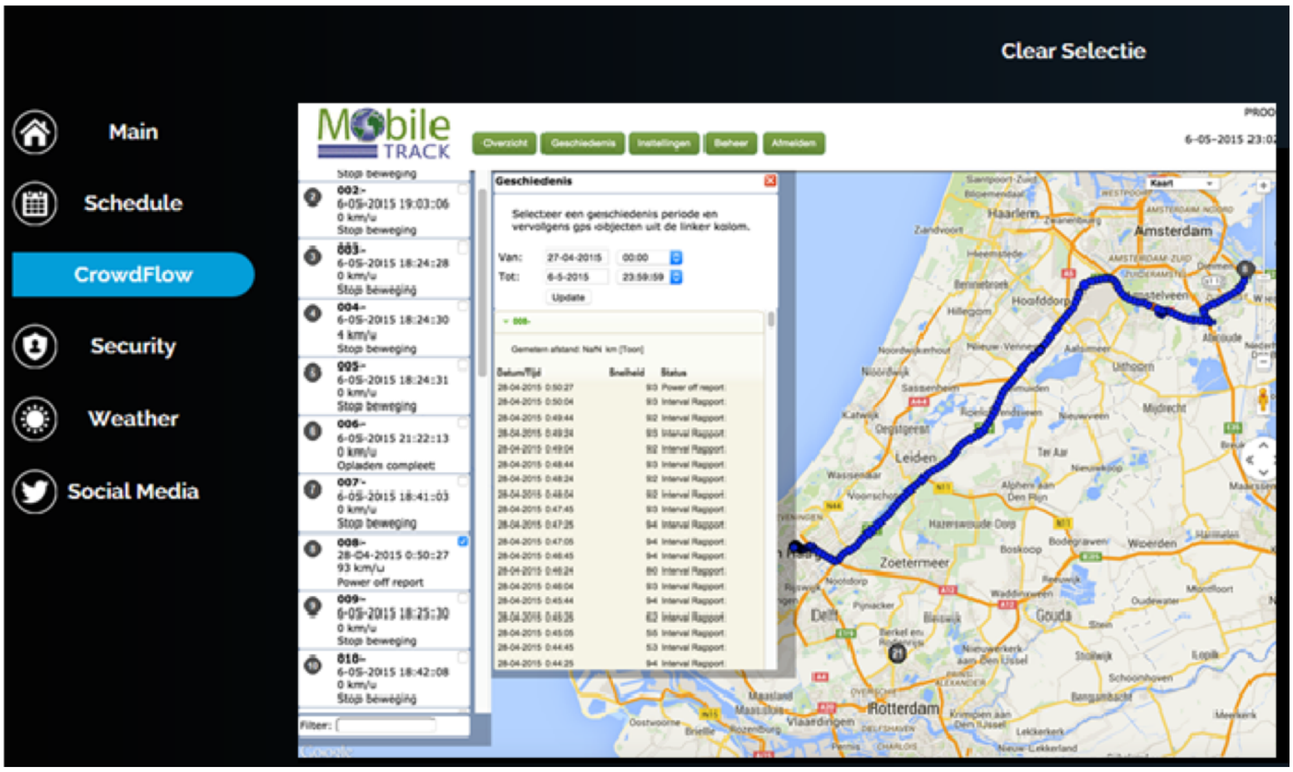


Figure 2: Map of crowd densities reported by security guards

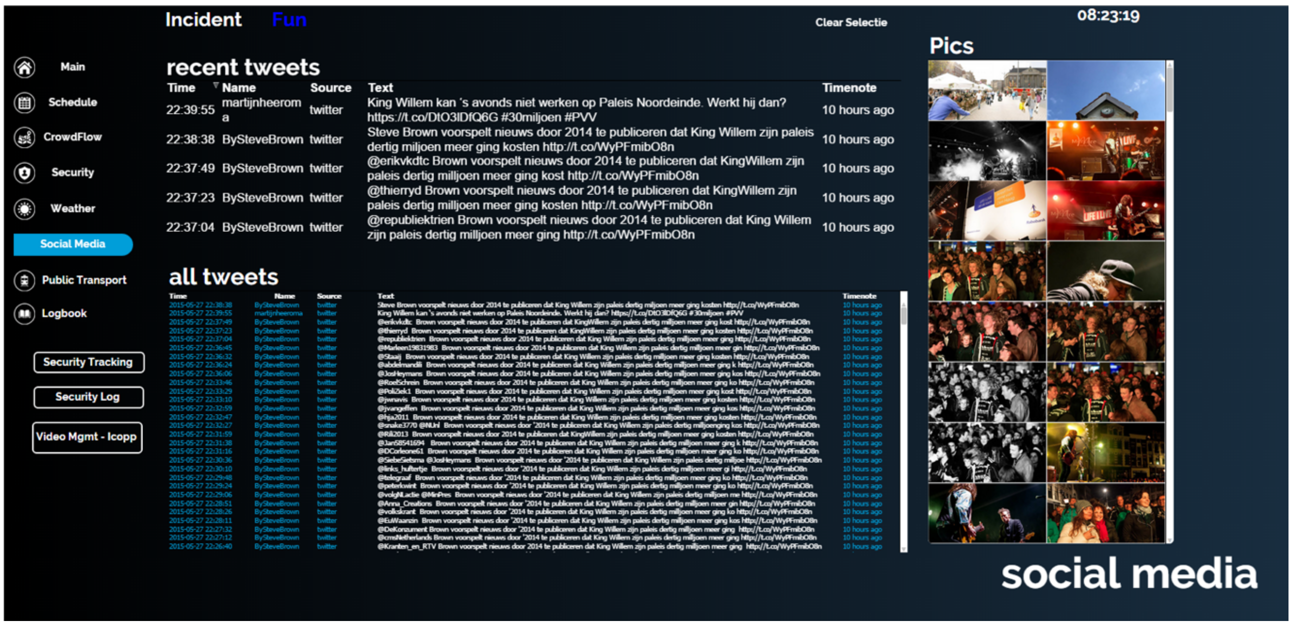


002- 6-05-2015 19:03:06 0 km/u Stop beweging
003- 6-05-2015 18:24:28 0 km/u Stop beweging
004- 6-05-2015 18:24:30 4 km/u Stop beweging
005- 6-05-2015 18:24:31 0 km/u Stop beweging
006- 6-05-2015 21:22:13 0 km/u Opladen compleet
