Haaga-Helia
University of Applied Sciences

Three major cloud providers and their applicability for hosting a chatbot

Jarmo Hakkarainen

Bachelor's / Master's Thesis
Degree Programme in Business Information Technology
2017

Haaga-Helia
University of Applied Sciences

Abstract

8. August 2017

Author(s) Jarmo Hakkarainen	
Degree programme **Bachelor's Degree in Business Information Technology**	
Report/thesis title Three major cloud providers and their applicability for hosting a chatbot	**Number of pages and appendix pages** **22+3**

Up until this project, Liquid Studio has mainly used AWS. To avoid tying everything to one platform branching out was required. To test out the capability and viability of both Microsoft Azure Cloud and Google Cloud Platform a chatbot would be deployed on each platform to map out the process and requirements. The chatbot was chosen due to its dependency on external APIs, thus demonstrating a multi-cloud solution. A chatbot is in essence a software capable of having an automated conversation with the user.

The key metric used was the need for refactoring, the less the better with an added consideration to time spent on the deployment and configuration. The end goal was to have the chatbot running on all platforms.

The plan was to keep the code base the same, and only modify the scripts required for deployment. This was achieved sufficiently since only a few scripts were modified and a few files added. Resources were chosen according to documentation available on each platform and best practices in the field.

The project was timed for three months. Tooling used in the thesis was the same development setup that is generally used within Liquid Studio. This setup includes Visual Studio Code as the code editor and a MacBook Pro as a workstation. The default terminal was used for console access and related work.

Keywords
Cloud, chatbot, Azure, AWS, Google Cloud

Table of contents

1 Introduction

This thesis has been commissioned by the Helsinki Liquid Studio which is a part of Accenture. Liquid Studio is an Accenture initiative centred around rapid prototyping and development with new disruptive technologies and methodologies such as artificial intelligence (Accenture Technology 2017). The chatbot which is the centre piece for this thesis is based on the Microsoft Botframework SDK. Most of the chatbot's development and concept has been done by Jarkko Ylipaavalniemi, PhD.

This thesis is set to compare three major cloud platforms and their applicability for hosting a chatbot. These are the Microsoft owned and developed Azure cloud, Amazon Web Services (AWS) and Google Cloud Platform (GCP). As far as the chatbot itself is concerned, the technologies used are agnostic and should be compatible with any modern cloud platform. As an added benefit this thesis will also help to better map out the capabilities and suitability of each platform for the AI team and Liquid Studio in general.

For the sake of scope, no major adjustments will be made to the existing code base, beyond the needed deployment scripts and some other minor changes to other project related scripting, such as NPM-related scripts which manage dependencies and installation of them and the build-flow.

2 Core concepts explained

The asset that is the centre piece for this thesis is a chatbot named Kiana. Chatbots and conversational bots are the current trend in tech and user interface design (Laurinavicius 2016). A lot of the time chatbots are used to enhance the user experience in team chats and on services like Yammer, and to supplement a company's customer service by giving every customer a chance to talk about their needs rather than just complete their business via a form.

2.1 Two main types of chatbots

In general, there are two kinds of chatbots in use at the moment, a rule-based chatbot and an artificially intelligent one. A chatbot in itself is defined as "A computer program designed to simulate conversation with human users, especially over the Internet." (Oxford living dictionaries 2017).

A rule-based chatbot only reacts to what it has been coded to react and respond to, an AI (Artificial Intelligence) learns and adapts accordingly to what it experiences. When creating a chatbot, the developer instructs it what to answer to which kinds of queries and how to do it, and if it runs into a prompt that it isn't taught to handle, it will not and will usually crash instead, or simply display a default error message. Instead of crashing, an AI would learn from this failed interaction and next time, or after a few similar failures, learn to deal with it and improve itself.

