Juuso Ansaharju

Improving Software Development with Platform-as-a-Service Product – Using Heroku in Web Application Project

Helsinki Metropolia University of Applied Sciences

Master's Degree

Information Technology

Master's Thesis

18 April 2016

Helsinki Metropolia University of Applied Sciences

Author	Juuso Ansaharju
Title	Improving Software Development with Platform-as-a-Service Product – Using Heroku in Web Application Project
Number of Pages	58 pages + 1 appendix
Date	18 April 2016
Degree	Master of Engineering
Degree Programme	Information Technology
Instructors	Ville Jääskeläinen, Metropolia UAS Arto Leinonen, Kielikone Oy

Platform-as-a-Service (PaaS) products promise to offer developers tools to simplify and speed up software development. The case organization, Kielikone Oy, a software development company specialized in digital language services, initiated a process to evaluate and pilot a PaaS product to see whether these promises can be fulfilled.

The process to find a suitable PaaS product to use, and the web development project to pilot it with was the focus of the study as well the assessment of lessons learned after the pilot. The assessment was done by issuing a questionnaire to relevant actors involved in the development of the pilot project and analyzing its results.

The evaluation of PaaS products in the market resulted in choosing Heroku PaaS as the provider with which to conduct further investigations. The use of Heroku during the pilot project to create a new web application for vocabulary learning was successful and generated positive feedback by the developers. Critique was also reported, but it can be stated that benefits of using PaaS outweigh the drawbacks.

A set of improvement suggestions for the continued use of the chosen PaaS product, Heroku, in the case organization is the main outcome of the study. The majority of improvement suggestions gathered hoped for further utilization of Heroku's features including deepening the integration with Heroku and Github, simplification of add-on management and finding ways to deal with security concerns introduced by Heroku's open doors policy.

Keywords	Cloud computing, PaaS, DevOps, Heroku, web development

Tekijä Työn nimi Sivumäärä Päivämäärä	Juuso Ansaharju Ohjelmistokehityksen parantaminen PaaS-palvelun avulla – Heroku-palvelun käyttö web-kehitysprojektissa 58 sivua + 1 liite 18. huhtikuuta 2016
Tutkinto	Master of Engineering
Koulutusohjelma	Tietotekniikka
Ohjaajat	Ville Jääskeläinen, Metropolia AMK Arto Leinonen, Kielikone Oy

PaaS-palvelut tarjoavat kehittäjille lupauksia siitä, että ne nopeuttavat ja yksinkertaistavat ohjelmistokehitystä. Kielikone Oy, ohjelmistotalo, joka orikoistunut digitaalisiin kielipalveluihin, ja jolle työ tehtiin, oli kiinnostunut näkemään toteutuisivatko lupakset yrityksen ohjelmistokehityksessä.

Työ koostuu PaaS-tuotteen valintaan johtaneen evaluaatioprosessin ja tuotteen pilotointiprojektin läpiviennin raportoimisesta sekä kyselytutkimuksen avulla saatujen tietojen analysoinnista. Kyselytutkimuksessa kysyttiin yrityksen ohjelmistokehittäjien kokemuksia PaaS-tuotteen käytöstä.

Evaluaatioprossin tuloksena pilotoitavaksi PaaS tuotteeksi valikoitui Heroku, jota käytettiin pilottiprojektissa uuden web-pohjaisen sanastonoppimispalvelun rakentamiseksi. PaaS-tuotteen käyttö pilottiprojektissa oli menestyksellistä. Palaute oli pääosin posiitivista, vaikkei kritiikiltäkään vältytty.

Työn päätuotos on palauteanalyysin perusteella koottu lista parannusehdotuksista Heroku PaaS -tuotteen jatkokäyttöä silmälläpitäen yrityksessä. Parannusehdotukset liittyvät suurimmilta osin toiveisiin hyödyntää Herokun ominaisuuksia laajemmin syventämällä integraatiota Herokun ja Githubin välillä, yksinkertaistamalla Herokun liittännäisten hallintaa, sekä löytämällä tapoja hallita Herokun käytöstä seuraavia tietoturvaseuraamuksia.

Avainsanat	Pilvipalvelut, PaaS, DevOps, Heroku, web-kehitys

Contents

Abbreviations

API	Application Programming Interface
AWS	Amazon Web Services
CLI	Command-Line Interface
DBaaS	DataBase-as-a-Service
DevOps	Development and Operations
EC2	Elastic Compute Cloud
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IaaS	Infrastructure-as-a-Service
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
JWT	JSON Web Token
NIST	National Institute of Standards and Technology
OS	Operating System
PaaS	Platform-as-a-Service
REST	REpresentational State Transfer
RIA	Rich Internet Application
SaaS	Software-as-a-Service
SPA	Single-Page Application
SSL	Secure Sockets Layer
VPC	Virtual Private Cloud
WSGI	Web Server Gateway Interface

1 Introduction

The rise of cloud computing has provided the developer community with service based software application development tools and application hosting possibilities. Among those is the Platform-as-a-Service (PaaS) model that promises to allow developers to focus solely on creating software instead of doing server and infrastructure management.

Iterative agile software development methods encourage setting up rapid feedback loops between developers, operations team and various stakeholders. This can only be achieved if deployment of new working software to production is fast and easy and operations information such as usage statistics and other metrics are easily available. There are established software development methodologies and practices that focus on these topics, namely continuous deployment and test automation under the umbrella term DevOps.

1.1 Context and Goals

This study was conducted at Kielikone Oy, a software development and service company specializing in digital language learning products. At the time of the thesis project, the company was undergoing a technology modernization process affecting most aspects of its software engineering from programming languages and development environments to hosting solutions. The technological changes were catalysed by company's shift towards agile software development.

In the case company, simplifying the deployment processes and the service infrastructure was one of the current goals of the company's technology modernization process. A PaaS service provider evaluation study was started at Kielikone Oy in early 2014. In particular, there was a need to evaluate what benefits adopting the PaaS model for hosting would provide to the company and how development, and operations activities should be organized to support the aforementioned hosting approach.

This study focuses on evaluating the benefits and drawbacks of PaaS product Heroku as part of company’s development process in the context of a particular web develop-

ment project (Section 4.3). The objective of the study was to identify hosting and deployment best practices and gather ideas for further improvements for developers, operations staff and IT personnel that would enable faster deployments and easy application hosting. Thus, the outcome of the study is an analysis of developer staff's experiences with Heroku and a set of improvement suggestions to be used in developing the company's software development processes.

1.2 Research Background

The objective of the thesis project was to utilize PaaS provider Heroku in a software engineering project, analyse the experiences of the effort and generate a set of future improvement ideas for making use of PaaS cloud computing model to improve the company's software development process. The thesis project was performed in the context of a particular web development project where Heroku PaaS service provider was used for application hosting for the first time in large-scale development project in the company.

General level background information for the thesis was gained by gathering theoretical information about cloud computing (PaaS model, in particular) and software development methodologies that can make use of PaaS model (DevOps, in particular). On the company level, background information for the thesis was provided by describing the relevant development environments, processes and practices (current state analysis) in use, and by giving an overview of goals and results of *PaaS provider study* (Section 4.2) that was performed by the researcher. The results of the *PaaS provider study* acted as the basis of the initial PaaS set-up for the web development project (Section 4.3) for creating a new language learning service MOT+.

The researcher's role in the web development project was to lead the investigations and collaborate with developers to specify, organize, and implement the hosting and deployment procedures to use with Heroku. At the end of the web development project's *version 1.0 release*, the researcher gathered further improvement ideas by analysing the lessons learned during the project so far. Data for analysis was acquired by means of a questionnaire to the developers in the company.

The goal this thesis project is, thus, to improve the company's web development process so that it would successfully continue to incorporate the use of a PaaS provider as

one tool in the technology stack. The outcome of the thesis project is documentation that contains improvement suggestions of how to better utilize Heroku PaaS service in the company's future web development projects.

Process

The research process consisted of five consecutive phases:

1. Describing the PaaS provider evaluation study: its motivations, goals and outcomes,
2. Gathering other background information from relevant topics: cloud computing – platform-as-a-service model in particular, and DevOps,
3. Setting up Heroku practices together with developers as part of a web development project,
4. Issuing a questionnaire and analysing its results, and
5. Composing a set of improvement suggestions for future use.

Strategy, methods, and techniques

The research strategy in use was qualitative.

A mixed set of research methods was utilised depending on the research phase. Phases 1 and 2 two utilized analysis and information gathering. Phase 3 was a pilot and phases 4 and 5 utilized questionnaires, analysis and documentation.

Data collection and analysis

Background information was collected from existing research and literature. Secondary data to provide a framework for final analysis was composed from company material processed as the current state analysis section (Section 4).

The primary data was generated from the questionnaire survey results.

Layout of thesis

The thesis is organized into an introduction section (Section 1), two theory sections (Sections 2 and 3), a current state analysis section (Section 4), a questionnaire analysis section (Section 5), results section (Section 6) and summary (Section 7).

2 Platform-as-a-Service

This chapter contains basic information about the PaaS cloud computing model, its characteristics and service and deployment models, and the Heroku PaaS product, as to its platform and solution stack and deployment flow.

2.1 Cloud Computing

Cloud computing refers to both a model for computing and a set of technologies implementing the model. According to Jésus, “cloud computing is a model that provides web-based software, middleware, and computing resources on demand. By deploying technology as a service, users have access only to the resources they need for a particular task, which ultimately enables them to realize savings in investment cost, development and deployment time, and resource overhead" [1] and “cloud computing is about delivering a set of IT capabilities and business functions as services on demand over the Internet or a private network…” [1].

The main characteristics, service models and deployment models of cloud computing are defined by the standardizing body National Institute of Standards and Technology (NIST).

2.1.1 Characteristics of Cloud Service

The characteristic of a cloud computing service as defined by NIST:

- *On-demand self-service* – consumer cloud computing service can provision computing resources automatically without requiring human interaction [2, 2],
- *Broad network access* – service’s capabilities are accessed over network [2, 2],
- *Resource pooling* – service’s computing resources are dynamically assigned to customers without them necessarily knowing the exact sources of the resources [2, 2],
- *Rapid elasticity* – computing resources can be scaled automatically to accommodate changing demands [2, 2],

- *Measured service* – “Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service” [2, 2].

The characteristics listed above are common traits of all cloud computing services regardless of their service or deployment models.

2.1.2 Service Models

The different service models can be categorized into the following classes:

- *Infrastructure-as-a-Service (IaaS)* – IaaS service offers consumer basic computing capabilities such as processing time or data storage on top of which consumer can deploy arbitrary software [2, 3],
- *Platform-as-a-Service (PaaS)* – "The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider" [2, 2],
- *Software-as-a-Service* (SaaS) – SaaS service provides consumers the possibility to use the service provider’s software running on the cloud infrastructure [2, 2].

The characteristics of service models are often combined in hybrid models. See capabilities offered to users in different service models in Figure 1.

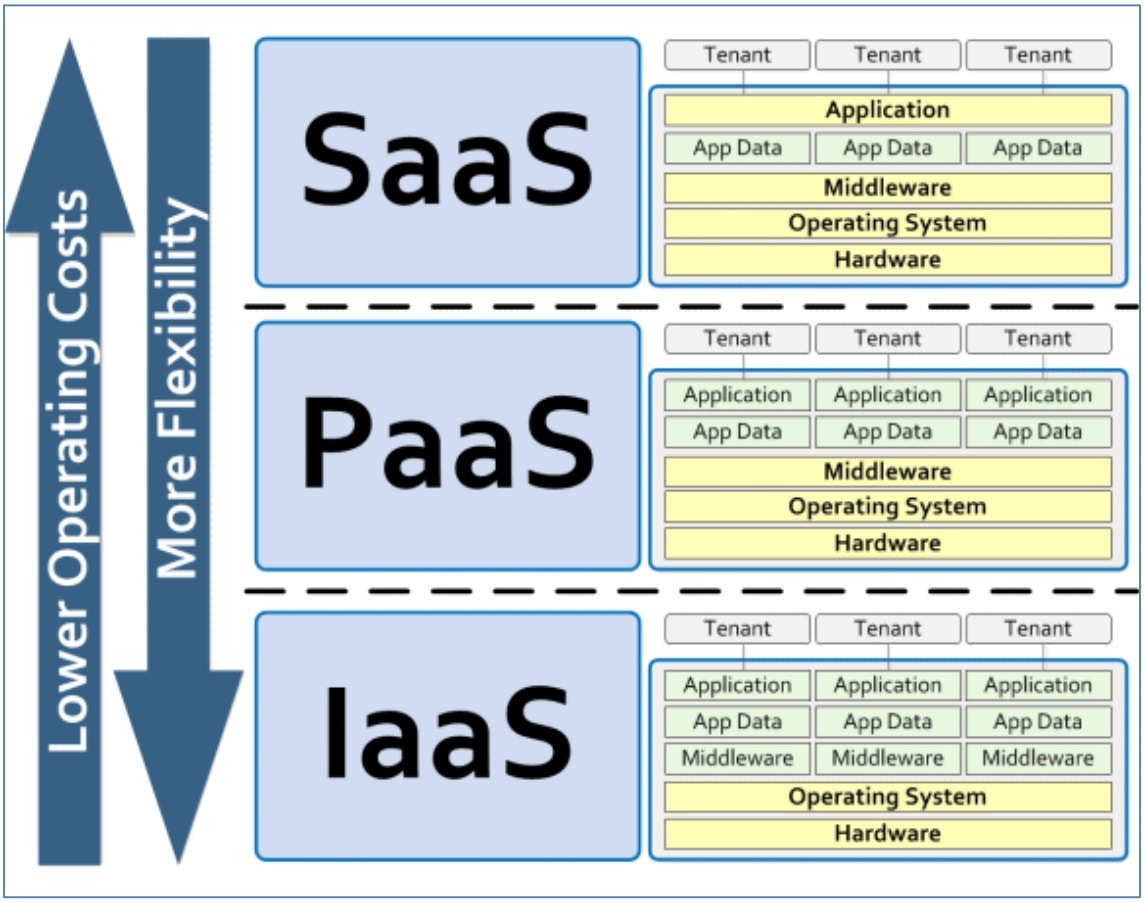

Figure 1. The capabilities offered to consumers in different typical cloud computing service models [1]. Yellow boxes represent the capabilities offered as a service.

The service models create a hierarchy in which models offering lower operating costs depend of models that offer more flexibility. IaaS offers the most flexibility by providing only hardware (and possibly operating system level services) for users build on. SaaS provides users with an end-to-end solution where users only own their application data. PaaS is located between the aforementioned models offering some middleware with which developers can build their own applications.

2.1.3 Deployment Models

Cloud computing services can be classified according to whom the cloud infrastructure is provisioned. There are four deployments models:

- *Private cloud* refers to a situation where the capabilities of the cloud service are provisioned for the use of a single organization [2, 3],

- *Community cloud* refers to a situation where the capabilities of the cloud service are provisioned for the use of multiple organizations having shared concerns [2, 3],
- *Public cloud* refers to a situation where the capabilities of the cloud service are provisioned for the use of the general public [2, 3], and
- *Hybrid cloud* combines the characteristics of any of the aforementioned models [2, 3].

Heroku PaaS product is an example of a hybrid cloud, combining private and public deployment models.

2.2 Platform-as-a-Service Model

PaaS cloud computing services are targeted towards application developers. Orlando states that “the defining factor that makes PaaS unique is that it lets developers build and deploy web applications on a hosted infrastructure” [3]. PaaS can be understood to provide a *platform* and a *solutions stack (see Figure 2)*.