007- 6-05-2015 18:41:03 0 km/u Stop beweging
008- 28-04-2015 0:50:27 93 km/u Power off report
009- 6-05-2015 18:25:30 0 km/u Stop beweging
010- 6-05-2015 18:42:08 0 km/u Stop beweging

Selecteer een geschiedenis periode en vervolgens gps objecten uit de linker kolom.

Van: 27-04-2015 00:00
Tot: 6-5-2015 23:59:59

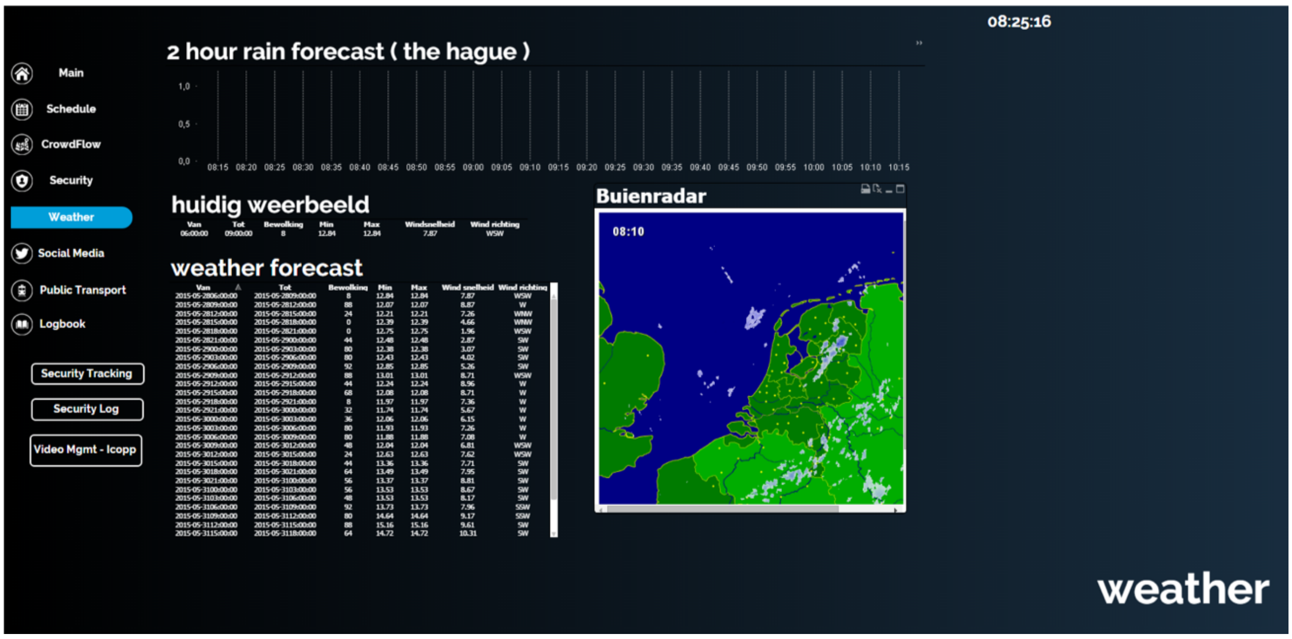
Figure 3: Tracking of visitors with an app