An AI is usually taught in two ways, supervised and unsupervised (Marr 2017). The most efficient, yet least reliable, way is completely self-learning mode, where the AI/software does it's best to figure out how to react to any which interaction. This has led to some questionable interactions between users and the AI itself, due to malicious intent on behalf of previous users interacting with, and thus teaching, the AI. Most notorious such implementation of a self-learning model was Microsoft's failed Twitter-bot "Tay" (Perez 2016), where Tay picked up racist and sexist rhetoric from users in less than a day. Instead the standard practice is to teach under supervision, so as to monitor what the software believes and to make sure that it does not pick up any racist or malicious undertones from deviating interactions.

The last distinction to be made between a chatbot and a regular bot, as they appear to the masses, is their functionality. A regular bot can usually handle a single task or set of tasks such as retweeting posts with a certain hashtag or executing a set of instructions based on input. These kinds of bots are simply used to automate repetitive tasks that do not require human intervention and can thus be left to run unmonitored.

2.2 The Cloud

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. (National Institute of Technology, 2011.)

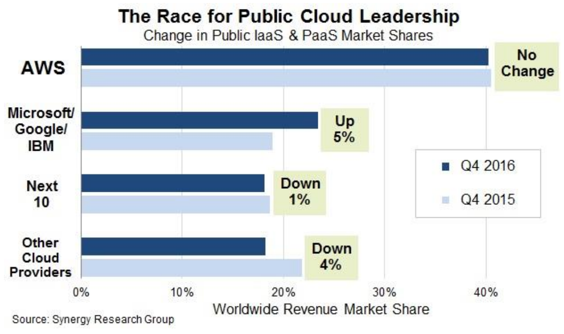
Cloud as a product and a service has developed a market where both consumers and companies alike are willing to give their data to someone else, in exchange of not having to micromanage their hardware and storage, and externalize the related costs to someone else in exchange for a relatively cheap price since the cloud works on a “pay for what you use” principle. An added benefit is also the ability for seamless scalability that comes from using the cloud. During peak usage, you can simply pay for a bigger allocation of resources to account for the extra load and let it scale down once the peak has died down, all of which can be automated.

Since virtually everything is being moved to the cloud, it makes sense to develop straight for it, which is where the topic for this thesis came from. In addition to providing storage space on the cheap, the cloud providers handled here offer computational services and power, such as Google Analytics or Azure data lake. When you couple the nigh-on unlimited storage space with easily scalable computational resources, it paints a clear image as to why companies are moving to the cloud en masse. Infrastructure as a Service (IaaS) has thus become the go-to method of doing business, since you do not have to spend huge amounts to securing the hardware for both the necessary storage allocation and computational power, but instead get it straight from an outside source.

The cloud in general has been embraced by big companies and even governments and states wanting to reduce their overhead that comes from huge server rooms and gear (IBM 2015). Updating, repairing and upgrading these servers has been extremely costly up until now, with the added burden of licence fees if the company used Microsoft Server or other such commercial solution. In case a company used a proprietary Linux-based server then the upkeep of that and the cost of personnel made it cumbersome. Smaller companies see the same benefits, with the added benefit of never having to have invested in their own servers to begin with.

The three platforms were chosen much due to their popularity within the market. AWS is the biggest and Google Cloud and Microsoft Azure are trailing behind it (Figure 1). Since these are the three biggest platforms available, it makes sense to see what they are capable of. All of their resources follow pretty much the same pricing scheme, and their resource catalogues in general look a lot alike. There are differences within the services and mainly their pricing policies. Key differences include Google Cloud Platform (GCP) charging by the minute whereas the other two (Azure and AWS respectively) by the hour.