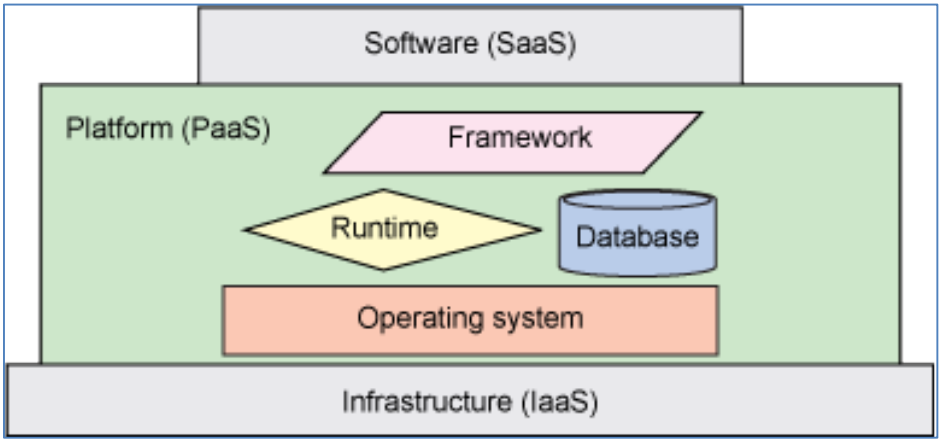

Figure 2: An example of a solution stack consisting of multiple service offered as part of a PaaS platform [3]

Platform is typically an operating system or a software framework capable of executing code in consistent manner [3]. *Solution stack* refers to a set of services developers may utilize to build, deploy and manage applications in the cloud service [3].

2.3 Heroku

“Heroku is a cloud platform that lets companies build, deliver, monitor and scale apps” [4]. Heroku is a PaaS cloud computing service.

Heroku is the name of the product as well as the name of the company behind it, Heroku Incorporated. Heroku is a subsidiary of Salesforce Incorporated and part of their cloud service product catalogue, Salesforce App Cloud [5].

According to Heroku’s own documentation, Heroku is an application development platform that focuses enabling its customers to create and manage web applications without being distracted by hardware or servers [4]. Heroku achieves that by offering a platform to help with application deployments, configuration, management and scaling [4]. Heroku also offers other services such as data persistence tools.

An overview of the features and characteristic of Heroku as described in the technical documentation branded *Heroku Dev Center* is given in the next sections [6]. Focus is put on the aspects relevant to the thesis project. In particular, the following features and properties are omitted from the overview: data persistence services (Heroku Postgres, Heroku Redis), experimental features (Heroku Labs), features for extending the Heroku platform (Heroku Platform API, building custom buildpacks and add-ons), features available only in the Heroku Enterprise licencing model, Heroku Elements marketplace (excluding add-ons), user account management and billing, and other features not used or evaluated during the thesis project.

2.3.1 Terminology

Heroku uses its own set of terms to communicate various abstractions and concepts in the platform and redefine terms that have an altered and a more specific meaning in Heroku compared how they might be understood in other contexts. The terms are listed in alphabetical order in Table 1.

Term	Definition
Add-on	"*Add-on*s are third party, specialized, value-added cloud services that can be easily attached to an *application*, extending its functionality" [7].
Application	"*Application*s consist of your source code, a description of any dependencies, and a *Procfile*" [7].
Building	Building is the process triggered when Heroku receives the *application*. The process utilizes a *buildpack* to produce a *slug* [7].
Buildpack	"Buildpacks lie behind the *slug* compilation process. Buildpacks take your *application*, its dependencies, and the language runtime, and produce *slug*s. [7]"
Config vars	Config vars (i.e. configuration variables) are seen by the application as environment variables and are used to configure the application [7].
Deployment	Deployment is the process of sending an *application* to Heroku [7].
Dyno	"Dynos are isolated, virtualized Unix containers, which provide the environment required to run an *application*" [7].
Dyno (one-off)	"One-off dynos are temporary *dyno*s that run with their input/output attached to your local terminal. They're loaded with your latest *release*" [7].
Dyno formation	"Application's dyno formation is the total number of currently-executing *dyno*s, divided between the various process types you have *scaled*" [7].
Dyno manager	"The dyno manager is responsible for managing *dyno*s across all *application*s running on Heroku" [7].
Dyno runtime	Dyno runtime is a component of Heroku platform that provisions *dynos*, manages dyno lifecycle, adds or removes dynos according to scaling actions, provides network configurations for dynos, routes web traffic to dynos and captures log output of dynos [7].
Ephemeral filesystem	"Each *dyno* gets its own ephemeral filesystem - with a fresh copy of the most recent *release*. It can be used as temporary scratchpad, but changes to the filesystem are not reflected to other *dynos*" [7].
HTTP routing	"Heroku's HTTP routers distribute incoming requests for application across running web *dynos*" [7].
Logplex	Logplex is Heroku's log delivery system that combines logs from various sources to be viewed in the context of a single *application* [7].
Process type	"Each line [in a *Procfile*] declares a process type - a named command that can be executed against your built applica-

	tion" [7]. Process type 'web' is executed automatically by Heroku after a *slug* has been generated.
Procfile	Procfile is a text file generated by the user in root directory of the Git repository containing the source for the application that list commands to be executed by Heroku after deployment [7].
Region	The geographical location where an *application* can be deployed. Available regions depend on the type of *Dyno Runtime* [7].
Release	A *slug* combined with a set of *config vars* is called a release [7].
Rollback	Rollback is an action user can perform to redeploy a previous *release* [7]
Scaling	Scaling is the process of changing the number and type of *dyno*s allocated for a *process type* [7].
Sleeping	"*Application*s that use the free *dyno* type will sleep. When a sleeping application receives HTTP traffic, it will be awakened - causing a delay of a few seconds" [7].
Slug	"A slug is a bundle of your source [code], fetched dependencies, the language runtime, and compiled/generated output of the build system - ready for execution. [7]"
Slug compiler	Slug compiler is the part of Heroku's machinery that transforms an application into a *slug*" [7].
Stack	"A stack is an operating system image curated by Heroku" [6].

Table 1: Heroku's terminology

In order to understand how Heroku works, and how it should be used by the developers, it is important to familiarize oneself with the Heroku specific terminology. The terms are used extensively in the later parts of the document.

2.3.2 Platform and Solution Stack

The characteristics of Heroku are described according to Orlando's division to platform and solution stack [3].

Stack and buildpacks

The *platform* applications run on Heroku is called a *stack.* Currently, all new applications deployed to the cloud are using the latest stack called *cedar-14* that is a customized version of Ubuntu 14.04 operating system.

On top of the stack, representing the lowest level of the *solution stack*, Heroku offers build environments that are responsible for turning an application into an entity that is executable on the platform. These build environments are called *buildpack*s. *Buildpack*s come in different flavours from Heroku and can be created by the developers themselves as well. Buildpacks are run by the *slug compiler* to produce executable applications (*slug*s). Developers may create their own buildpacks to make Heroku support additional languages or software frameworks.

Heroku supports running applications written in Ruby, Node.js, Java, Python, Clojure, Scala, Go and PHP programming languages by using Heroku's official *buildpack*s. For each language, a language runtime is provided as part of the solution stack.

Procfile, process types, and dynos

After deployment, an application may start processes of different types on the platform. The processes are defined in a *Procfile* in the application source code. The processes are allocated computing resources in the form of Heroku's virtual computing units, *dyno*s.

Dynos come in various configurations of computing power, ephemeral filesystem storage size and available memory, and can be scaled in types and numbers by process to create a dyno formation for an application (see Figure 3). Dynos are typically attached to an application for its whole lifetime except special *one-off dyno*s that can be used to perform non-persistent operations.

Dyno runtime automatically manages dynos for an application, and *dyno manager* manages the dynos for all the platform.

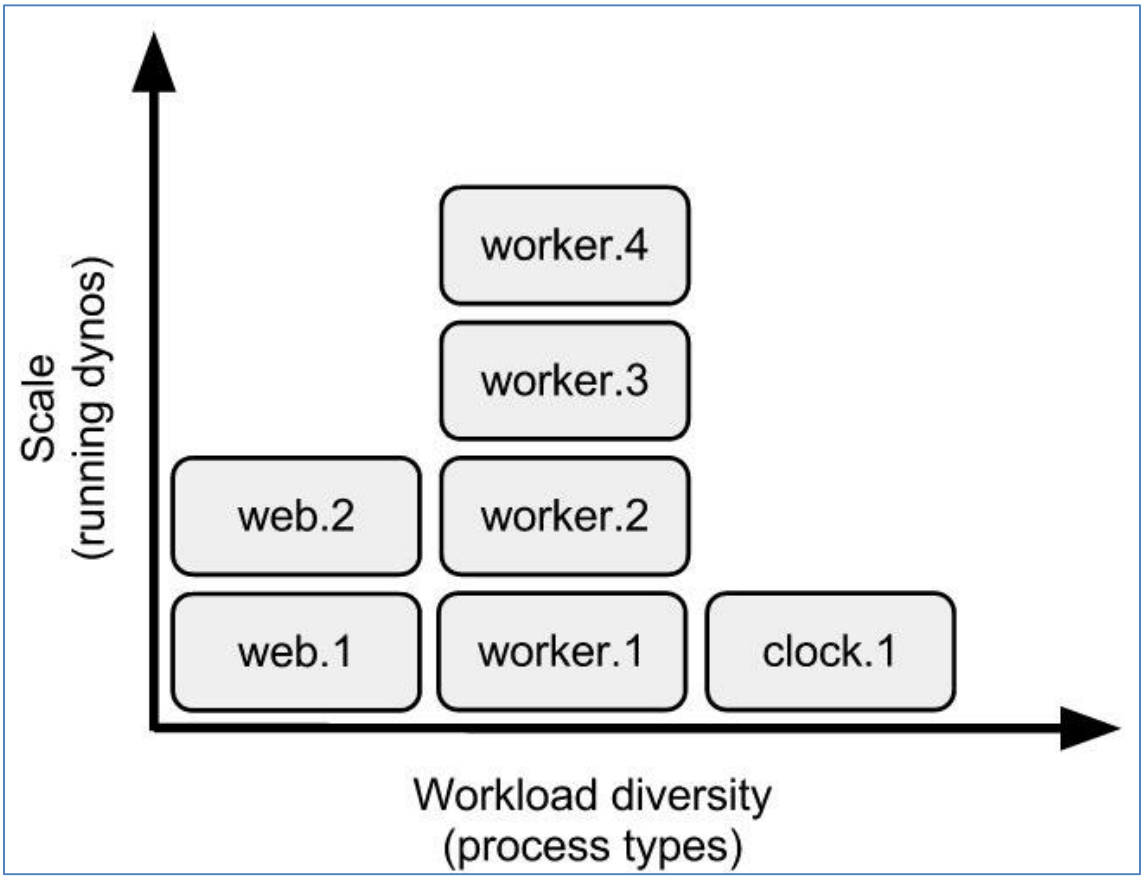

Figure 3. Dyno formation of an imaginary application on Heroku [6]

Routing, logging, regions and add-ons

Heroku's platform also includes components that route network requests to applications (*HTTP router*) and aggregate logs from application's and platform components' log output.

Heroku applications can be configured to run in a datacentre in a specific region of the world. Public applications can currently be deployed to either US or EU regions.

Heroku's solution stack is extended by the add-on ecosystem. With it, 3rd party SaaS or PaaS providers can offer their services to application developers and application developers can easily add the 3rd party functionality to their applications.

2.3.3 Deployment Flow

To have an application running in Heroku, developers need to deploy to Heroku. This is done performing the following steps:

1. Application's source code, dependency declarations and Procfile is created in Git a repository.
2. A Heroku application is created in Heroku under the developer's Heroku user account either with the web interface Heroku Dashboard or with the Heroku CLI tool.
3. An application is automatically assigned a Git repository endpoint by Heroku. It is added as a remote repository to the Git repository containing the application source by the developer.
4. Application is deployed to Heroku by transferring the application source to the remote repository provided by Heroku using Git.
5. Heroku will build the *application* to a *slug,* create a *release,* assign default *dyno formation* to it and execute the process of type *web* declared in the Procfile.

By default, an application is given a *free dyno* that will *sleep* to save processing power.

2.3.4 Runtime Management

After an application is successfully deployed, users can for example:

- scale applications horizontally by changing dyno types and assigning more or less dynos per process,
- assign add-ons to the application,
- set up automatic deployment integrations with Github or Dropbox,
- view release history and perform rollbacks to previous releases,
- manage collaborator access rights,
- perform basic application management (e.g. renaming application, configuring environment variables, changing buildpacks, setting up custom domains).

Applications can be managed via the Dashboard using a web browser or by using the CLI tool.

3 DevOps

This chapter contains basic information about the DevOps (Development and Operations) software development method.

DevOps is a software development method that relies on automation, virtualization and tooling [8, 1]. DevOps aims to automate various processes and free developers to focus on creative application development work instead of mundane and repetitive task [8, 1]. Loukides defines the actors involved in DevOps as "…sophisticated operations experts who work closely with development teams to get continuous deployment right; to build highly distributed systems that are resilient…" [9].

Most concepts that DevOps promotes are not new in the sense that they have existed and been practised well before the term DevOps was coined. DevOps repackages those concepts such as test automation, continuous integration and particular development culture into a single methodology [8, 2].

DevOps can be seen as part of the continuum in the evolution of software development methods form waterfall process through agile development to DevOps. DevOps builds heavily on iterative models incorporated in most agile methodologies [8, 2]. DevOps is a holistic approach that incorporates tools, processes and development culture.

3.1 Components of DevOps

Stordell and Klemetti list the components of DevOps and their goals as follows:

- *Requirement management*: requirement specifications should be visible to all interested parties and they should be kept up-to-date to correlate with the actual progress done and features released [8, 4],
- *Development environments*: development environments should be easy to set-up, be integrated with requirements specifications, source code version control, release system and automatic testing tools [8, 4],
- *Continuous deployment*: ultimately, continuous deployment aims to enable automatic releasing of each change in the product. This can be implemented by

automating each step of the release process so that the changes can be pushed to production environment with confidence [8, 4],

- *Acceptance testing*: requirement specifications are linked in real-time to automated tests that verify the status of the features under development. The tests should by written in a form that is understandable to all stakeholders [8, 5],
- *Virtualization*: running environment should be virtualized to make it easier for applications to stay consistent, easy to scale and test during the lifecycle of the development effort [8, 5],
- *Monitoring*: production environments' performance should be monitored in order to help develop the product [8, 5],
- *APIs*: well documented and standards-compliant APIs should be the focus of development [8, 5].

All components rely on automation in order to be successfully implemented.

3.2 Organizational Culture

Walls states that "…a general consensus has started to form around DevOps being a cultural movement combined with a number of software development practices that enable rapid development" [10, 1]. Walls lists four aspects that are required in an organization culture for a company to succeed in DevOps: *open communication*, *incentive and responsibility alignment*, *respect* and *trust [10, 5]*.

Communication should be product-centric, open and extensive. All development artefacts such as requirements and metrics should be made visible to everyone [*10, 5*].

The product being build should be the main source of dedication in the team and be the basis of incentivization [10, 6].

Walls states that "...everyone needs to recognize the contributions of everyone else, and treat their team members well" [10, 6]. All members of the teams should be free to express their ideas [10, 6]. Wells also states that "trust is a massive component of achieving a DevOps culture" [10, 6]. This applies to trust between team members on personal level and trust between the different functions (e.g. engineers, quality assurance, and management) or sub-teams.

3.3 Platform-as-a-Service Model and DevOps

PaaS caters directly to the needs of DevOps providing automation and management tools to help in achieving the goals of DevOps. Sharma suggests that when evaluating whether a PaaS product is a *DevOps PaaS*, one should find features that offers 'DevOps services', e.g. monitoring-as-a-service, build-as-a-service or test-as-a service, to users [11]. See Figure 4.