recent tweets

Time	Name	Source	Text	Timenote
22:39:55	martijnheeroma	twitter	King Willem kan 's avonds niet werken op Paleis Noordeinde. Werkt hij dan? https://t.co/DtO3IDfQ6G #30miljoen #PVV	10 hours ago
22:38:38	BySteveBrown	twitter	Steve Brown voorspelt nieuws door 2014 te publiceren dat King Willem zijn paleis dertig miljoen meer ging kosten http://t.co/WyPFmibO8n	10 hours ago
22:37:49	BySteveBrown	twitter	@erikvkdtc Brown voorspelt nieuws door 2014 te publiceren dat KingWillem zijn paleis dertig miljoen meer ging kost http://t.co/WyPFmibO8n	10 hours ago
22:37:23	BySteveBrown	twitter	@thierryd Brown voorspelt nieuws door 2014 te publiceren dat KingWillem zijn paleis dertig miljoen meer ging kosten http://t.co/WyPFmibO8n	10 hours ago
22:37:04	BySteveBrown	twitter	@republiektrien Brown voorspelt nieuws door 2014 te publiceren dat King Willem zijn paleis dertig miljoen meer ging http://t.co/WyPFmibO8n	10 hours ago

Figure 4: Social media overview



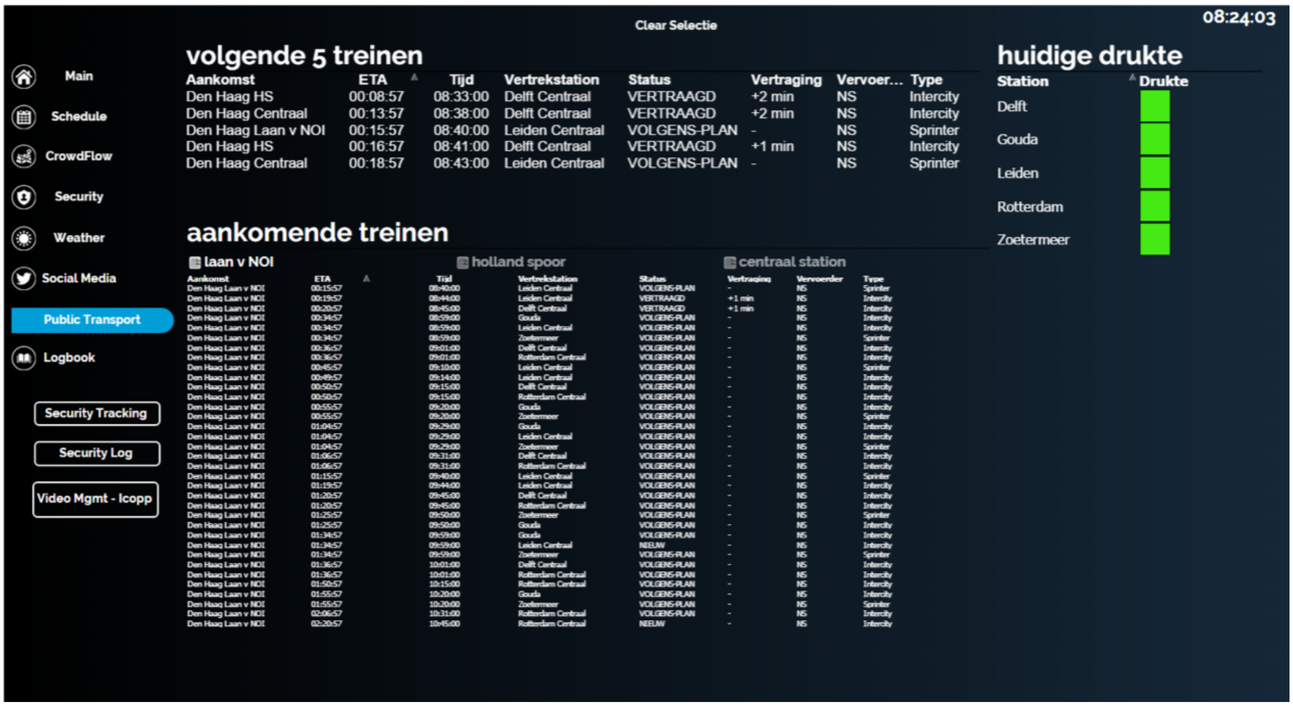
huidig weerbeeld

Van	Tot	Bewolking	Min	Max	Windsnelheid	Wind richting
06:00:00	09:00:00	8	12.84	12.84	7.87	WSW

weather forecast

Van	Tot	Bewolking	Min	Max	Wind snelheid	Wind richting
2015-05-2806:00:00	2015-05-2809:00:00	8	12.84	12.84	7.87	WSW
2015-05-2809:00:00	2015-05-2812:00:00	88	12.07	12.07	8.87	W
2015-05-2812:00:00	2015-05-2815:00:00	24	12.21	12.21	7.26	WNW
2015-05-2815:00:00	2015-05-2818:00:00	0	12.39	12.39	4.66	WNW