Figure 1. Cloud market shares in Q4 2016, Synergy Research Group 2017



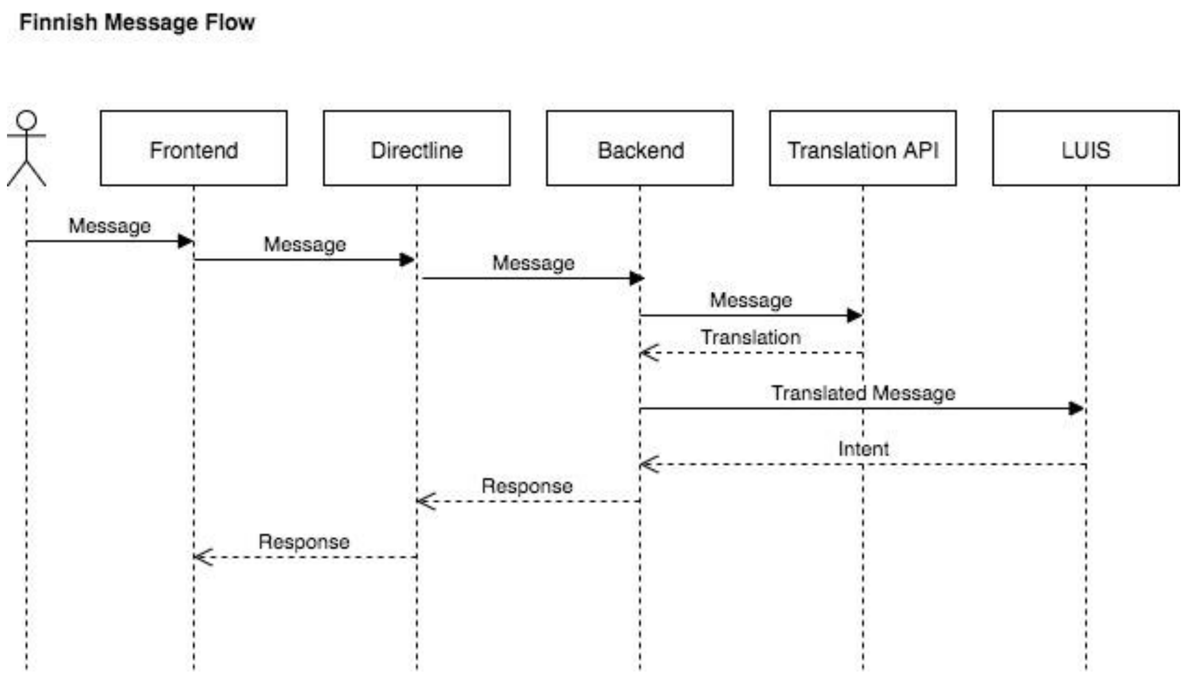
2.3 Technologies used

The product itself, chatbot Kiana, has been developed in NodeJS. The main concept behind Kiana was to develop a multilingual chatbot that can handle more than just English as an input language (Ylipaavalniemi 2017). This is also consistent with the notion that at Liquid Studio, only the newest and best technologies are used for development. Since the chatbot is based on the Microsoft Botframework SDK it also bases some of its technology to Typescript, which is considered a superset of JavaScript. The chatbot uses Microsoft Directline as a way to secure the connection between the frontend and the backend. Directline is a Microsoft provided Software as a Service solution which comes with the Botframework that Kiana is based on.

The base code for the frontend came from Microsoft's Botframework SDK, which is publicly available under an MIT license. The Botframework SDK is a part of a bigger trend of companies wanting developers to use their development kits to develop bots and other apps. Microsoft's Botframework essentially teaches the developers how to use their services and thus makes it easy and intuitive for the development to continue using Microsoft's services, thus tying the project to Azure and other Microsoft based services. The botframework hasn't been used as it is, but instead it has been heavily modified to suit Liquid Studio's purposes and to fit the vision for the concept.

2.3.1 Software architecture

The chatbot itself has been developed in NodeJS with the basis in the Botframework SDK. A bot works on intentions, which are in plain terms actions that the bot can perform based on input. A basic example of an intention is a greeting, when the bot receives a greeting from a user it replies with its own greeting. The backend handles all the logic that comes with a chatbot and connects to the external APIs (Picture 1).