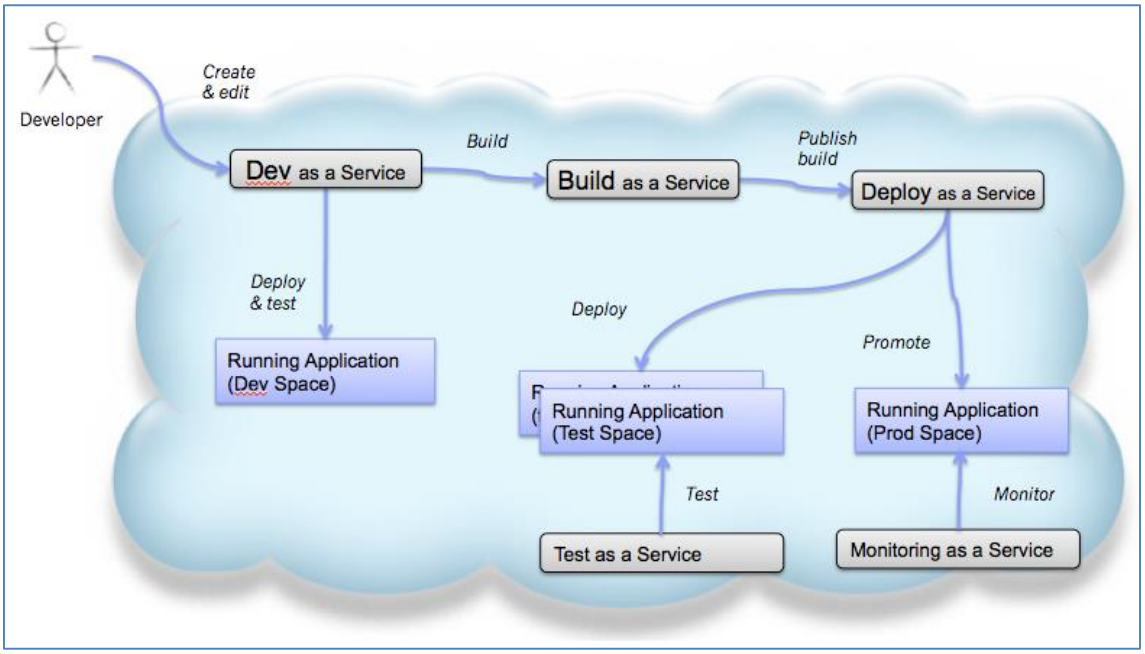

Figure 4. Various components of DevOps presented as services offered by a PaaS [11]

A DevOps PaaS can offer separate running environments (or *spaces* as in Figure 4) for different deployment lifecycle phases of an application. Often in a web development context these separate environments are called: *development*, *testing* and/or *staging* and *production*. Heroku, for example, offers different running environments through their Pipelines feature that allows users to manage application's running environment and *promote* applications from an environment to another without redeploying.

Application performance monitoring and testing services can also be part of the solution stack in PaaS products and well as services to develop, build, and deploy applications. Heroku, as described in Section 2.3, offers these services with the exception of testing-as-a-service features. Heroku also does not offer development-as-a-service

capabilities in the sense that, for example, no cloud-based IDE (Integrated Development Environment) is part of their solution stack.

4 Current State Analysis

This section provides background information about the state of affairs preceding the thesis project, outlines the course of actions during the evaluation study for finding a suitable PaaS provider and elaborates on the web development project where Heroku was piloted.

4.1 Relevant Technology Stacks and Hosting Environments

Kielikone's software development efforts prior to the time of the study were focused on, but not restricted to, the web platform. Multiple different technology stacks were used, but the stack relevant to the web development activities in the context of utilizing PaaS consisted of

- Python and Node.js web applications backed by MongoDB (document database server) in the back-end, and
- single-page applications (SPA) implemented with AngularJS JavaScript framework in the front-end.

The web applications were typically run on servers running Linux, often a version of server variant of Ubuntu OS Linux distribution. In web development projects prior to the PaaS pilot, hosting was arranged in-house using either own hardware or virtualized servers or by utilizing the IaaS by Amazon Web Services (AWS), Elastic Compute Cloud (EC2) and Virtual Private Cloud (VPC) in particular.

More information about how the technology stack evolved during the thesis project is found in Section 4.3.4.

4.2 PaaS Provider Study

This chapter contains basic information about the PaaS provider study conducted to select a PaaS product for further investigations.

A PaaS provider study was started in 2013. Its goal was to find a PaaS product to be piloted in a web development project. It was motivated by scarce server maintenance

resources the company had at the time, experiences from previous projects where Google AppEngine PaaS had been tried out in a small scale, and the need to find new application development and hosting models that would be suitable for prototyping oriented development projects.

The study was implemented in three consecutive phases codenamed: *long list*, *short list* and *pilot*. The researcher conducted the first two phases, while the last and ongoing phase is a joint effort of the researcher and developers at Kielikone Oy.

4.2.1 Phase: Long List

The number of products providing PaaS services at the time was large. More than 95 service providers were identified by means of analysing various online resources. The most valuable resources turned out to be lists of PaaS providers gathered and maintained by non-aligned individuals within the cloud computing community. No record was kept of these resources in the PaaS provider study documentation.

Two evaluation factors were used to narrow down the search base: (1) candidate services should support hosting of Python, JVM and Node.js applications – technologies relevant for the technology stacks in use in existing and planned applications at that time that were planned to utilize PaaS; (2) candidate services should have a generally good level of reliability – either by being operated by a large and international company or by having reference customers that were known to the researcher. Using the aforementioned criteria, a *long list* of candidates was created (see Table 2).

PaaS service provider	Website address
AppFog	https://www.appfog.com/
Appsempler	http://appsembler.com/
Clever Cloud	http://www.clever-cloud.com/
Cloudify	http://www.cloudifysource.org/
dotCloud	https://www.dotcloud.com/
Elasticbox	https://www.elasticbox.com/
Heroku	https://www.heroku.com/
OpenShift	https://www.openshift.com/
Pogoapp	http://www.pogoapp.com/
Windows Azure	http://www.windowsazure.com/

Table 2: The long list

The service providers on the long list were used as the input for the next phase where the list of potential options was narrowed down more.

4.2.2 Phase: Short List

In order to select a suitable service provider for a pilot, a deeper research into the services on the long list was conducted. The criteria for which to base the evaluation was the following:

- Platform support – providers with wide support for technologies (programming languages, application frameworks, runtimes, databases) found in Kielikone's software solutions were favoured,
- Simplicity – providers with easily accessible web management consoles and CLI tools and convenient level of abstraction across the board were favoured,
- Documentation – providers with extensive, detailed and logical documentation were favoured,
- Tools – providers with built-in application monitoring tools or other value adding tools and/or an ecosystem of add-ons or plugins were favoured,
- Reliability – providers that openly published their services' uptime history were favoured,
- Support services – providers with large user community were favoured,

- Scalability – only providers with semi-automatic or fully automatic horizontal and/or vertical scaling as part of their services were considered,
- Pricing – only providers that provided real-life examples of hosting costs were considered,
- Security – only providers that supported custom domain SSL were considered.

Using the aforementioned criteria, a *short list* of candidates was created (Table 3).

PaaS service provider	**Website address**
AppFog	https://www.appfog.com/
dotCloud	https://www.dotcloud.com/
Heroku	https://www.heroku.com/
OpenShift Online	https://www.openshift.com/

Table 3: The short list

Of the service providers on the short list, Openshift Online and dotCloud were dropped due to an additional business requirement add at a later stage:

- Data centre locations – due to network latency and data privacy considerations, only providers offering hosting on European soil were considered

That left two providers, AppFog and Heroku. After setting up simple test applications with service providers, Heroku was chosen as the pilot platform. Decision between Heroku and Appfog was grounded on the fact that developers had some previous experience with Heroku and none with Appfog.

The selection was done in time for the start of the pilot project in mid-2014.

4.3 Web Development Pilot Project

This section contains information about the web development project in which Heroku was piloted.

Heroku's capabilities were tested in action during the development of MOT+ service.

MOT+ is a language learning and dictionary lookup service with which users can learn and teach English words using their native language. With MOT+, users can, among performing other activities, create their own word lists (vocabularies) and practise the words in the lists authored by themselves or others using a selection of word games. The learning experience is gamified by awarding points and various achievements to users based on their activity and letting users follow their own and others' learning progress.

The service's first launch, *MOT+ private beta*, was targeted to Finnish high school level English teachers and students (see Figure 5).

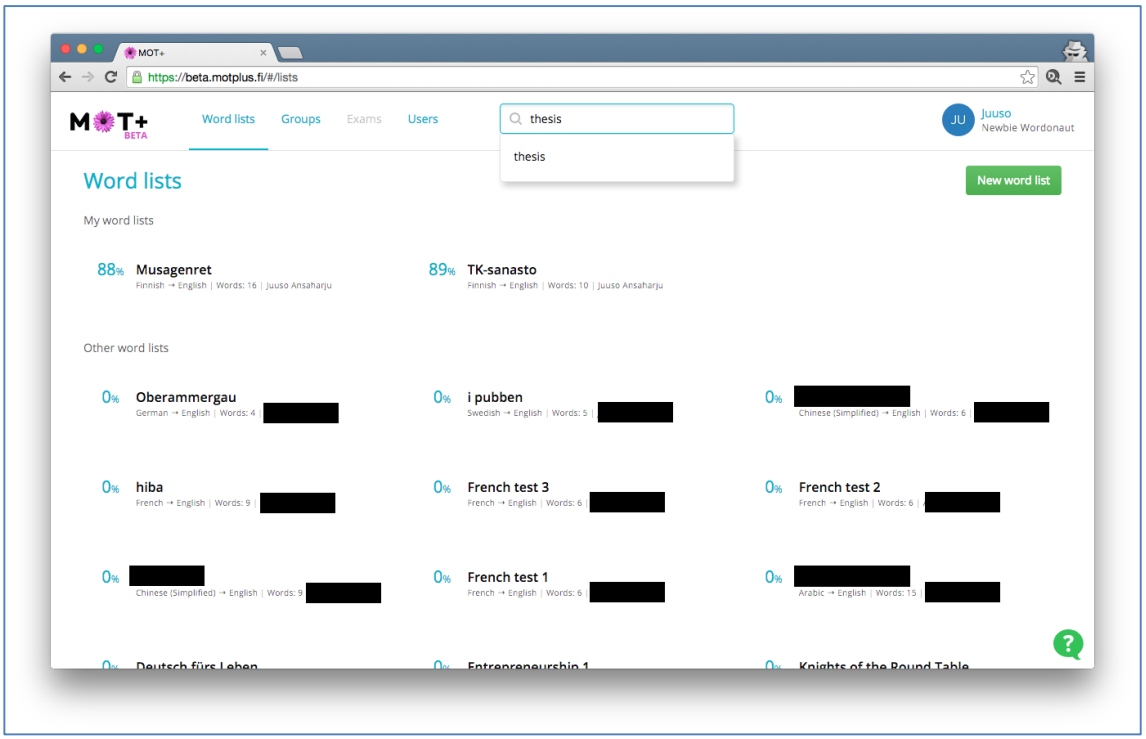

Figure 5: screenshot from MOT+ web client at the time of the private beta release

This study focuses on the state of affairs of the MOT+ project prior to and after the time of the launch of public service to the general service (*version 1.0 release*). Actual software development efforts for the MOT+ service started in Q2 2014. The service was launched for the general public as version 1.0 in late Q4 2015.

At the time when the researcher outlined the development schedule for this document, the project had a development history of 22 months. Consequently, the development project had since its inception had many different development stages and intermediate

milestones, sub-goals, shifts in focus, changes in team composition, changes in project management processes, various 3rd party collaborations, technology changes, and other nuances, most of which are out of the scope of the thesis project.

To provide necessary context, the pilot project's development activities are described in a simplified manner by describing the

- major development phases in chronological order including: an overview of development activities performed, overview of evolution of the system architecture and technology stack from phase to phase, overview of hosting and deployment set-up throughout the phases, and the researcher's role during the different phases,
- team composition and development practices during the project,
- state of the system architecture in Q1 2016,
- state of the technology stack in Q1 2016,
- development environments in place in Q1 2016, and,
- hosting and deployment set-up in Q1 2016.

Focus is put on the state of the affairs after the public 1.0 release (see Figure 6).

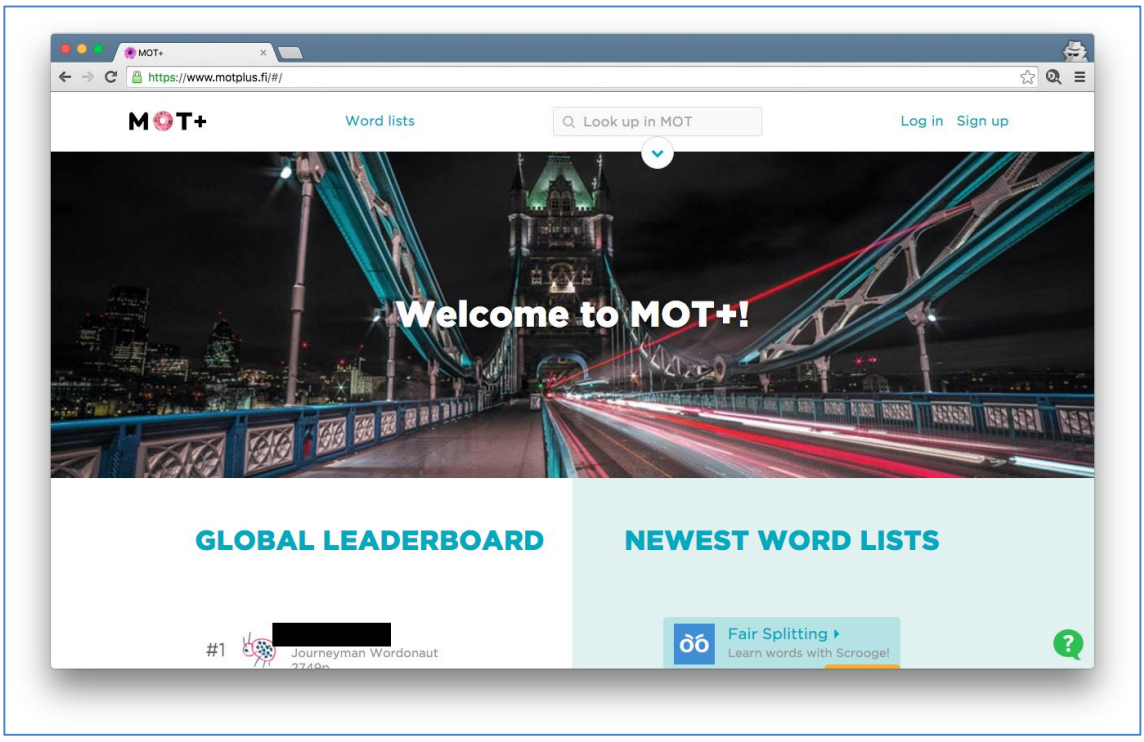

Figure 6: screenshot from MOT+ web client front page in Q1 2016

At the time of writing the thesis report, MOT+ was still undergoing active development.

4.3.1 Development Schedule

The development history of the pilot project can be divided into five roughly distinct consecutive development phases:

1. Prototyping from Q2 2014 to Q4 2014,
2. Preparation for first launch and **private beta release** during Q1 2015,
3. Preparation for second launch and **public beta release** from Q2 2015 to Q3 2015,
4. Preparation for third launch and **release of version 1.0** during Q4 2015,
5. Development efforts after third launch during Q1 2016.

First phase

The *first phase* was focused on verification of the business idea and clarification of the use cases (see Figure 7 for an early vision of use cases for student and teacher roles) the service should fulfil through prototyping.