2015-05-2818:00:00	2015-05-2821:00:00	0	12.75	12.75	1.96	WSW
2015-05-2821:00:00	2015-05-2900:00:00	44	12.48	12.48	2.87	SW
2015-05-2900:00:00	2015-05-2903:00:00	80	12.38	12.38	3.07	SW
2015-05-2903:00:00	2015-05-2906:00:00	80	12.43	12.43	4.02	SW
2015-05-2906:00:00	2015-05-2909:00:00	92	12.85	12.85	5.26	SW
2015-05-2909:00:00	2015-05-2912:00:00	88	13.01	13.01	8.71	WSW
2015-05-2912:00:00	2015-05-2915:00:00	44	12.24	12.24	8.96	W
2015-05-2915:00:00	2015-05-2918:00:00	68	12.08	12.08	8.71	W
2015-05-2918:00:00	2015-05-2921:00:00	8	11.97	11.97	7.36	W
2015-05-2921:00:00	2015-05-3000:00:00	32	11.74	11.74	5.67	W
2015-05-3000:00:00	2015-05-3003:00:00	36	12.06	12.06	6.15	W
2015-05-3003:00:00	2015-05-3006:00:00	80	11.93	11.93	7.26	W
2015-05-3006:00:00	2015-05-3009:00:00	80	11.88	11.88	7.08	W
2015-05-3009:00:00	2015-05-3012:00:00	48	12.04	12.04	6.81	WSW
2015-05-3012:00:00	2015-05-3015:00:00	24	12.63	12.63	7.62	WSW
2015-05-3015:00:00	2015-05-3018:00:00	44	13.36	13.36	7.71	SW
2015-05-3018:00:00	2015-05-3021:00:00	64	13.49	13.49	7.95	SW
2015-05-3021:00:00	2015-05-3100:00:00	56	13.37	13.37	8.81	SW
2015-05-3100:00:00	2015-05-3103:00:00	56	13.53	13.53	8.67	SW
2015-05-3103:00:00	2015-05-3106:00:00	48	13.53	13.53	8.17	SW
2015-05-3106:00:00	2015-05-3109:00:00	92	13.73	13.73	7.96	SSW
2015-05-3109:00:00	2015-05-3112:00:00	80	14.64	14.64	9.17	SSW
2015-05-3112:00:00	2015-05-3115:00:00	88	15.16	15.16	9.61	SW
2015-05-3115:00:00	2015-05-3118:00:00	64	14.72	14.72	10.31	SW