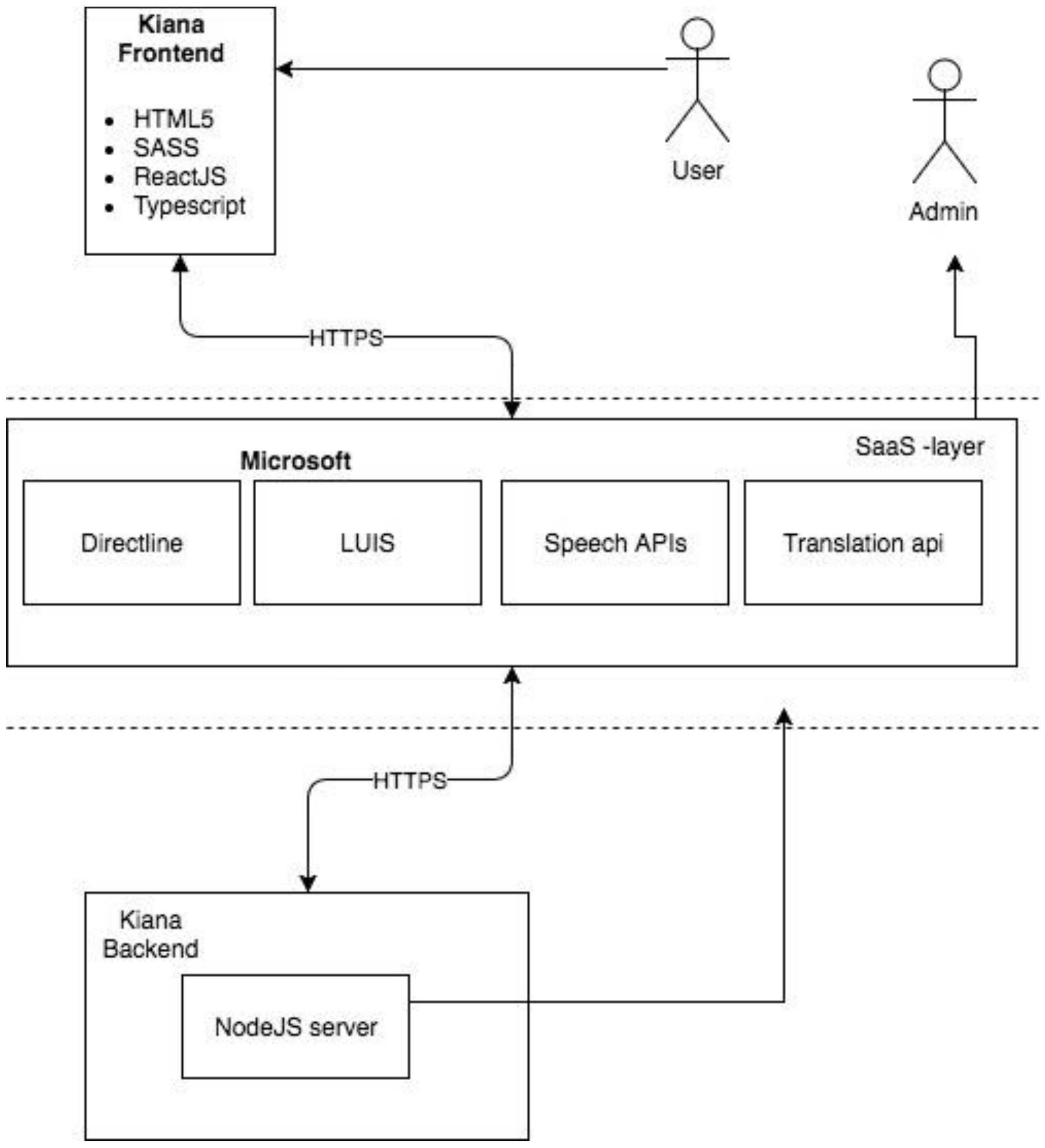


Picture 1. Kiana message flow diagram

The frontend has been written in HTML5, JavaScript, Typescript, ReactJS and Sass with CSS3. The Typescript used in the project is what has caused some of the problems faced in the deployment to Azure, since Webpack and Typescript are required for compilation during the deployment process. For most cloud platforms, this is not an issue but some of the local configurations in the project did not replicate properly to the cloud upon deployment. This was much due to project Kudu's inner workings. Otherwise the software requires nothing special from the chosen cloud platform, only that it supports modern JavaScript.