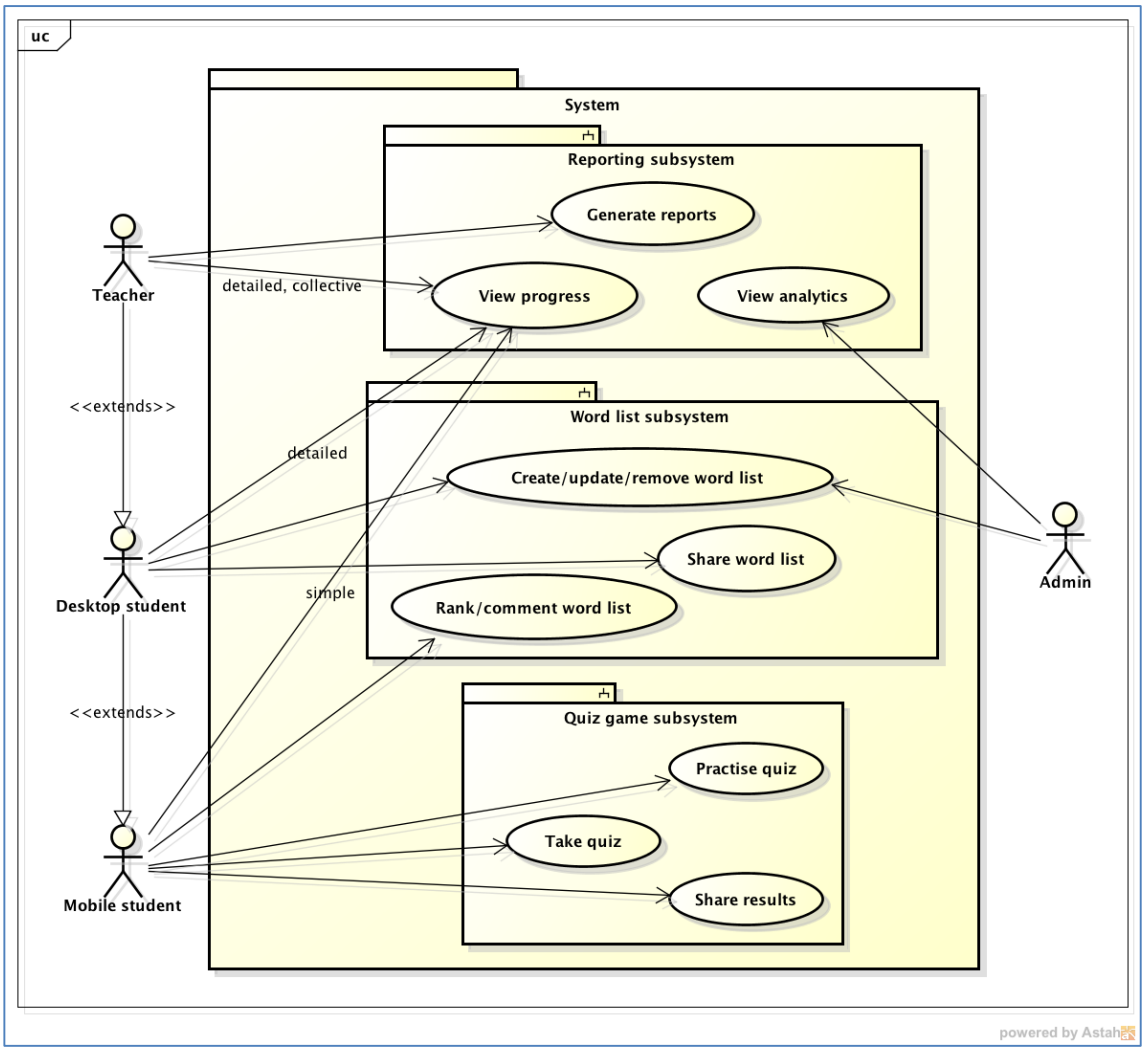

Figure 7: Use case diagram from Q2 2014 depicting an early vision of MOT+'s core feature set. The use cases are divided into multiple subsystems based on the types of use cases they provide to the users of the system.

The development efforts were mainly in the front-end as suggested by UI-first software development method. Development of the core back-ends was started (see Figure 8 for an early vision of system components and their intercommunication for the MOT+ system). The *first phase* was when team set up both front-end and back-end development environments, evaluated various technologies and tools, and got familiarized with Heroku as the deployment and hosting environment. Eve was chosen as the back-end application framework for its easy tools to create RESTful (REpresentational State Transfer) APIs (Application Programming Interface).

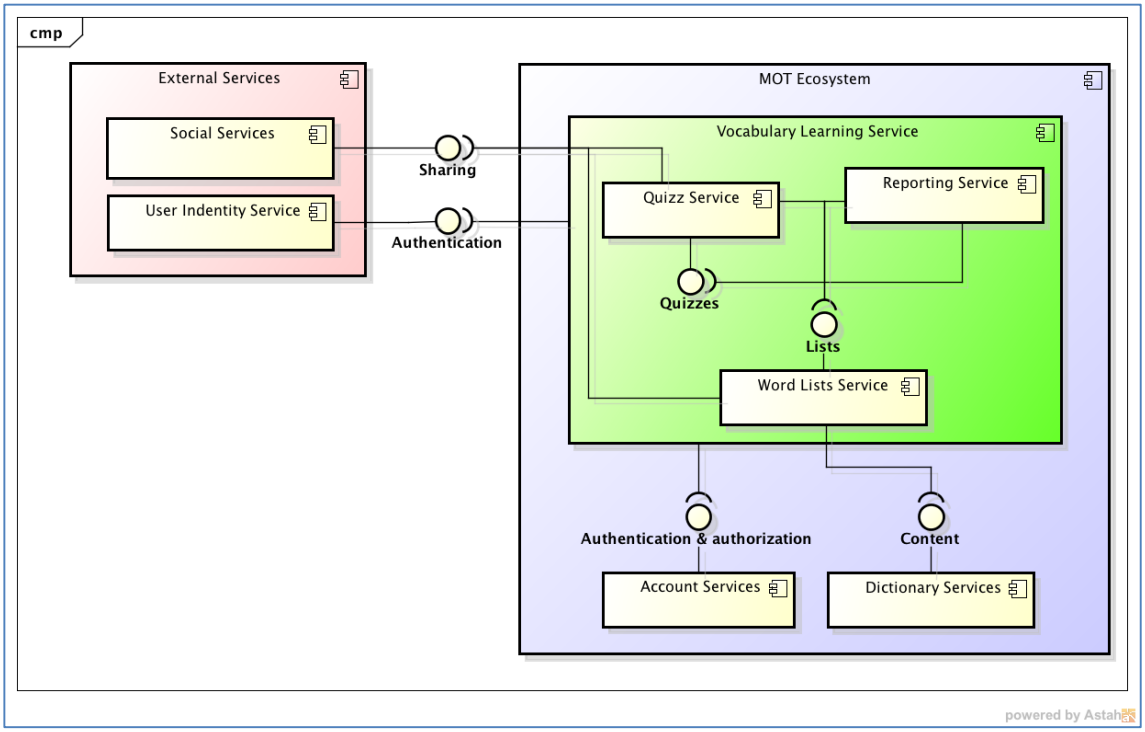

Figure 8: An early component diagram created in the beginning of the prototyping phase. Main system components are visible as well as their envisioned API endpoints and how different components depend on each other.

In this phase, the researcher designed the initial system architecture, got acquainted with Heroku platform and set up the application instances for the required components of the MOT+ system. The researcher collaborated with application developers to share knowledge about how to deploy applications to the PaaS and how to make MOT+ components' codebases run as expected in Heroku's environment.

Second phase

The focus of the *second phase* was to reach a feature-complete state for the first launch and to release the service in *private beta* for selected users. The front-end of the service was taken to a level that was considered acceptable from user experience point-of-view. All required back-end services were developed to a state where integration between all system components was possible. At this phase, development efforts branched between Kielikone's internal development team and various outsourcing partners (see Figure 9).

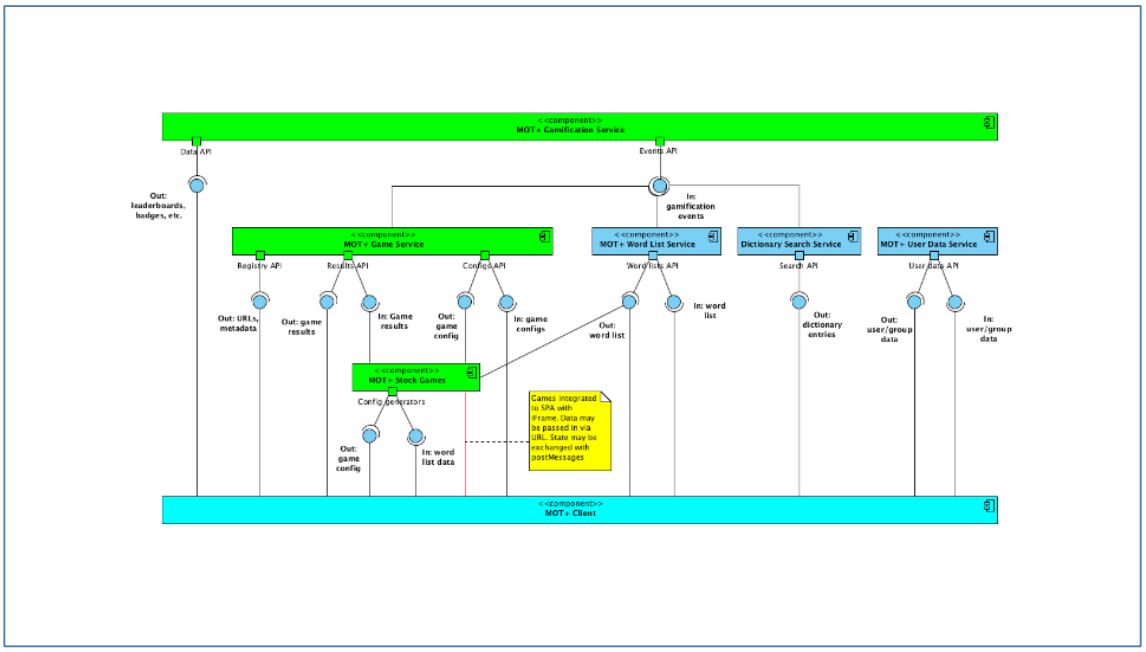

Figure 9: Plan of MOT+ system components from early 2015 before the private beta release. Entities in bright green represent components that were developed by 3rd party collaborators and entities in blue the components that were developed in-house.

Other activities in this phase involved fixing critical bugs in front-end and back-ends and performing quality assurance tasks. Integration work for bringing in all system components was performed by the team. The back-end application framework was changed from Eve to Flask to gain more flexibility in defining RESTful APIs. External user identity service Auth0 was integrated to the service to provide user authentication functionality. See technology stack diagrams Figure 10 and Figure 11 for details about the technologies used.

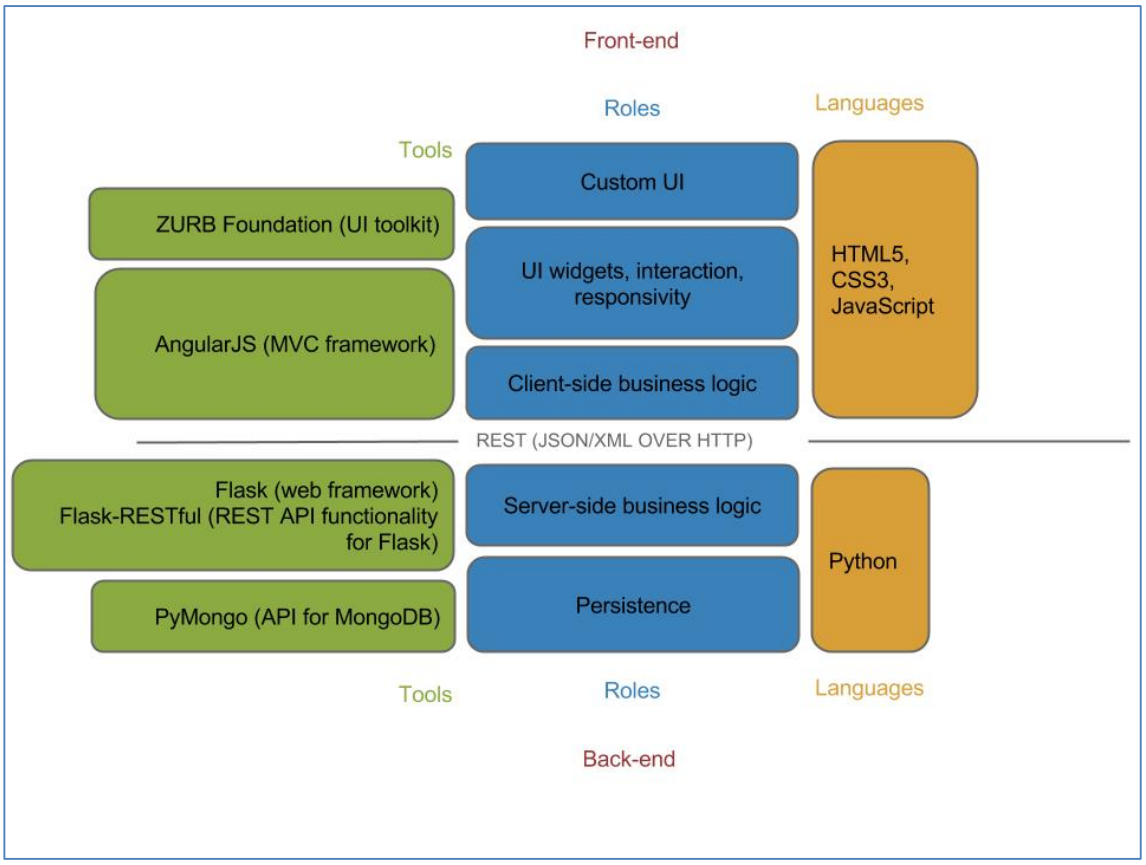

Figure 10: Front-end and back-end technology stacks during the second phase of MOT+ development.

The front-end stack was based on AngularJS JavaScript framework and ZURB Foundation UI toolkit that provided client-side business logic and UI, and the back-end stack was based on Flask Python framework, its extensions and MongoDB database providing data persistence and server-side business logic. The front-end and the back-ends communicated with each over HTTPS using RESTful APIs.

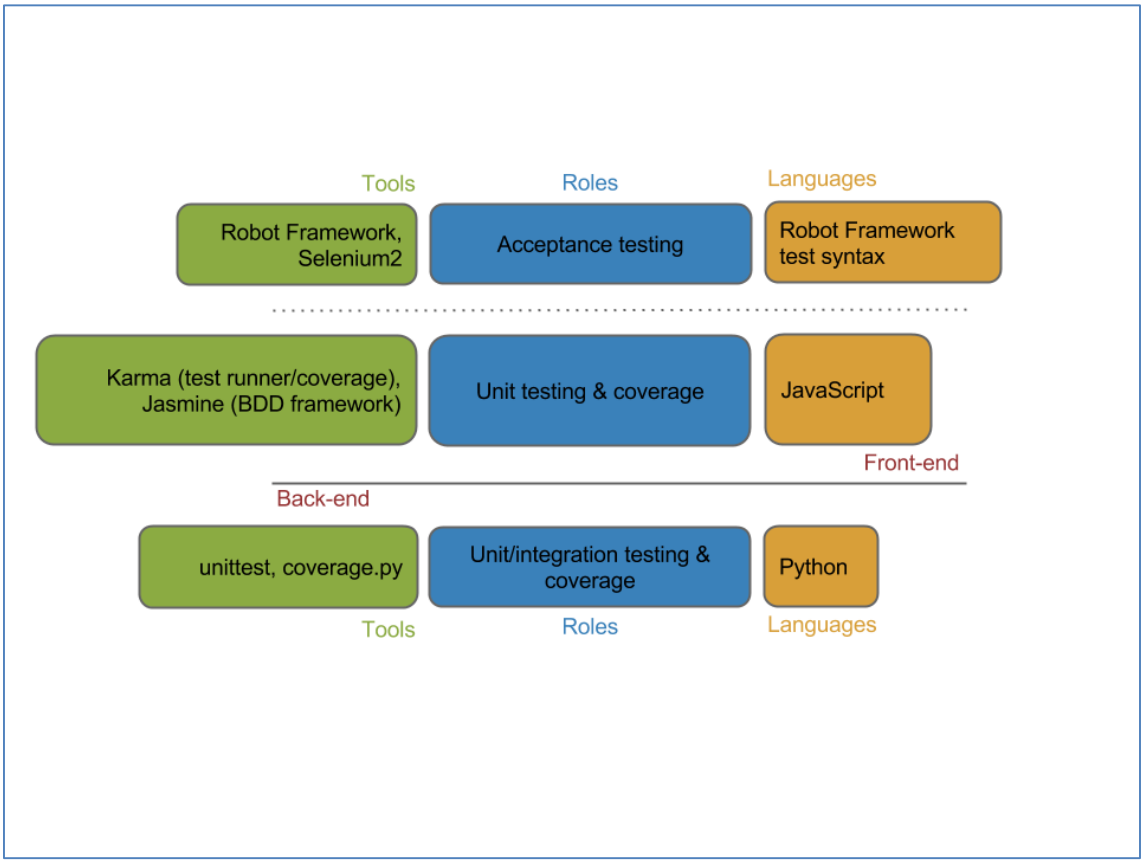

Figure 11: testing technology stack during the second phase of MOT+ development.