Figure 5: Weather forecast



volgende 5 treinen

Aankomst	ETA	Tijd	Vertrekstation	Status	Vertraging	Vervoer...	Type
Den Haag HS	00:08:57	08:33:00	Delft Centraal	VERTRAAGD	+2 min	NS	Intercity
Den Haag Centraal	00:13:57	08:38:00	Delft Centraal	VERTRAAGD	+2 min	NS	Intercity
Den Haag Laan v NOI	00:15:57	08:40:00	Leiden Centraal	VOLGENS-PLAN	-	NS	Sprinter
Den Haag HS	00:16:57	08:41:00	Delft Centraal	VERTRAAGD	+1 min	NS	Intercity
Den Haag Centraal	00:18:57	08:43:00	Leiden Centraal	VOLGENS-PLAN	-	NS	Sprinter

huidige drukte

Station	Drukte
Delft	🟩
Gouda	🟩
Leiden	🟩
Rotterdam	🟩
Zoetermeer	🟩

aankomende treinen

laan v NOI | holland spoor | centraal station

Aankomst	ETA	Tijd	Vertrekstation	Status	Vertraging	Vervoerder	Type
Den Haag Laan v NOI	00:15:57	08:40:00	Leiden Centraal	VOLGENS-PLAN	-	NS	Sprinter
Den Haag Laan v NOI	00:19:57	08:44:00	Leiden Centraal	VERTRAAGD	+1 min	NS	Intercity
Den Haag Laan v NOI	00:20:57	08:45:00	Delft Centraal	VERTRAAGD	+1 min	NS	Intercity
Den Haag Laan v NOI	00:34:57	08:59:00	Gouda	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	00:34:57	08:59:00	Leiden Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	00:34:57	08:59:00	Zoetermeer	VOLGENS-PLAN	-	NS	Sprinter
Den Haag Laan v NOI	00:36:57	09:01:00	Delft Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	00:36:57	09:01:00	Rotterdam Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	00:45:57	09:10:00	Leiden Centraal	VOLGENS-PLAN	-	NS	Sprinter
Den Haag Laan v NOI	00:49:57	09:14:00	Leiden Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	00:50:57	09:15:00	Delft Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	00:50:57	09:15:00	Rotterdam Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	00:55:57	09:20:00	Gouda	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	00:55:57	09:20:00	Zoetermeer	VOLGENS-PLAN	-	NS	Sprinter
Den Haag Laan v NOI	01:04:57	09:29:00	Gouda	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	01:04:57	09:29:00	Leiden Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	01:04:57	09:29:00	Zoetermeer	VOLGENS-PLAN	-	NS	Sprinter
Den Haag Laan v NOI	01:06:57	09:31:00	Delft Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	01:06:57	09:31:00	Rotterdam Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	01:15:57	09:40:00	Leiden Centraal	VOLGENS-PLAN	-	NS	Sprinter
Den Haag Laan v NOI	01:19:57	09:44:00	Leiden Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	01:20:57	09:45:00	Delft Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	01:20:57	09:45:00	Rotterdam Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	01:25:57	09:50:00	Zoetermeer	VOLGENS-PLAN	-	NS	Sprinter
Den Haag Laan v NOI	01:25:57	09:50:00	Gouda	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	01:34:57	09:59:00	Gouda	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	01:34:57	09:59:00	Leiden Centraal	NIEUW	-	NS	Intercity
Den Haag Laan v NOI	01:34:57	09:59:00	Zoetermeer	VOLGENS-PLAN	-	NS	Sprinter
Den Haag Laan v NOI	01:36:57	10:01:00	Delft Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	01:36:57	10:01:00	Rotterdam Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	01:50:57	10:15:00	Rotterdam Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	01:55:57	10:20:00	Gouda	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	01:55:57	10:20:00	Zoetermeer	VOLGENS-PLAN	-	NS	Sprinter
Den Haag Laan v NOI	02:06:57	10:31:00	Rotterdam Centraal	VOLGENS-PLAN	-	NS	Intercity
Den Haag Laan v NOI	02:20:57	10:45:00	Rotterdam Centraal	NIEUW	-	NS	Intercity