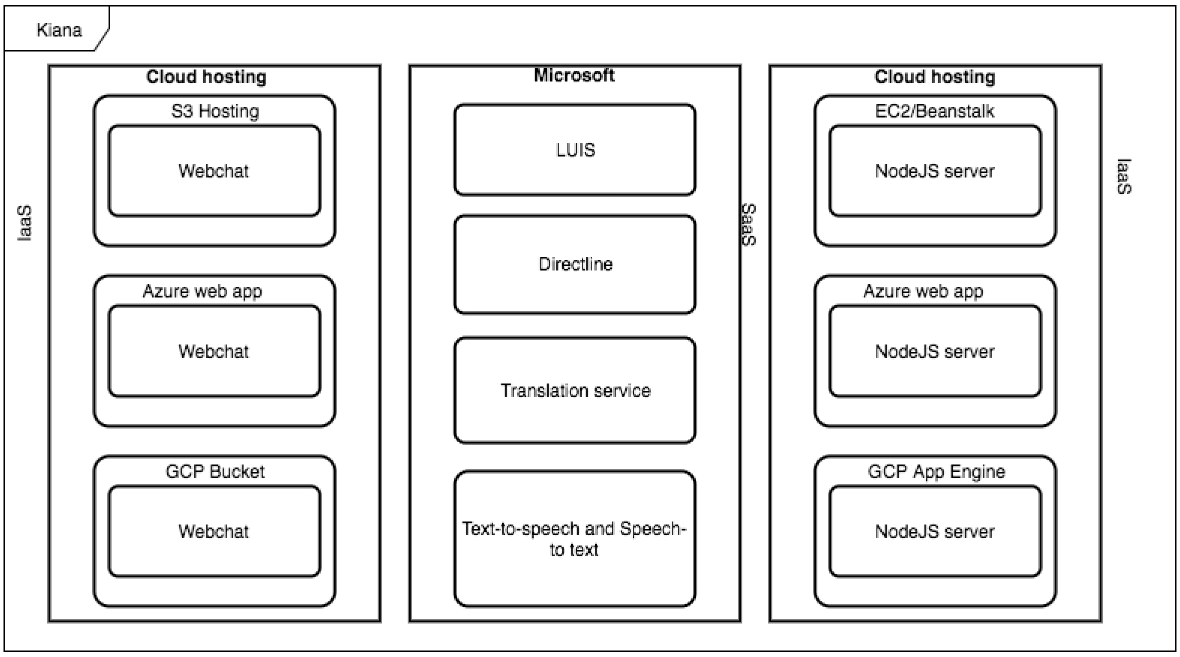
2.3.2 Third party services

In addition to internal logic and functionality Kiana uses third party web services, which are based on Azure (Picture 2). These services include LUIS (Language Understanding Intelligent Service), Bing Speech APIs and Microsoft Translation service. Since Kiana is based on the Botframework SDK the support for these services came essentially out of the box, or at least with sufficient support that it was easy enough to enable them. The most important service here is LUIS which essentially enables the functionality of a bot like this. LUIS essentially lets the user use normal language in their interactions with the bot, instead of them having to carefully pick their words for the bot to understand them properly.



Picture 2. Kiana architecture

Kiana’s architecture consists of three layers, two Infrastructure as a service (IaaS) layers and a Software as a Service (SaaS) layer. The main difference between these two layers is that SaaS is provided as-is and ready-to-use, whereas IaaS requires user managed installations and software. Both the frontend and the NodeJS backend are hosted on each cloud provider’s IaaS layer and connect to Microsoft’s SaaS where LUIS and the other API’s reside (Picture 3).