The testing stack was based on typical AngularJS ecosystem's testing tools, namely Karma test runner and Jasmine behaviour-driven test framework on the front-end; and using the tools offered by Python standard library and a 3rd party tool *coverage.py* to gather code coverage metrics on the back-end. Additionally, automated UI testing was done using Selenium2 browser automation tool controller by test code written with test automation framework Robot Framework's test syntax.

The researcher collaborated with internal and external developers to set up deployment and hosting environments for all system components in Heroku. Application performance and logging were set up in utilizing Heroku add-ons by the researcher. All application components were hosted in Heroku using minimal horizontal scaling (utilizing the free *dyno* options available at that time) and the system was duplicated to separate staging and production environments.

Third phase

The *third phase* focused on improving the private beta and bringing the service to a level ready for public beta and releasing it. During this period the service saw a lot changes. Due to changes in business needs and the feedback received from users and various stakeholders, a lot of effort was put into improving the service. The improvements manifested in redesigning the client UI, as well as doing extensive bug and user experience defect fixing. Some parts of the application were completely re-implemented while other parts were dropped for good. New 3rd party collaborations were started. The system architecture had shifted from the original plan into a variant that had an added amount of inter-dependency between the back-end services (see Figure 12 in which the orange entities represent the system's back-end components, blue entities represent front-end components and grey entity represents an external dependency).

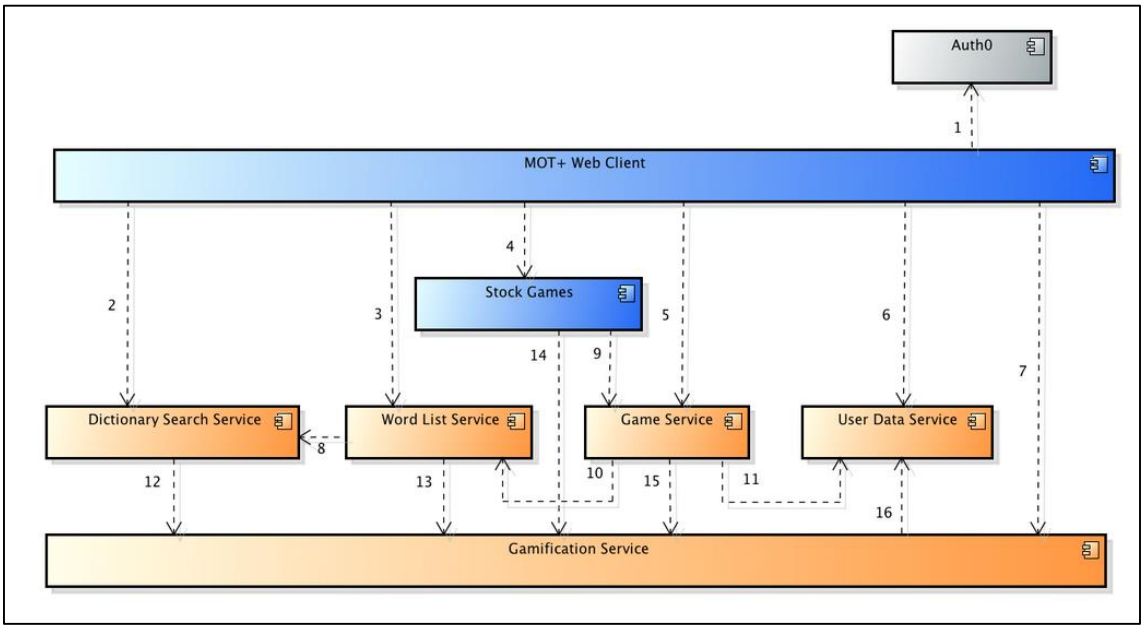

Figure 12: system components and their inter-dependencies after the public beta release.

The researcher helped in bug fixing and prepared the hosting environment for handoff to operations personnel. At the time of the *public beta release*, the production environment was handed over from the researcher to the operations team. MOT+'s Heroku applications were migrated to a new stack (*cedar-14*) and add-on plans were upgraded according to changes in system requirements by the researcher in collaboration with

application developers and the operations personnel. Heroku apps we scaled up due to Heroku starting to limit the possibility to use free dynos in applications running continuously 24 hours a day.

Fourth phase

The *fourth phase* focused on changing the service to a direction to better cater to a larger audience and to support new business requirements. Many features that had existed during the *public beta* period were disabled and the dependency to the external authentication service Auth0 was removed. See Sections 4.3.4 to 4.3.6 for information about the technical details of the MOT+ system after the *version 1.0 release*.

The researcher enabled the Review Apps feature in Heroku for the client component, and, together with the operations personnel, scaled up those MOT+'s production environment's Heroku apps that had suffered from performance problems due to increased user base and shortcomings in system architectural details.

Fifth phase

In the last phase, the development of MOT+ continued. The researcher's responsibilities included leading the team's software development work, and tending the application and system architecture and hosting environment.

4.3.2 Team and Development Process

The team involved in the making of MOT+ did not remain fixed during the development. If numbers are normalized throughout the development history of MOT+, the core team has been composed of:

- 0 to 1 technical manager or architect or lead developer,
- 1 to 2 software engineer(s) with focus on front-end development,
- 0 to 2 software engineer(s) with focus on back-end development,
- 0 to 1 QA person,
- 1 product manager.

The researcher has fulfilled the role listed first with a slightly different focus and set of responsibilities depending on the development phase.

In addition to the core team, five development collaborations with external developers have taken place during different phases of the development history and for different purposes:

- outsourced developers from a partner company 1 working at Kielikone's premises helping with back-end service development during the second and third development phases,
- developers from partner company 2 working with the game and game related backend-development during the second phase,
- developers from partner company 3 working with the game development during the third and fourth phases,
- developers from partner company 4 working with the game development during phase 2,
- designer from partner company 5 working with the UI design during phases two, three and four.

The researcher served as a technical coordinator between the external developers and in-house personnel to assure efficient collaboration and integration of different components into a working entity.

4.3.3 System Architecture and Technologies

The system architecture and technology stack are described here as they were set up in Q1 2016.

MOT+ is a web application. It consists of user-facing front-ends and multiple data-persisting back-end services. At the time of version 1.0 release it had nine inter-dependent system components (see Figure 13 in which the entities in orange represent front-end components and green entities represent back-end components. Blue entities are external services the system depends on and white entities are Kielikone's services that are not hosted in Heroku but interact with MOT+).

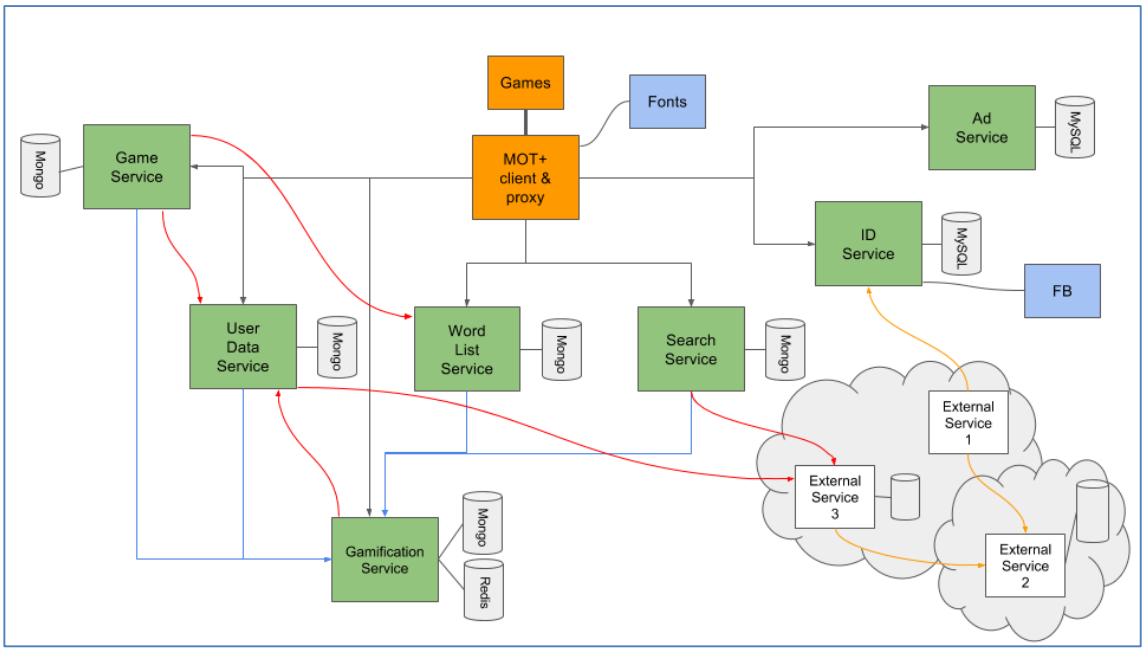

Figure 13: MOT+ system components and their communication with each other.

The components communicate with each other over HTTPS (HyperText Transfer Protocol Secure) using RESTful JSON (JavaScript Object Notation) APIs.

Front-end: generic properties

The front-ends are implemented with web technologies: JavaScript, HTML and CSS. More specifically they a single-page-applications (SPA) that have been built using the AngularJS framework in the JavaScript (EcmaScript version 5, ES5) language. The client is loaded in user's web browser and runs and interacts with the host environment through the browser's JavaScript engine. The front-end codebases include a web server implemented in Node.js using Express.js web framework that serve client assets.

Two separate front-ends exist: *MOT+ web client* and *MOT+ games UI.*

Front-end: Web client and proxy server

Web client is a SPA that contains most of the application business logic and application state. It has a responsive user interface making its UX acceptable on browsers utilizing varying window sizes. Web client fetches and manipulates data owned by the back-end services through the APIs they expose.

Web client includes a non-caching transparent reverse proxy that routes all request from users' browser to back-end services through it.

Front-end: Games client

Games client contains the game UIs. The games utilize HTML5 canvas and Phaser game engine to create the vocabulary learning games.

Back-end: generic properties

All the back-end services are implemented with Python (2.7) using Flask web development micro framework. Back-end services expose themselves to users via RESTful APIs. The data transfer formats, data representations, and API semantics vary between different back-ends depending on their role in the system, the design decisions of the APIs at the time they were first implemented, and the preferences of the implementing party responsible for a specific component. The back-ends persist their data in MongoDB document databases, MySQL relational databases or Redis key-value stores depending on the service. Most APIs provide a CRUD interface to the data their host service owns. Security is provided by using only HTTPS protocol and by signing every request with JSON Web Token (JWT) technology.

Back-end: game service

Game service is responsible for creating and persisting game sessions. Game data is stored in a MongoDB database.

Back-end: user data service

User data service is responsible for persisting MOT+ users' data such as user names and memberships in user groups. User data is stored in a MongoDB database.

Back-end: gamification service

Gamification service receives gamification events such as game scores from other back-end services, processes them and stores the data. It exposes the gamification

data as scores, achievements and leaderboards to the front-end components. The gamification data is stored in MongoDB and Redis (leaderboards) databases.

Back-end: word list service

Word list service persist the word lists that MOT+ users create. The data is stored in a MongoDB database.

Back-end: search service

Search service exposes an API for performing searches to the company's dictionary content. Part of the dictionary data is stored within the service, in a MongoDB database and some search queries are proxied to another back-end outside of MOT+'s system.

Back-end: ID service

ID or identity service provides user authentication and session management services for MOT+. The account and session data is persisted in a MySQL database.

Back-end: Ad service

Ad service provides MOT+ with web advertisement services. Ad data is persisted in a MySQL database.

Technology stack

All system components combined, the technology stack is as follows.

Front-end components technologies:

- AngularJS web framework
- Phaser.js game engine
- SASS written in SCSS format
- HTML5
- Grunt task runner
- Express.js framework running on Node.js

Back-end components written in Python

- Flask micro framework
- Selection of Flask extensions

Persistence

- MongoDB
- MySQL
- Redis

Testing code written in JavaScript and Python

- Protractor end-to-end testing tool
- Jasmine testing library
- Karma test runner
- unittest test library

The front-end technology stack is built on the typical AngularJS 1.x ecosystem's technologies that date back to the time of the inception of the project in 2014. The developers write features using JavaScript, HTML and SASS. The resulting assets are then converted into a deployable entity ready for users' web browsers by executing various post-processing scripts with Grunt task runner. The final assets are served to the users' browsers with a web server written with Express.js framework that is running on Node.js runtime. Vocabulary game code has been implemented with the help of Phaser.js game library.

The back-end technology stack is built on the Flask micro framework ecosystem. Flask is a web framework that when combined with an appropriate WSGI (Web Server Gateway Interface) web server can be used to implement back-end functionality. MOT+'s back-end services utilize many Flask extensions that provide features that help with, for example, implementing RESTful APIs.

Different data persistence solutions are used depending on the needs of a particular back-end service. Data persistence to MOT+ is provided by Heroku DBaaS (database-as-a-service) add-ons.

4.3.4 Source Code Hosting and Development Workflow

Source code for all system components is hosted in social coding service Github's Git repositories. Continuous integration is provided by CircleCI, a service that integrates with Github and can automatically run various tasks against a code base such as test runs upon a commit in a Github repository.

The developers use Github flow to introduce changes to the application codebases. Github flow is a branch based workflow that utilizes Git branches and Github pull requests and defines conventions to them in a structured manner [12].

The hosting environment Heroku is integrated both to Github and CircleCI (see section 4.3.6).

4.3.5 Hosting and Deployment

All of MOT+ components are hosted in Heroku. Deployment system depends on the integration capabilities of Heroku, CircleCI and Github.

Heroku applications

Each component has at least two Heroku applications in Heroku: a production environment application and a staging environment application. All applications are hosted in the European *region*. See Figure 14 showing the applications of every MOT+ system component in the Favourites listing and deployment activity details for the MOT+ web client production app.

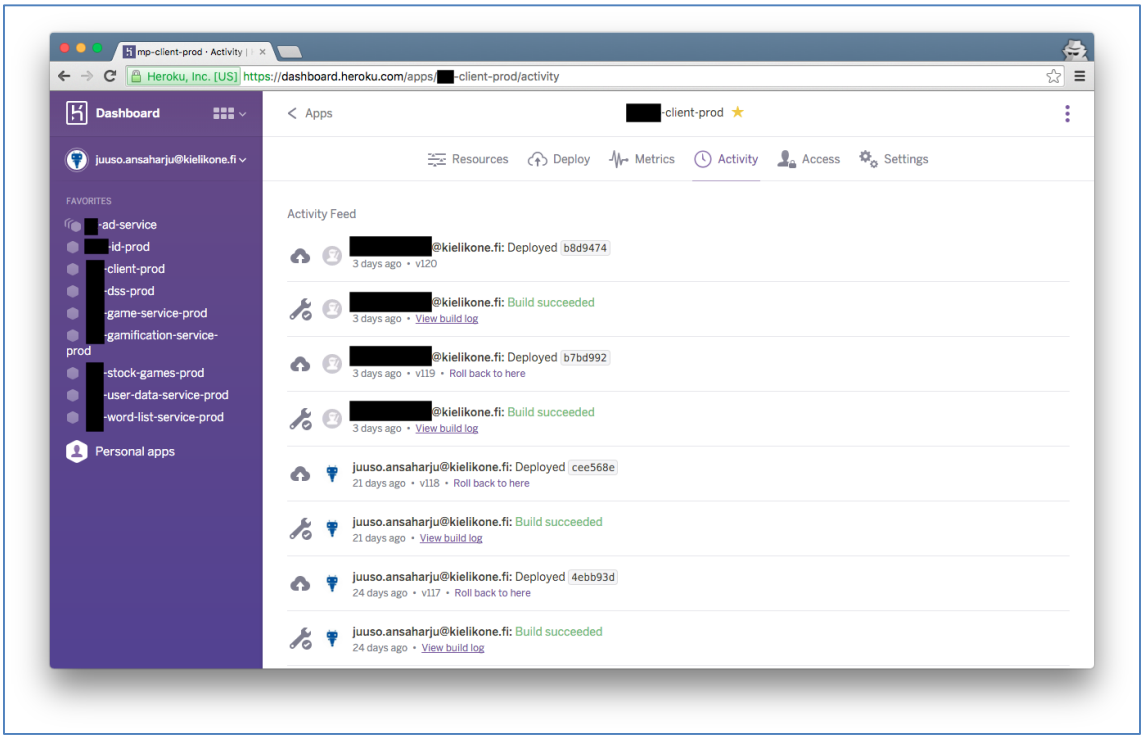

Figure 14: Screenshot from Heroku Dashboard.