Figure 6: Crowd density on train stations

Appendix B: Scheme's for different architectures

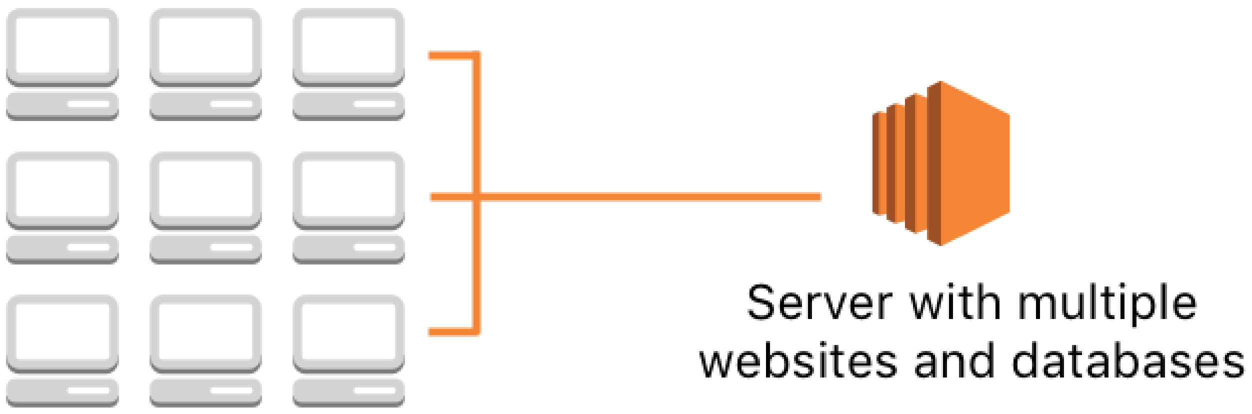


Figure 7: Shared Hosting Diagram

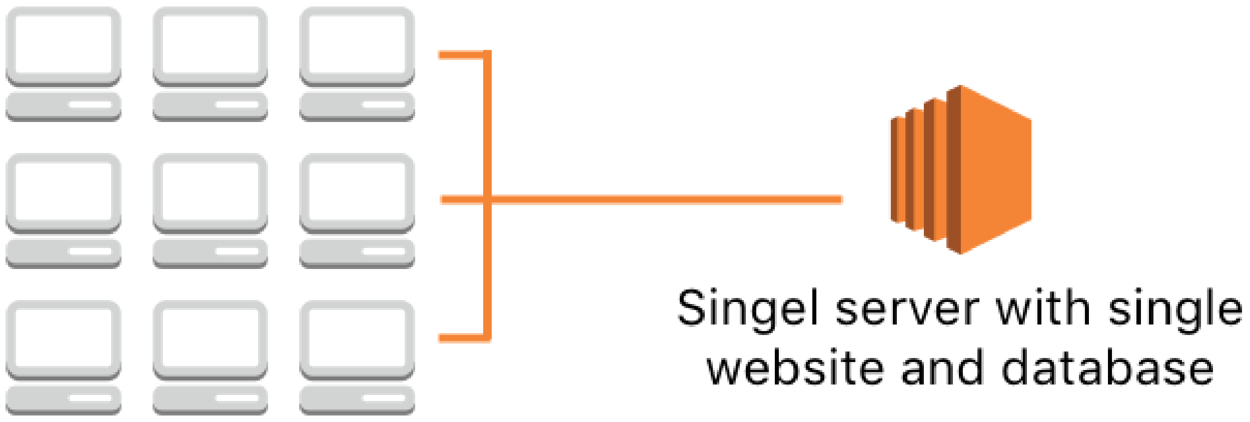