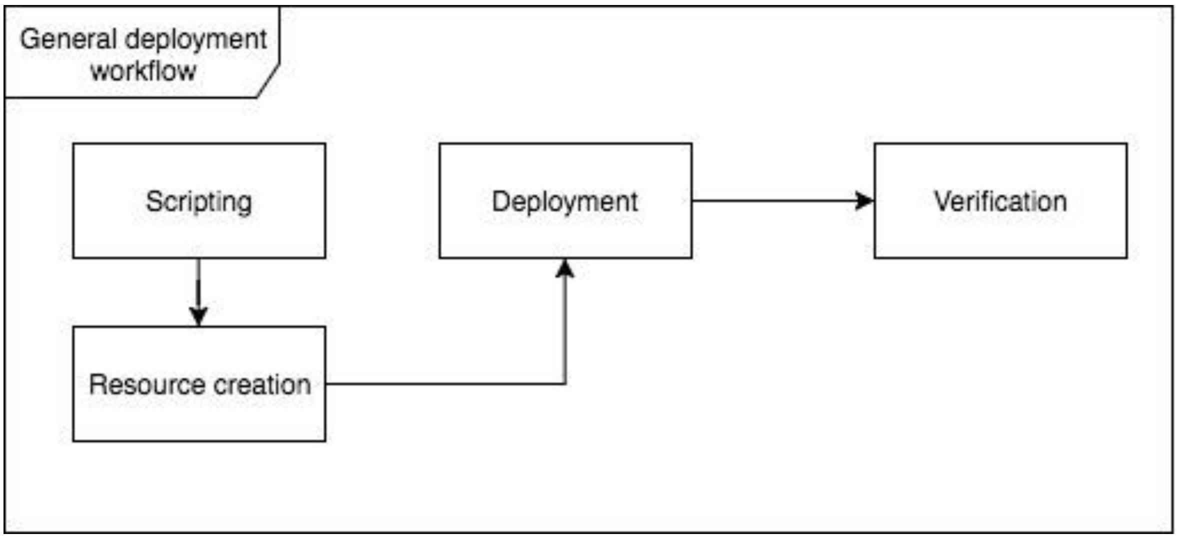


Picture 3. General architecture

Since a lot of external services are required for functionality, consideration over them will be given in this thesis. Each of the three platforms offers something to the table respectively and this has been taken into account within the consideration for each platform. The aforementioned resources in Artificial Intelligence and Machine Learning services are usually only available in US-East of each service provider’s resourcing, much due to the services being actively in development and them being the easiest to manage for the development if they are not widely available.

3 Deployment process

The deployment process was kept as consistent as possible between the different platforms, with no attempts at creating a CI-pipeline (continuous-integration) for any of the platforms. Due to the company's existing technology stack and experience, at least the AWS-deployment process will be easy to replicate with a Cloud Formation Stack or another already established method. The general workflow (Picture 4) is consistent across platforms, starting with the creation of necessary scripts and resources and then command line based deployment.



Picture 4. General deployment workflow

The resource choices for each of the platforms were made so that they represent the easiest and the most cost-efficient way of achieving the end goal. All of the resources are comparable between each other (appendix 1), since they represent a recommended way to deploy a node application to the respective platform.

3.1 Azure cloud

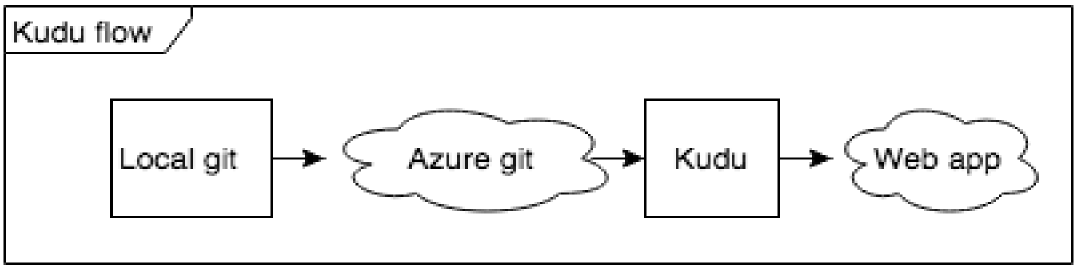
Azure cloud is operated by Microsoft. Azure cloud promises top of the line stability and availability, with an SLA of up to 99.9% depending on the service (Butler 2017). Azure has the backing of Microsoft's extensive hardware and software capabilities and thus is a platform supportive of all the most common development tools and technologies.