All production apps run on paid dyno plans. Web client application runs on a single *1x Professional dyno* and all the rest run on *Hobby dyno*s. Web client application has been scaled up unlike other applications as it functions as a proxy for all the other components and receives their traffic as well. Staging applications run on free dynos. This scaling set-up has been appropriate to serve the user base of MOT+ sufficiently.

Development deployments

Heroku's Review apps (see Figure 15 in which references to Github by pull request numbers can be seen) feature has been enabled for the Web client's staging application. With it, an application instance is automatically created when a new pull request is opened in Github in the repository that contains the code base for the Web client.

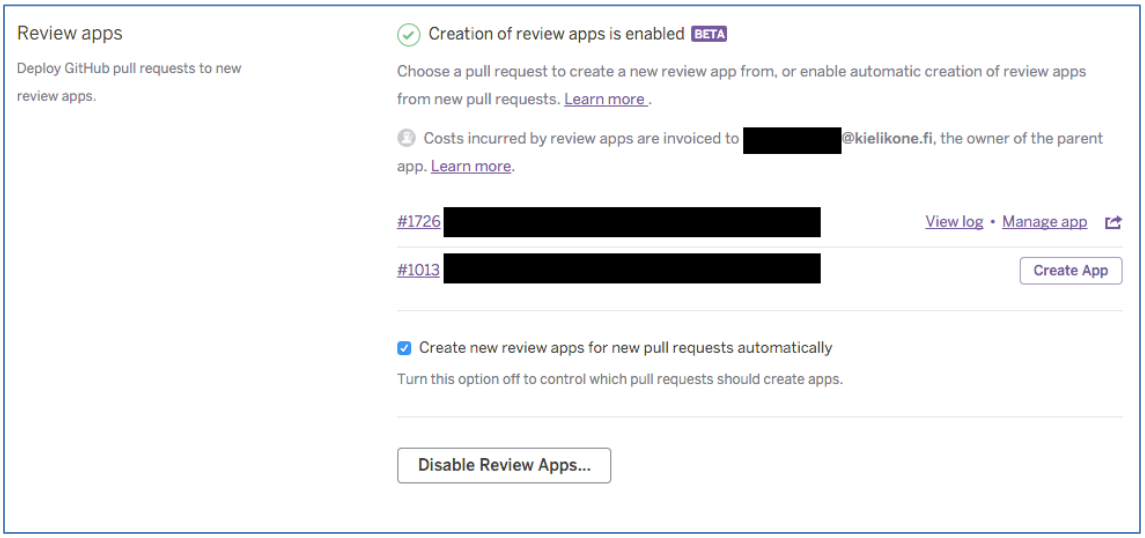

Figure 15: Screenshot from Review apps section in Heroku Dashboard's application deployment configuration view.

Web client is automatically redeployed to the Review app every time a new commit is pushed to the branch that the pull request is based on. The review app inherits all configurations from its parent application.

Staging deployments

Deployment to the staging environment apps has been automated utilizing CircleCI. Each time a new commit is pushed to the master branch in Github for any of the components' repositories, CircleCI will deploy the code base to Heroku to the corresponding staging environment application. CircleCI is configured per codebase by adding a configuration file to the root of the repository. The configuration is read and acted on by CircleCI when a new commit is pushed to a repository on Github that has been connected to CircleCI.

Production deployments

Deployments to the production environment are done manually by the developers from their workstations by pushing to the Git repositories Heroku exposes for the applications.

5 PaaS Questionnaire

A questionnaire regarding the introduction and use of Heroku at Kielikone in the MOT+ development project was created by the researcher and issued to selected developers in Q1 2016. The motivation for the questionnaire was to gather feedback about the PaaS evaluation study and the use of Heroku to understand the benefits and drawbacks the introduction of PaaS has brought to the company's software development as well as to arouse input for improvements to the current hosting and deployment practices. The questionnaire was directed towards developers and is technical in nature. The original questionnaire and its results are included in Appendix 1.

The questionnaire was divided into four categories all of which had several questions or topics within. The categories were:

- Heroku's features,
- Heroku's general properties,
- Heroku's add-on and integration system,
- Kielikone's adoption of Heroku.

5.1 Respondents

The questionnaire was issued by email to three application developers and one operations specialist who were working with MOT+ development as Kielikone's employees and had been exposed to Heroku. Of the four respondents, three provided processable results: one operations specialist in charge of MOT+'s production environment's hosting in Heroku, one developer with front-end development focus who had extensive experience with Heroku and one developer with back-end development focus who had some experience with Heroku.

The different respondents were referred to as:

- Respondent 1, the operations specialist,
- Respondent 2, the developer with front-end development focus,
- Respondent 3, the developer with back-end development focus.

The codenames for the respondents are used to identify the source of comments in the later sections.

5.2 Results of the Questionnaire

The results were processed by each questionnaire topic category by paraphrasing the answers provided by the three respondents. The respondents were asked to describe their involvement regarding each topic and give out their personal views about the positive and negative experiences they had had.

5.2.1 Heroku's Features

The questions in the first category inquired the respondents experience and opinions about the feature set that Heroku provides. In particular, the researcher wanted to gather information about using Heroku Dashboard and Heroku command-line tool to manage applications and their settings.

Involvement

Respondent 1's had had experience with basic application management including managing application collaborators, scaling dyno configurations, adjusting environment variables, setting custom domains and doing deployments with Git. Respondent 1 had also experience interacting with Heroku both through the Dashboard as well as with CLI tool.

Respondent 2 had used Heroku's application management features extensively by creating applications, configuring them (environment variables and add-on configuration), scaling the applications (on/off), using the CLI tool (for viewing logs), manipulating access control settings, doing Git based deployments and deployment rollbacks. Pipelines and Review apps were not included in the set of features used by the respondent.

Respondent 3 had only limited experience with scaling running apps, using the CLI tool, deploying using Git, and adjusting environment variables.

Pros and cons

Respondent 1 gave very positive feedback about Heroku's core features stating they were "really straightforward", "very easy [to use]" and "useful". Environment variable management was hailed by the respondent as "one great part of Heroku". The lack of Git diffs in deployment history view section was considered odd. The respondent, though having used Heroku CLI tool, preferred to use the Dashboard primarily. Negative feedback was given for the user interface for manipulating environment variables in the Dashboard, which the respondent described as an annoyance.

Respondent 2 considered Heroku's core features easy-to-use, useful and altogether positive. Especially, he highlighted the possibility of using Git as the deployment tool as well the deployment rollback feature. Praise was given also to collaboration management. Criticism was given for potential for confusion in cases where an application is dependent on a large number a configuration variables and add-ons hinting that the Dashboard's user interface is not optimal for such situations. The respondent noted that even though it is easy to scale the apps, it is hard to know when it should be done and by what amount. The respondent also stated that Heroku's core features do not help much in managing deployment orchestration in situations where there is a need to spin up multiple customized versions of same applications. Heroku CLI was criticized by the respondent for the fact that using it with multiple user identities from a single machine required OS level configuration work from the user. The respondent also casted doubt on Heroku's application servers' capability of running heavy running post-deployment tasks should those be needed at some point. Collaboration management was thought to require some added functionality to make organizational level operations easier. The lacking possibility to group applications by a category in the Dashboard was reported by the respondent.

Respondent 3 provided no opinions about the benefits and drawbacks of Heroku's core features.

Improvement ideas

Respondent 1 did not provide any direct improvement ideas, but did suggest that finding ways to quickly see differences in codebases between different deployments would potentially be useful.

Respondent 2 suggested trying out Pipelines and App Review features to see the benefits they provide for deployments; investigating performance monitoring features that open up for an app when it scaled up from Hobby dynos; testing the post-deployment hook to build the MOT+ Web client component on server; and mitigating the problems that environment variables caused by add-ons by not using them directly in code.

Respondent 3 provided no opinions about the benefits and drawbacks of Heroku's core features.

5.2.2 Heroku's General Properties

The questions in the second category inquired the respondents experience and opinions about the general properties of Heroku. In particular, the researcher wanted to gather information about the ease-of-use, platform support, system availability, troubleshooting possibilities and features related to setting a production-level hosting environment in Heroku such as SSL support and ability to use custom domain names for applications.

Involvement

Respondent 1 was very knowledgeable about the topics having done stack upgrades and application troubleshooting, being responsible for production applications availability and setting up SSL endpoints, custom domain names as well as utilizing Heroku's maintenance mode.

Respondent 2 had had experience with some of the topics in the category including utilizing Heroku's logging system for debugging and being involved in upgrading stacks.

Respondent 3 had had little experience with the topics in the category with debugging being an exception.

Pros and cons

Respondent 1 stated that overall user experience was good along with the documentation Heroku provides for its users. SSL endpoints that were set up for some applications had “worked as expected”. The respondent gave negative feedback for the stack upgrade process that was forced by Heroku deprecating old technology. Criticism was also voiced about the difficulties in setting up custom domains for applications that had SSL endpoints activated for them. There was some uncertainty voiced by the respondent about the reliability of the maintenance mode feature.

Respondent 2 had a positive view about the ease-of-use of Heroku appreciating the large community-based support and Dashboard that is simple to use. The respondent considered these aspects especially useful when stack upgrades were being performed using the CLI tool. The respondent felt that Heroku’s uptime was on good level and communication about system maintenance from Heroku had been sufficient. Critique was offered for occasional problems with accessing the add-ons’ management interfaces through the Dashboard and the lack of graphical log viewing tool native to Heroku. The need to depend on multiple add-ons was considered annoying – it would often be more convenient if the features provided by the add-ons in use were provided by Heroku itself.

Respondent 3 highlighted the ease-of-use of deployments are rollbacks and described troubleshooting of problems difficult. The respondent rated the availability of the platform positively.

Improvement ideas

Respondent 1 hinted that custom domain names for applications accessed via HTTPS would need more work to function properly. Also, getting more insight about deployment build failures’ causes and the correct usage of maintenance mode would be useful according to the respondent.

Respondent 2 suggested extending the use of CLI tools stating that it would provide a more unified interface to manage all setting including those provided by the add-ons.

Respondent 3 suggested utilizing the maintenance mode feature.

5.2.3 Heroku's Add-on and Integration system

The questions in the third category inquired the respondents' experience and opinions about the add-on and integration system of Heroku. In particular, the researcher wanted to gather information about the different service integrations that enable automatic deployments and about the add-on ecosystem built that is built around Heroku.

Involvement

Respondent 1 was not familiar with the automated deployment possibilities but had some experience in managing application add-ons.

Respondent 2 had experience with setting up and configuring multiple add-ons.

Respondent 3 had experience with automatic deployments to Heroku set up with CircleCI service.

Pros and cons

Add-ons received critique from respondent 1 who considered managing the add-ons more difficult than managing their parent Heroku applications. The fact that that add-ons are external to Heroku and all have their own management UIs and different user experience was considered problematic.

Respondent 2 felt that the selection of different kinds of application add-ons provided by Heroku was rich and appreciated that there are competing products to choose from. The respondent said that add-on installation process was very easy. However, varying pricing models and feature sets in different add-ons in the same category was perceived problematic. Also, heavy utilization of add-ons, as is the state of affairs with MOT+ it consisting of numerous Heroku applications, produce lots of maintenance overhead as each add-on adds its own environment variables to the application and has their own distinct management interfaces.

Respondent 3 stated that automatic deployment system showed potential, but could be improved.

Improvement ideas

Respondent 1 suggested that better ways to manage Heroku add-on should be found to better utilize their functionalities.

Respondent 2 suggested that services provided by the add-ons should be combined across all the Heroku application so that there would be only one point of access to manage whatever a specific add-on product provides.

Respondent 3 provided no improvement ideas.

5.2.4 Kielikone's Adoption of Heroku

The questions in the third category inquired the respondents' experience and opinions about how Heroku PaaS has been adopted at Kielikone. In particular, the researcher wanted to gather information about the respondents' insight to how the adoption process had developed, and what kinds of effects Heroku had exerted on the pre-existing development and the hosting processes and procedures.

Involvement

Respondent 1 did not participate in the PaaS evaluation process. The respondent cooperated with the developers in setting up hosting of applications on Heroku.

Respondent 2 was collaborating with the researcher in the later phase of the evaluation process by doing first test deployments to Heroku and assessing how to use it in MOT+'s development flow. The respondent had extensive experience in debugging MOT+'s components hosted in Heroku. The respondent had encountered Heroku security aspects many times.

Respondent 3 had some experience with debugging application in Heroku's environment.

Pros and cons

Respondent 1 appreciated the automation possibilities Heroku provided for improving hosting procedures and considered Heroku very suitable for hosting in a micro service architecture oriented environment. The respondent rated the training received as sufficient. Negative feedback was given by the respondent with regards to situations were operations staff and developers failed to communicate how to properly set up environment variable configurations for applications.

Respondent 2 rated Heroku's documentation high regarding information about the platform security. The foremost security concern was due to the fact that all applications, including applications used for testing purposes, are accessible by anyone in the public Internet knowing the host name for the application. Any access control would need to be implemented on the application level or by utilizing an add-on. The respondent appreciated the Dashboard as it allows easy and visual access to basic configuration information about an application that previously would have been only accessible with command line interface through a remote access. Dashboard also makes it possible for less technically oriented personnel to take over maintenance tasks that would previously require a developer to perform. The respondent felt that Heroku has made it possible to move significantly towards continuously releasing new features to the end-user. Deployments using Git and easy tools such as rolling back problematic deployments had been the key. Heroku had been very suitable for our chosen system architecture.

Respondent 3 criticized Heroku's debugging capabilities and praised its potential for automation and integration with CircleCI. The respondent raised concern about the security implications.

Improvement ideas

Respondent 1 suggested increasing communication between operations staff and developers.

Respondent 2 suggested changes to MOT+'s system architecture to make debugging of request sequences easier. Particularly, the respondent hoped some of the common functionality in all back-end services could be moved to the client component's server. The respondent also advised to look into ways of mitigating the security problem

caused by the fact that all Heroku applications are publicly available. The respondent suggested that additional training to use Heroku and keeping track with its new features should be arranged. Newer features such as Review apps and Pipelines should be tried out.

Respondent 3 provided no improvement suggestions.

5.2.5 Additional Comments

Respondent 1 summarized his experiences with Heroku as positive and stated that it suits well as a hosting environment for MOT+ service. Having had the background in doing more explicit and manual server administration work, the respondent appreciated the smaller workload that Heroku offers to system administrators. The respondent considered the use of Heroku in MOT+ service as a success.

Respondent 2 thought that PaaS suits the MOT+'s development process and the benefits outweigh the drawbacks. The respondent underlined the importance of adding even more automation to the process.

Respondent 3's opinion was that Heroku is suitable for development, but might not be as suitable in production environment if user base grows.

The answers provided by the respondents raised interesting points about how Heroku had been used at Kielikone and how developers had perceived its capabilities, limitations and its effects to efficient development of web software. It is also worth noticing that the respondent base to the questionnaire was very limited. Only two developers, in addition to the researcher, had studied Heroku in any significant detail. This was visible in the scope of results provided.

The results are analysed in greater detail in the next section.

6 Results and Conclusions

PaaS evaluation study including the pilot project in which Heroku was used extensively for web application hosting purposes provided a large resource of information for analysis. The work performed by the researcher in collaboration with the other developers to set-up web applications in Heroku and using it daily as an integral part of the development flow for almost two years gave MOT+ team a good understanding of the benefits and drawbacks of using a PaaS product.

The lessons learned through everyday work gathered using the questionnaire are parsed in the next three sections, 6.1 to 6.3. Researcher's own comments regarding each topic are included in the analysis.

6.1 Benefits of Using Heroku

Heroku was considered by the respondents to be "useful", "easy to use" and altogether a positive experience. The best features were the Dashboard and especially its environment variable editor, the collaborator management tool and the activity feed with rollback functionality; and the deployments using Git. Dashboard was considered easy enough to use in order to offload some hosting maintenance work to non-technical staff. The support documentation received positive comments. It had provided a valuable help during the stack version upgrade process. The CLI tool was listed a nice addition to the toolset even though Dashboard was preferred by the respondents as the go-to interface for interacting with the service.