Figure 8: Single Server Diagram

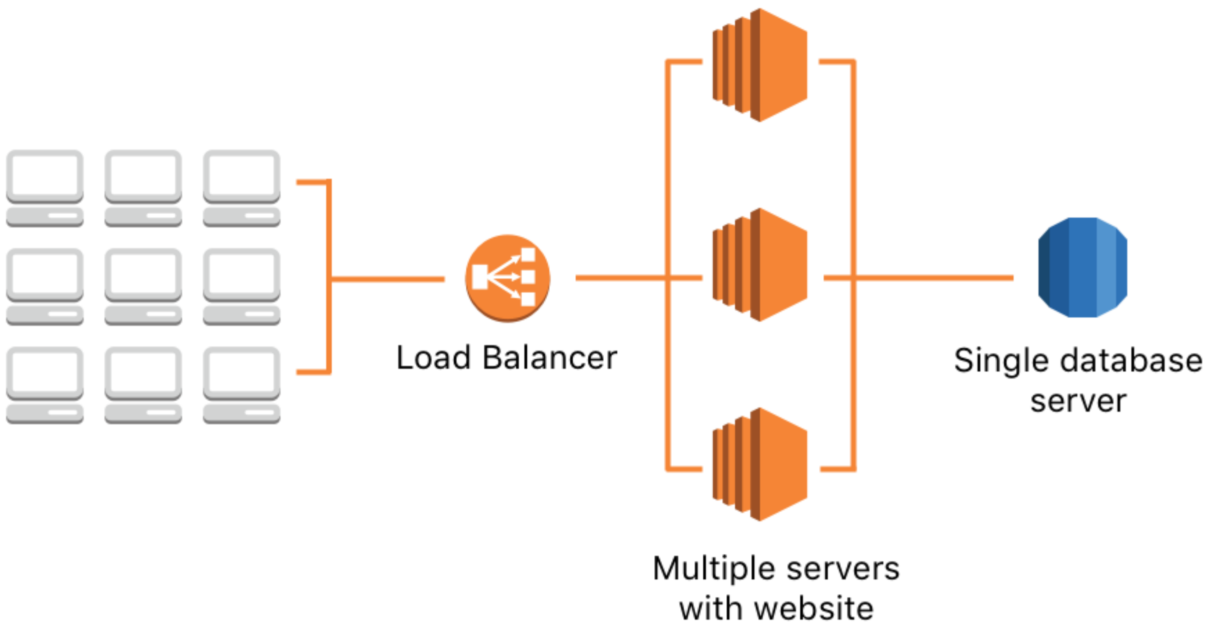


Figure 9: Load Balancing with multiple servers Diagram

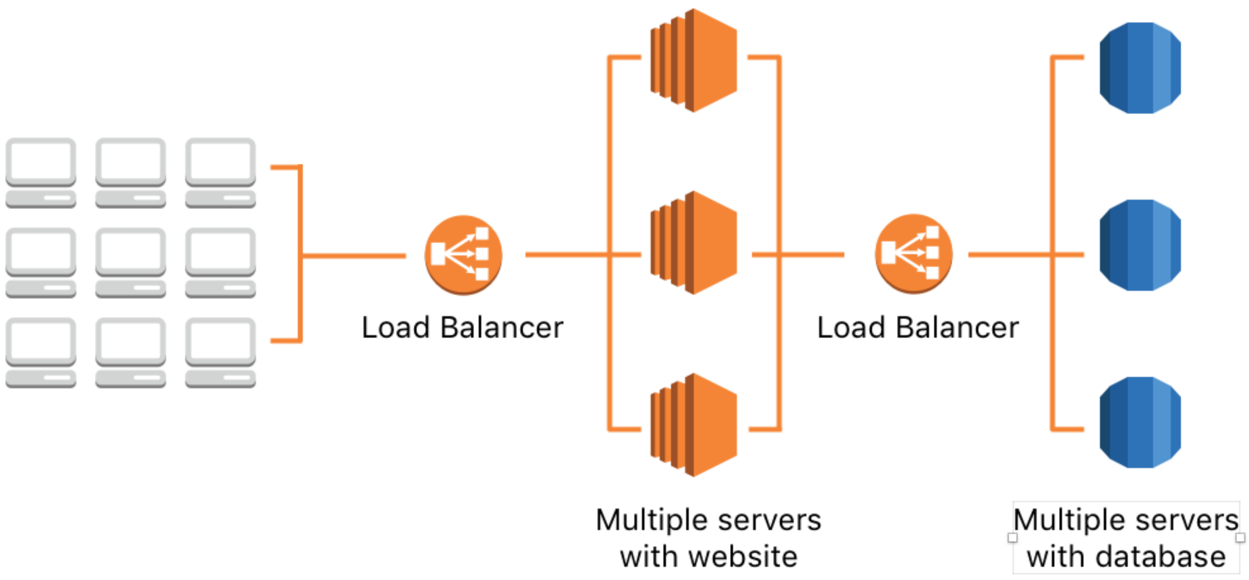


Figure 10: Load Balancing with multiple web- and database-servers Diagram