3.1.1 Azure and resources

As far as chosen resources and resource types go, I decided to go with standard web apps coupled with the required server farm. Resource sizes were kept decidedly small, at the standard tier, so as to minimize unnecessary costs for the project. Most of the resources pertinent to the chatbot are Microsoft based technologies, such as LUIS, Bing Text-to-speech and Speech-to-text interfaces, same as the Directline security interface.

3.1.2 Project Kudu

A local repository inside Azure was used for the deployments since Liquid Studio uses GitLab for versioning and at the time of deployment integration with GitLab was not directly supported. The internal repository is powered by Project Kudu which is a versioning control system that is a part of the underlying architecture in Azure. By pushing to this repository as a remote repository from a local machine, Project Kudu will then build and deploy the app independently and automatically (Picture 5).

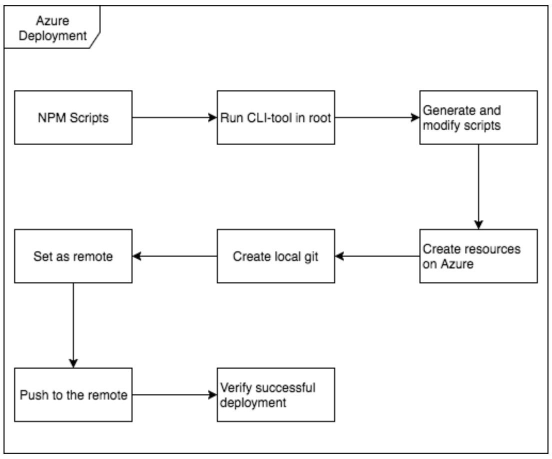


Picture 5. Project kudu general flow

3.1.3 Deployment to Azure

The deployment process in itself requires two additional scripts to be generated with the Azure Cli –tool, “deploy.sh” and “.deployment” respectively. The deployment -file is a basic bash script file and required some additional modifications to accommodate our chosen technology stack, which includes TypeScript, Webpack and Sass. After a modification the project’s package.json –file and build-script therein, the project was now ready for deployment. The build script needed modification because project Kudu cannot run multipart commands during the deployment process flow (Harms, Joshua 2017).

As a complete process, that is creating necessary resources and then deploying an application to Azure (Picture 6), the process took two complete workdays (16h) in the end and as such can be considered an inefficient process. Now that the guideline and guide for deployment and resource creation has been established, the deployment process should take no more than a business day (8h) in the future, and if the project contains the same technology stack as here, then the deployment should be replicable in a matter of an hour or two.



Picture 6. Azure deployment

Even though it was not the recommended way the frontend was also deployed unto Azure Blob Storage. This process was a bit more complicated than the others but as a whole a lot easier than the recommended web apps way. This method definitely has it's uses but I would refrain from using it in most cases since it is far from optimised, especially in production use where proper SSL-configurations would be required.

For future reference when using Azure, I would refrain from using Sass or Less and instead stick to plain CSS3. Sass and Less require compilation, which is usually done during build in conjunction with deployment. This in turn creates problems when using project Kudu for deployment handling, since Webpack is required for compilation with Sass etc. Errors produced by Kudu from compilation during build-time are in turn a bit unfortunate, since using Webpack has become standard when dealing with ReactJS (Ray 2016). The mismatch caused by standard practice and Kudu's workflow restrictions can make the deployment feel arduous and complicated if refactoring is required.

3.2 AWS

AWS is the development platform by Amazon. As far as cloud platforms for hosting apps are considered, AWS is the most popular one out there, with a market share of up to 40% globally (at the time of writing), with Microsoft, Google and IBM trailing behind having equal shares of the market (Synergy Research Group 2017).