The add-on selection was considered rich and with enough options available to choose from. Add-ons were also told to be easy to connect to applications.

The deployment automation possibilities that Heroku offers were thought to be suitable for the MOT+'s technology stack and the system architecture. Deployment integrations between CircleCI, Github and Heroku worked and were considered convenient.

The researcher shares the opinions of the respondent with regards to the benefits of Heroku and underlines the importance of fast painless deployment using Git. Also, Heroku's newer features such as Review apps and Pipelines were proved useful by

simplifying the deployment flow even further. The researcher has found the CLI tool a faster-to-use interface to perform management tasks than Dashboard, but acknowledges that command-line approach may not be the preferred solution for all. The potential of service integrations with e.g. Heroku and Github has not yet been exhausted and there are even more benefits to be reaped with additional automation.

6.2 Drawbacks of Using Heroku

Some parts of Heroku's feature set received criticism from the respondents. The drawbacks for Dashboard were: cumbersome UX to manipulate the application's environment variables especially if variables were present in high numbers, cumbersome management of access control permission in an environment with a large number of applications such as is the case with MOT+. It was noted that sometimes the problems associated with using the environment variable editor in Dashboard were actually more due to insufficient communication between developers than UX limitations in the Dashboard. The CLI tool received criticism as its usage with multiple user identities from a single workstation required extra configuration that was not immediately obvious.

The add-ons system was criticized for its fragmented nature – add-ons need to be configured for each application separately and since the add-ons themselves are provided by parties external to Heroku, they all have their own management UIs. Heroku provides a single-sign-on system for easier access to the add-ons, but sometimes authentication sessions did not work as expected. This caused occasional confusion. The add-ons are offered without a unified pricing model, which had caused problems in choosing the best add-ons when several add-ons provided a similar service.

One type of criticism was the lack of features. In particular, some sort of a tool to manage or create multiple applications simultaneously was in the wish list. The ability to handle groups of applications would have made some management actions faster to execute.

The stack upgrade process caused some gray hairs. The fact that Heroku might make changes to their infrastructure and that their customers will need to adjust to it, is a common trait of cloud based services that can have a negative impact on a customer's software development process.

Troubleshooting problems was considered problematic in Heroku's environment. As Heroku provides limited logging services itself, developers were forced to rely on various add-ons to debug errors. It was also noted, though, that most of the problems in debugging was not caused by Heroku itself. Heroku just does not provide much tooling to debug, for example, a networking problem between different applications.

One category of drawbacks was the implications that Heroku exerted on application and network security. As all Heroku application are publicly accessible, the application level access control mechanics are needed to safeguard the applications from unauthorized access. Also, since Heroku guarantees no permanent dedicated IP addresses for applications, typical access control measures involving whitelisting or blacklisting server connections based on IP addresses were impossible to implement without resorting to an add-on.

The researcher agrees with the respondents about the pain points. Dashboard, while providing easy access to manage applications, is not perfect. Occasional slow response times of the Dashboard can be added to the list of UX annoyances reported by other developers.

While economical aspects of using PaaS were not in the scope of the thesis project, it must be noted that it is easy to increase costs in a system with a large number of applications all which might be connected to a large number of non-free add-ons. Luckily, dynos and add-on plans support a pricing model where services can be scaled up dynamically according to how system's performance requirements grow.

Some drawbacks stated by the respondents can be alleviated by starting to use the advanced automation integrations that Heroku provides for Github.

6.3 Improvement Suggestions

The improvement suggestions are presented on two lists. The first list contains the suggestions given by the respondents of the questionnaire. The researcher's comments are included in italics for each suggestion where they apply. The second list is composed of researcher's own opinions and covers only topics that were not highlighted by the respondents.

6.3.1 Respondents' Improvement Suggestions

The main improvement suggestions made by the respondents are listed below:

- Find ways to enable Git diffs to see code level changes between deployments. *This can be achieved by configuring auto-deployment from Github in the Dashboard,*
- Try out the Pipelines feature. *The researcher has enabled Pipelines for Ad service component in Q1 2016. Initial experiences look promising*,
- Try out the Review Apps features. *The researcher has enabled Review apps for multiple components. Initial experiences in Q1 2016 look promising*,
- Make use of Heroku's own application performance management capabilities that are automatically activated when application's dynos are scaled above *Hobby* level. *One MOT+ component is running on Professional dyno since Q1 2016*
- Try out utilizing the post-deployment hooks to perform processing that is now done locally. *The researcher is looking into potential use cases for utilizing post-deployment hooks as of Q1 2016,*
- Stop using the environment variables created by the add-ons as is – instead, rename or map them to the other environment variables to be more descriptive and context aware when used in code,
- Implement application level handling of domain redirects,
- Start using maintenance mode during service updates. *This practice could be reinstated easily*,
- Utilize the CLI tool more in order to provide a more unified interface to Heroku and its add-ons. *Scripts to e.g. perform multiple deployments simultaneously could be useful*,
- Simplify management of add-ons. *It has proven convenient to group multiple applications' add-ons under a single add-on account. This practice could be continued. Heroku's enterprise features might provide additional tools for this*,
- Increase communication between developers and operations staff,
- Simplify MOT+'s system architecture to make debugging easier,

- Find ways to mitigate the security problems caused by Heroku's open doors policy. *Application level access control mechanism such as requiring authentication with HTTP Basic authentication have been put into place*, and
- Invest in training of the staff to make most out of Heroku and keep developers up-to-date with new features and other changes.

To summarize, the improvement suggestions provided by the respondents can be divided into two categories: (1) the suggestions that encourage the team to utilize Heroku's features more extensively (either by starting to use features that are not in use currently or by altering the way currently used features are used), and (2) the suggestions that encourage either altering the system that is hosted in Heroku to make it easier to manage and improving collaboration between the users of Heroku at Kielikone.

All of the improvement suggestions were useful and implementable as concrete improvements. Especially the improvement suggestions in the first category can easily be implemented incrementally along with other development activities.

6.3.2 Researcher's Improvement Suggestions

The researcher's own improvement suggestion are listed below:

- Audit the hosting deployment procedures. Outside point-of-view could prove to be valuable,
- Look into Docker and other container technologies to make company less dependent of a particular hosting provider in the future, and
- Utilize slug ignore files to omit unneeded files from deployments to speed up the deployment and build process in Heroku.

Heroku has been in use at Kielikone for approximately two years. Lots of experience has been gained about using Heroku and that information has been collected into what can be described as unwritten good practices. It might be beneficial to search for external feedback about the company's hosting and DevOps practices in the future. External feedback combined with the findings of this study could prompt new kinds of ideas and improvement possibilities.

Technological advances in hosting in cloud computing world have continued during the MOT+ project. There are technologies, container virtualization, for example, that have become popular in the development community, but have not yet been researched in detail at Kielikone. Some of these new technologies could extend or replace Heroku.

7 Summary

The goal of the thesis project was to improve the web development process at Kielikone Oy by evaluating and piloting a PaaS product as a solution to fulfil company's web application hosting and deployment needs.

Heroku was chosen as the PaaS product that was then used in a long and relatively large and complex web application development project to build MOT+, a language learning service to extend Kielikone's product catalogue. Heroku turned out to be able to fulfil the needs and expectations set for it, and, according to the results of the questionnaire issued to MOT+'s technical staff as part of the thesis project, Heroku was considered easy-to-use, reliable, and a good fit for the technologies and the system architecture in place. Despite the occasional criticism and worries about the suitability of Heroku as hosting environment should the user base grow enough to make Heroku's pricing model challenging economically, it can be concluded that the evaluation was a success. Company's web development process was improved as the deployment times decreased.

The feedback from developers and the improvement suggestions generated as a result of analysing the answers to the questionnaire will be useful for the company. The improvement suggestions are aimed for, but not limited to, the continued use of Heroku as the go-to solution for hosting web applications – especially those in active development.

The analysis did not cover the pricing of Heroku and thus is ambivalent with regards to economical aspect of the software development. The respondent base of the questionnaire was limited to only three developers and much more could have been learned if the project and business management points-of-view were included as additional goals for the thesis project.

The researcher recommends Heroku as an easy-to-use tool for a modern web development from developer's viewpoint and encourages enhancing its use at Kielikone according to the improvement suggestions listed in the previous section.

References

1 Jésus, J. Navigating the IBM cloud, Part 1: A primer on cloud technologies [online]. Armonk, NY: IBM; 2012-06-20.
URL: http://www.ibm.com/developerworks/websphere/techjournal/1206_dejesus/1206_dejesus.html.
Accessed 2016-03-28.

2 National Institute of Standards and Technology. The NIST Definition of Cloud Computing. Gaithersburg, MD; 2011

3 Orlando D. Cloud computing service models, Part 2: Platform as a Service [online]. Armonk, NY: IBM; 2011-01-28.
URL: http://www.ibm.com/developerworks/cloud/library/cl-cloudservices2paas/.
Accessed 2014-06-04.

4 Heroku Inc. What is Heroku [online]. San Francisco, CA: Salesforce Inc.
URL: https://www.heroku.com/what.
Accessed 2016-03-25.

5 Heroku Inc. About Heroku [online]. San Francisco, CA: Salesforce Inc.
URL: https://www.heroku.com/about.
Accessed 2016-03-25.

6 Heroku Inc. Reference, Heroku Dev Center [online]. San Francisco, CA: Salesforce Inc.
URL: https://devcenter.heroku.com/categories/reference.
Accessed 2016-03-25.

7 Heroku Inc. How Heroku works [online]. San Francisco, CA: Salesforce Inc.
URL: https://devcenter.heroku.com/articles/how-heroku-works.
Accessed 2016-03-25.

8 Lindholm T, Virkkala R. DevOps Asiantuntijoille, Eficode Oy, Helsinki;

9 Loukides M. What is DevOps?. Sebastopol, CA: O'Reilly Media; 2012.

10 Walls, M. Building a DevOps Culture. Sebastopol, CA: O'Reilly Media; 2013.

11 Sharma, S. DevOps and PaaS: ‘Give me a platform. Let’s rock, let’s rock, today’ [online]
URL: http://devops.com/2014/05/01/devops-paas-give-platform-lets-rock-lets-rock-today/
Accessed: 2016-03-20

12 Github Inc. Understanding the Github flow. San Jose, CA: Github Inc.
URL: https://guides.github.com/introduction/flow/
Accessed: 2016-03-20

Questionnaire and answers

Questionnaire

Welcome to the questionnaire about the use of Heroku Platform-as-a-Service web application hosting provider during the MOT+ development project. The audience for the questionnaire are you, the developers and operations personnel, who have used Heroku in your work. The results of the questionnaire (your answers) will be used as the main input in analysing the experiences of piloting Heroku during MOT+'s development at Kielikone. The outcome of the analysis will be documentation in the form of a set of good PaaS hosting practices and further development ideas for application developers and operations personnel.

The questionnaire covers 26 topics are that are divided into 4 categories. You are asked to provide your own personal view regarding each topic by elaborating on your experiences from four different view-points:

1. Involvement - Describe your involvement in the topic in question
2. Pros - Elaborate on your positive experiences regarding the topic
3. Cons - Elaborate on the negative aspects regarding the topic
4. Input - Give improvement ideas, comments, critique or other relevant input

The topic categories are:

- Heroku's features
- Heroku's general properties
- Heroku's add-on and integration system
- Kielikone's adoption of Heroku

QUESTIONNAIRE TOPICS

> Background info

Describe your overall role in MOT+ development (e.g. “I am a front-end developer who has participated in MOT+ since early prototyping phase. My responsibilities have been… etc.”)

> Heroku’s features

Topic 01: Basic app management: Elaborate on you experiences with the topic (e.g. creating/deleting/renaming apps)

Topic 02: Scaling: Elaborate on you experiences with the topic (horizontal scaling with dynos)

Topic 03: Pipelines: Elaborate on you experiences with the topic

Topic 04: Review apps: Elaborate on you experiences with the topic

Topic 05: Heroku CLI (example commands: run, cert, logs): Elaborate on you experiences with the topic (command examples: run, cert, logs)

Topic 06: Deployments: Elaborate on you experiences with the topic (activity feed, diffs, rollbacks, Github integration)

Topic 07: Access control: Elaborate on you experiences with the topic (e.g. adding collaborators)

Topic 08: Settings management: Elaborate on you experiences with the topic (e.g. config variables, custom domains)

> Heroku’s general properties

Topic 09: Ease of use: Elaborate on you experiences with the topic (e.g. dashboard UX, CLI UX, documentation, support services, community)

Topic 10: Stacks & Buildpacks: Elaborate on you experiences with the topic (e.g. changing or updating stacks or experimenting with different buildpacks)

Topic 11: Availability: Elaborate on you experiences with the topic (e.g. your impressions service uptime and handling of platform maintenances)

Topic 12: Troubleshooting: Elaborate on you experiences with the topic (e.g. your impressions about the tools and data provided by Heroku to help with debugging issues)

Topic 13: SSL endpoints: Elaborate on you experiences with the topic (e.g. setting up SSL certs)

Topic 14: Custom domain names: Elaborate on you experiences with the topic (e.g. setting up custom domains)

Topic 15: Maintenance mode: Elaborate on you experiences with the topic (e.g. utilising maintenance mode pages during MOT+ service breaks)

> Heroku’s add-on and integration system

Topic 16: Automatic deployments: Please elaborate on you experiences with the topic (e.g. deployments from Github and Dropbox)

Topic 17: Add-ons management: Please elaborate on you experiences with the topic (e.g. setting up and maintaining add-on services such as monitoring, database and logging tools)

> Kielikone’s adoption of PaaS

Topic 18: Evaluation process: Elaborate on your thought about the PaaS evaluation process that resulted in choosing Heroku for the pilot

Topic 19: Debuggability: Elaborate on your experiences with debugging MOT+ in Heroku’s distributed PaaS hosted environment

Topic 20: Security: Elaborate on your thought about security considerations with Heroku

Topic 21: Co-operation: Elaborate on your experiences with co-operation between developers and operations staff when dealing with Heroku

Topic 22: Training: Elaborate on your thoughts about the training company has provided for Heroku and using PaaS in general

Topic 23: Automation: Elaborate on your thoughts about how Heroku helps with automating deployment and hosting procedures

Topic 24: Continuous integration: Elaborate on your thoughts about how using Heroku has affected the continuous integration process

Topic 25: Continuous deployment: Elaborate on your thoughts about how using Heroku has affected the deployment frequency

Topic 26: Implications to system architecture: Elaborate on your thoughts about how using Heroku has affected the system architecture of MOT+

> Wrap up

Add any extra comments regarding Heroku, PaaS, hosting, or DevOps in MOT+ development

Answers, part 1

> Background info

My role in MOT+ is that of production service administration.

> Heroku’s features

Topic 01: Basic app management

Basic app management is really straightforward in Heroku, no complaints there.

Topic 02: Scaling

Scaling with dynes is also very easy.

Topic 03: Pipelines

No experiences, pipelines haven’t been a part in my Heroku responsibilities.

Topic 04: Review apps

No experiences.

Topic 05: Heroku CLI (example commands: run, cert, logs)

CLI commands have worked as expected. I’ve used the CLI only every now and then as most usually needed functions are available from the GUI.

Topic 06: Deployments

Activity feeds and rollbacks have been useful. Are there diffs also? I haven't found them… I haven't connected deployments to Github.

Topic 07: Access control

Adding collaborators is a trivial matter in Heroku. (Still I usually first forget it when creating a new app.)

Topic 08: Settings management

Config variables are one great part of Heroku. The UI with them sucks a bit because many variable values are long strings and the string field is narrow. Not a big deal, just an annoyance. Custom domains are crucial in production apps.

> Heroku's general properties

Topic 09: Ease of use

General UX is good, so is the documentation in general. No experiences on the community nor the support services that I'd recall.

Topic 10: Stacks & Buildpacks

There were issues in updating the stack (cedar-10 to cedar-14), but they were eventually solved by the developers. Also build packs have needed some updates (gevent/eventlet). It's often confusing when a build fails; happily that doesn't happen too often. Sometimes build fails first giving some weird error but then succeeds in the next similar run, with no changes done by the builder.