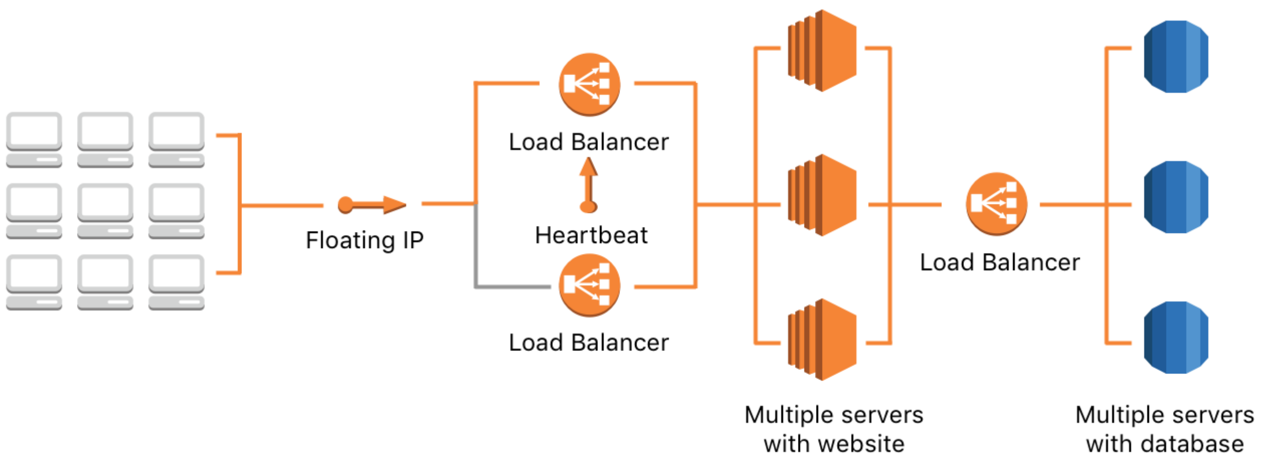


Figure 11: Floating IP with multiple servers Diagram

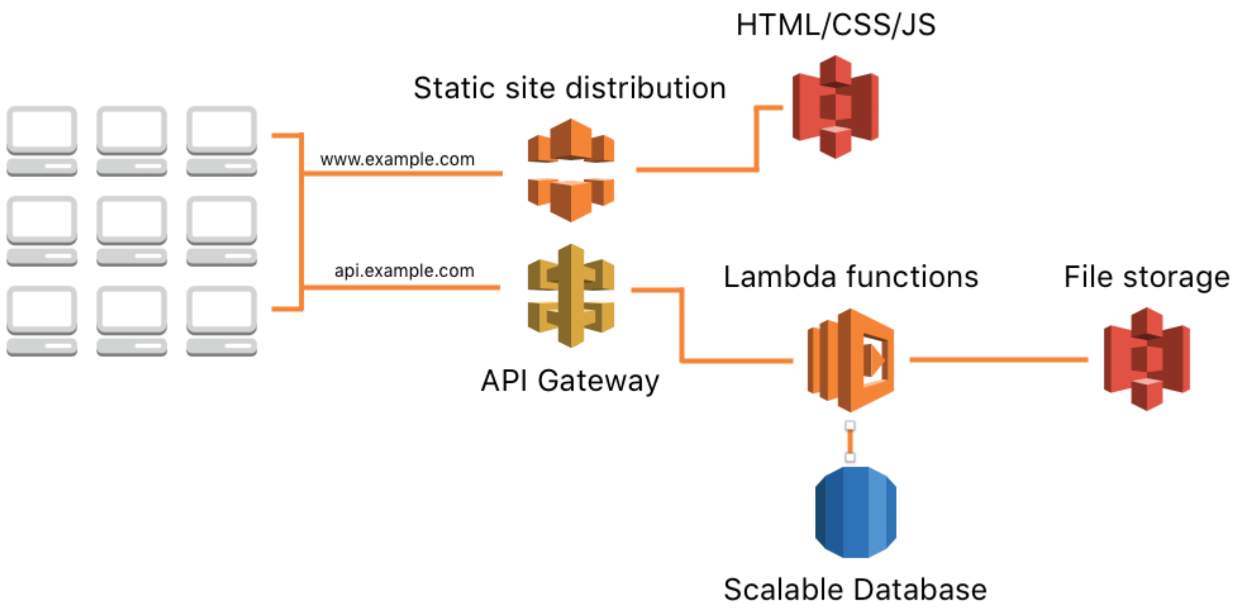


Figure 12: Serverless Diagram with Lambda functions