Topic 11: Availability

There have been varying issues in service uptime but they've been almost always caused either by software problems or by app sleeping.

Topic 12: Troubleshooting

Add-ons like Papertrail have been useful in troubleshooting and have worked OK. No further comments from the operations perspective.

Topic 13: SSL endpoints

SSL endpoints have worked as expected, no hassle.

Topic 14: Custom domain names

We had issues on https on some domains. They needed discussion with the DNS provider but were after all not solvable in the DNS nor in Heroku. Https excluded, custom domain names have worked well.

Topic 15: Maintenance mode

Maintenance mode was used at least in some DB updates. It seems to have worked but due to the service consisting of many apps I'm still not sure whether the maintenance mode is fully reliable or not.

> Heroku's add-on and integration system

Topic 16: Automatic deployments

No experiences, I haven't created any automatic deployment chains.

Topic 17: Add-ons management

Managing add-ons is harder than general Heroku management, due to the varying approaches to UI, user management etc. Some add-ons would have needed and would still need further studying to best utilise their functionalities.

> Kielikone's adoption of PaaS

Topic 18: Evaluation process

I wasn't involved in the process (or then I've forgotten about it).

Topic 19: Debuggability

No experiences.

Topic 20: Security

No Heroku-specific issues on security come to mind. We use the European service locations, it's good that they are available.

Topic 21: Co-operation

At some point there were some issues with new Heroku environment variables missing from the production apps. That can be considered a communication or procedure failure – anyway it complicated deployment a bit at that time.

Topic 22: Training

Training was needed and provided mostly in the beginning, no further comments.

Topic 23: Automation

At least for more trivial applications like info sites, the hosting is really easy. Not surprisingly, complicated applications require more effort.

Topic 24: Continuous integration

Developer stuff, no thoughts from the operations perspective.

Topic 25: Continuous deployment

Staging or production? And compared to what? Can’t say anything very meaningful in general.

Topic 26: Implications to system architecture

Dividing the architecture to different Heroku apps is probably a very good thing for clear distribution of work. Of course it may also have complicated some matters in how the whole system works together.

> Wrap up

While I haven’t compared Heroku to any alternatives, I believe it’s been a good choice for MOT+. Having gotten used to doing all configuration in the server side manually, I was a bit fearful of the possible limitations with PaaS, but those fears have been quite dissolved. Personally I do not miss e.g. the ability to configure a web server in the traditional manner. So far we’ve been, I believe, able to keep also the service costs reasonable and even cut them down by taking some unnecessary labour away from some Heroku add-ons. In conclusion, adopting Heroku for MOT+ has been a success in my opinion.

Answers, part 2

> Background info

I am a front-end developer who has participated in MOT+ since early prototyping phase. My responsibilities have been implementing the MOT+ SPA client using Angular framework.

Also I've been involved in the overall system design and implementation of some of the backend services of MOT+ written in Python using Flask framework and its various extensions.

Lastly, I've been involved in the migration of MOT+ authentication service from Auth0 to our custom implementation. That work included implementing a client module for authentication and taking part in the implementation of the server.

> Heroku’s features

Topic 01: Basic app management

Involvement: I've created new apps for different services and also copied app instances for testing purposes.

Pros: Creating apps is really easy. Just a couple of clicks. The configuration of the Procfile, plugin dependencies and (depending on the app) setting the environment variables can be a bit tedious when creating the app. Though I think that the env vars and plugins are copied as well when you make a copy of an app, which helps. There might be some configuration involved.

Cons: When you have an app that has a lot of plugins and environment variables, it can get confusing. Also the plugins are so easy to add, that an app might have a bunch of plugins set up that it doesn't actually need or even use. Also creating app in-

stances automatically for e.g. different development branches is quite difficult though it is possible.

Input: There are new features of Heroku like app preview and app pipelines for moving apps from staging to production. Those will most likely simplify and make it easy to automate things.

Topic 02: Scaling

Involvement: I've only used horizontal scaling to turn the app off and on (0 or 1 dyno)

Pros: It's really easy and fast. And you can do it from the terminal as well, even though I only used the web dashboard.

Cons: Pricing might be surprising and it's hard to know how many dynos is enough and how does it actually improve performance. If the performance bottle neck is the DB server, it doesn't really help scaling the app server. New relic plugin can give info on what actually is slow in the service, but that is also a new plugin in the Heroku app and it's subscription has to be scaled up to get all the info and benefits.

Input: There are some additional performance graphs provided by Heroku when the dynos are bumped from Hobby to Pro (something we just noticed). Price goes up, but the added insight might be valuable specially in a production environment.

Topic 03: Pipelines

Involvement: I know in general what the pipelines are for, but I haven't tried them.

Pros: If it works like it says on the box, the pipeline should simplify the production deployments a lot.

Cons: No comments

Input: No comments

Topic 04: Review apps

Involvement: I know in general what the review apps are for, but I haven't tried them.

Pros: This should solve the app instance per development branch problem. This way QA can basically verify new features faster and more isolated and that brings us closer to continuous deployment. (using the pipeline of course)

Cons: In a microservice architecture there might be a need to deploy app review from e.g. client and 2 services to see the actual feature in action. I think this would require custom orchestration to get the correct app reviews up and running and networking configured correctly so that the system uses the changed parts from app review and rest of the parts form staging.

Input: If the microservice orchestration can be solved, this could be quite powerful and fast way to manage and continuously deploy a microservice system.

Topic 05: Heroku CLI (example commands: run, cert, logs)

Involvement: I haven't used the CLI that much. Only for logs basically.

I guess some administrative tasks could be automated via CLI. Also some HC Linux guys are faster using the command line.

Cons: If you want to use different Heroku credentials (personal and company) you have to fiddle around with the certs and hosts.

And if any of your build scripts (buildcontrol) relies on those custom hosts, other developers need to setup the same host configuration.

Topic 06: Deployments

Involvement: I've done deployments and rolled back a few. Both client and backend service deployments

Pros: Deploying via Git push is so great. It's really easy to control the different deployment environments with Git remotes. Rolling back is instantaneous and seems quite reliable and it always creates a "new deployment" in the Activity history, so you can roll back a rollback and see the history of the deployments quite nicely.

Client deployments via buildcontrol lose the Git hash references in Activity feed, but that's not really Heroku's problem. The client build can be done on the Heroku server via post deployment hook. Question is, is it possible that some build tools might not install on the Heroku server? Or is there enough resources to do the build?

Input: Try the post deployment build for the client.

Topic 07: Access control

A: Involvement: I've given collaborator rights and transfered ownership of an app.

Pros: The collaboration management is super simple. Just add an email.

Cons: It would be nice to have some organization wide settings. Now all the collaborators need to be added/removed one by one. Would be nice to have assign organizations/teams as collabora-

tors. Also in a microservice architecture, you have to go through every app/service and control the rights separately. Would be nice to have control to a set of Heroku apps.

Topic 08: Settings management

A: Involvement: I've done environment variable configurations and plugin configurations. Not so much domain things.

Pros: Adding configuration variables and plugins is quite easy. Plugins usually automatically set up some default env vars for the plugin to work.

Cons: Apps with multiple plugins and other settings might have tons of environment variabless. When there's more than 10 variables, the UI for managing them is quite tedious to work with.

Input: Plugins (mainly DB plugins) usually give defaults that have some specific service provider names like MONGOLAB_URI or JAWS_DB_URI. It's a good practice to have a generic DATABASE_URI env var that is used in the actual app and copy the value from the plugin default. This way the plugin service details don't leak in to the app and confuse devs if the service provider changes in the future.

> Heroku’s general properties

Topic 09: Ease of use

Pros: There's a big community and usually always some solution if there's a Heroku specific problem. Generally the dashboard UI gets things done. Most of the UI is anyway delegated to the plugins, so the dashboard is quite simple.

Cons: because the UI is delegated to the plugins, each plugin can and most likely will have some very different looking UI and

functionality. And it's sometimes hard to figure out if you are using the service via Heroku app or if you just signed in to the service from their UI that is not connected to Heroku. Probably that doesn't really matter in most cases, but the question arises, that am I in the right place?

Input: Using the CLI to control the app and plugins might actually be more unified UI for managing the app. It's just CLI for everything. Of course there will be tons of different commands in the different plugins, but anyway it's still more unified. Of course to see all of the functionality, you'd have to see the web UI, but for basic tasks the CLI might be easier to document in the README

Topic 10: Stacks & Buildpacks

Involvement: I've updated one app from cedar-10 to cedar-14

Pros: It was really simple and I got all the instructions during a deployment. CLI informed of the old stack and gave instructions on how to update it. Also Heroku sent emails warning about the old stack going out.

Topic 11: Availability

Involvement: I haven't been involved too much in the availability monitoring.

Pros: Heroku hasn't died on us at least to my knowledge. And they send emails about any changes or problems that might be occurring. Also they have a Heroku status page that shows the current status of Heroku service with history.

Topic 12: Troubleshooting

Involvement: I've debugged the client and backend services

Pros: Heroku provides logging out of the box.

Cons: Heroku doesn't really provide any nice interface to their logs. You need to use plugins like Papertrail, Airbrake and New Relic to dig in to the problem. Plugins increase the price of the app. Also it might be weird for developers who are used to logging in to the app server and digging around in the system to not have that access.

Topic 13: SSL endpoints

Involvement: I haven't set up any SSL certs in Heroku

Topic 14: Custom domain names

Involvement: I haven't set up any custom domains in Heroku

Topic 15: Maintenance mode

Involvement: We haven't utilized any maintenance mode pages. We just let Heroku show their standard Application error page.

Input: Maintenance mode can be controlled easily from the CLI and custom error page should be also easy to set up. Maintenance mode should be set up and ready to be used in MOT+ ASAP.

> Heroku’s add-on and integration system

Topic 16: Automatic deployments

Involvement: I haven't done any automated deployments

Topic 17: Add-ons management

Involvement: I've setup and configured a bunch of plugins for different services

Pros: There's a lot to choose from and usually there are multiple providers for the same service so there's some healthy competition in the platform. Usually the plugins are very easy to install. Most often just a click away.

Cons: There's a lot to choose from and sometimes it's hard to compare the pricing and features of a service from different vendors. Sometimes the installation of a plugin with a free subscription requires a credit card and sometimes not, depending on the vendor. If there's a lot of plugins for an app, there are lots of env vars to control and each plugin has it's own UI. Sometimes it's easy to get lost. Specially if running a microservice system, where each app has it's own plugins, possibly of the same kind.

Input: The way we were able to setup new relic to only be the add-on in the client app and the rest of the backend services just used the credentials for that new relic account was really nice. Now we get a full picture of the whole system and easily access the detailed information in each service.

> Kielikone’s adoption of PaaS

The topics in the category are about how

Topic 18: Evaluation process

Involvement: I was testing the first deployments of our existing apps to Heroku

There were so many PaaS providers that it was impossible to try them all. There were a couple of good ones but if I remember correctly Heroku and Open Shift were the providers that were

most widely used and had good documentation. Open Shift was a no-go as they only had the infra available on American soil at the time. So I think we ended up with Heroku quite naturally. Juuso did most of the ground work, but to understand all the terminology and differences (specially in pricing) of the providers was really hard, just by looking at the excel sheets. So from my part it was just testing if Heroku works well in development flow, which it did.

Topic 19: Debuggability

Debugging is quite hard with the way we've set up the system. If there's an operation that involves several back-ends and something goes wrong. It's really hard to follow the flow between servers. Each server has it's own logging, but it's hard to follow a sequence with the logs being in separate places.

Also the fact that we use a proxy server to proxy the actual requests from the client to the back-ends requires an extra step of mental mapping, when debugging in the client. The proxy could be utilized to get a grasp on the server sequences by adding proper logging on that level (of course proper logging everywhere). Also the operations requiring multiple back-ends could be handled at this level, rather than chaining the back-ends together. It would be easier to follow the sequences required for complex operations.

Topic 20: Security

Heroku has quite nice documentation of their platform security and how it can be extended using plugins. Also their development center has some good articles on security. Mainly authentication related. But all security considerations apply in application development that apply when running in any other environment.

One thing that is security related is the staging/test deployments. Those apps are visible to the world. It is very practical, if we want to show work in progress to third parties etc. But it's open to the world and anyone with the URL can access it. Old Kielikone workers or partners that remember the URL can access it no problem and keep up with the development, even though they are no longer involved in the project. It would be good if there was a simple solution provided by Heroku or one of the add-ons to whitelist the users, who can access the app.

Topic 21: Co-operation

It's really good to have a web ui to your web app. Previously it was basically the app developer who would dig around in the server via CLI. With the web UI for Heroku and its add-ons even non-technical people have been peeking behind the curtains of the app without the developers even giving any instructions. This really helps with the transparency and also gives developers more time to develop, when maintenance and monitoring can be delegated somewhere else.

Topic 22: Training

Basically no training. We've learned the platform by using it. It might be beneficial to look in to some web courses or some other resources to get up to date information on the best practices and latest changes as the platform is developing all the time.

Topic 23: Automation

The new app review and pipeline functionality probably streamlines the deployments a lot.

Topic 24: Continuous integration

Using Heroku hasn't really affected the CI process yet. Only that instead of deploying the app to one of our servers, we 'Git push' it to Heroku. We could leverage the app review and pipeline features and go towards automated continuous deployment.

Topic 25: Continuous deployment

We've never deployed so often with our previous platforms. There used to be a 2-day deployment phase in the old system. Now we basically had a release every week or every other week. And if there were any hotfixes, those could be immediately deployed once they were tested. And that usually only involved one of developers who had collaboration rights. Previously the deployment could only be technically done by the server manager, but required the assistance of devs to get the configurations right etc. Also when deployment went wrong in Heroku, rollbacks were just a click away. Previously there were always cold sweat and frantic banging on the keyboard as we usually did not have any way to consistently and automatically rollback a deployment.

Topic 26: Implications to system architecture

I don't think Heroku has had a huge impact on the system architecture in general. We would have done the microservice architecture even in other platforms. But the fact that the deployments are fast and easy, it definitely encouraged towards that style of architecture. One impact it has had is the way we setup our apps. Heroku encourages to build the apps so that they are quite decoupled from the actual server they are running in. The heavy use of environment variables for example is one pattern we'll likely use even if app is not running in Heroku.

> Wrap up

PaaS is the way to go for now. With the limited resources, the savings in server maintenance and speed of deployments really

pay off the possible added costs of running a PaaS app. If we can get more automation in deployments etc. even better.

Answers, part 3

> Background info

Back-end developer also working with front end. Heroku usage mainly as a application/service developer.

> Heroku’s features

Topic 01: Basic app management

Not familiar, not used

Topic 02: Scaling

Familiar. Have done few times.

Topic 03: Pipelines

What's this?

Topic 04: Review apps

Not familiar, but may be needed

Topic 05: Heroku CLI (example commands: run, cert, logs)

Not so familiar, maybe used few times

Topic 06: Deployments

Familiar, doing repeatedly

Topic 07: Access control

Not familiar, not used

Topic 08: Settings management

Familiar, at least configuration variables

> Heroku’s general properties

Topic 09: Ease of use

Very easy deployment and rollback. These are the main features in my use.

Topic 10: Stacks & Buildpacks

Not familiar.

Topic 11: Availability

Familiar. Quite good availability

Topic 12: Troubleshooting

Somewhat familiar, not so easy.

Topic 13: SSL endpoints

Not familiar.

Topic 14: Custom domain names

Not familiar.

Topic 15: Maintenance mode

Not familiar, recommended.

> Heroku’s add-on and integration system

Topic 16: Automatic deployments

Familiar (if this means MOT+ circle usage), very useful, but not so good.

Topic 17: Add-ons management

Not familiar.

> Kielikone’s adoption of PaaS

Topic 18: Evaluation process

-

Topic 19: Debuggability

Familiar at least trace feature. Bad, current trace is not properly implemented in applications/services.

Topic 20: Security

Not possible to consider security.

Topic 21: Co-operation

Ok

Topic 22: Training

Heroku features were introduced at start?

Topic 23: Automation

Very good at deployment.

Topic 24: Continuous integration

Very good.

Topic 25: Continuous deployment

-

Topic 26: Implications to system architecture

Heroku security limits system security acrhitecture?

> Wrap up

Heroku is good at development. Possible not so good in production after number of users is large.