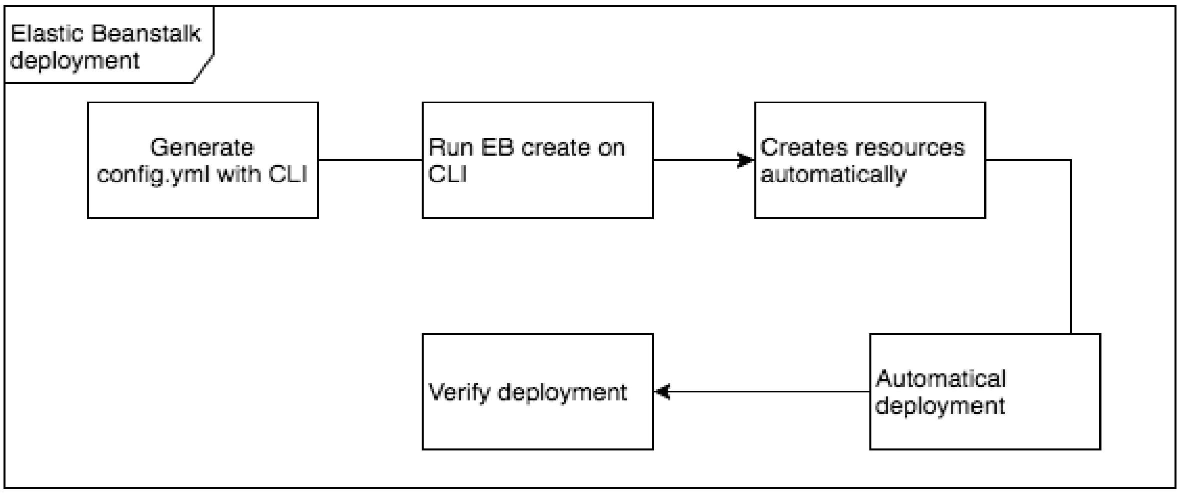
3.2.1 AWS and resources

As for the AWS resources for the project, I opted to use Elastic Beanstalk as the deployment service for the backend, since it works a lot like Azure's web apps. You simply define the language you use (NodeJS in this case) and create the resources. All of this is available in the EBS Cli tool that was used for the deployment. For the rest of the thesis any management that was necessary with Beanstalk was done through the web portal.

The frontend of the project was deployed to Simple Storage Service, or S3, since it is the easiest way of hosting a static website on AWS and directly comparable to that of GCP or Azure's storage. I briefly tried to achieve this goal with EBS too, but it did not pan out as expected and further inspection into documentation stated that S3 is the more optimal way to host a static website in general.

3.2.2 Deployment to AWS

Altogether the process took less than three hours of work and required no modifications to scripts or code base. This can be considered a frame of reference, that the deployment should be a relatively quick and effortless process when done correctly. The deployment was done to two different services, S3 and EBS respectively. As a process, deploying with EBS (Picture 7) was a straightforward process and thus leaves very little to discuss about.


Elastic Beanstalk deployment
Generate config.yml with CLI
Run EB create on CLI
Creates resources automatically
Automatical deployment
Verify deployment

Picture 7. AWS Deployment

EBS creates the needed resources and their configurations automatically upon deployment, which makes for an easy deployment process altogether. EBS deployment requires two files containing the relevant scripts, but these are done automatically by the CLI and no modifications were required there either. No changes were required to the actual codebase and it was left as it was after the Azure deployment.

The S3 bucket that hosts the frontend had its access rights configured to allow public access and static website hosting, available from the options menu. This as a process is well standardized across the three platforms used in the thesis, and as such extremely easy to replicate on the other platforms. For ease of resource management, EBS might be a viable option for deployment here also, but since the resource does not require much of management as it is I opted out of this.

3.3 GCP

Currently one of the big three cloud platform providers, Google has extensive knowledge of handling big data and analytics (Harvey, Cynthia 2017). Much of this knowledge is due to their renown search engine. Google Cloud Platform is the newest of the three handled in this thesis (Laurinavicius 2017), and as such is still gaining its foothold in the industry, but with Google's reputation in data analysis and handling, it is a serious contender for both AWS and Azure.

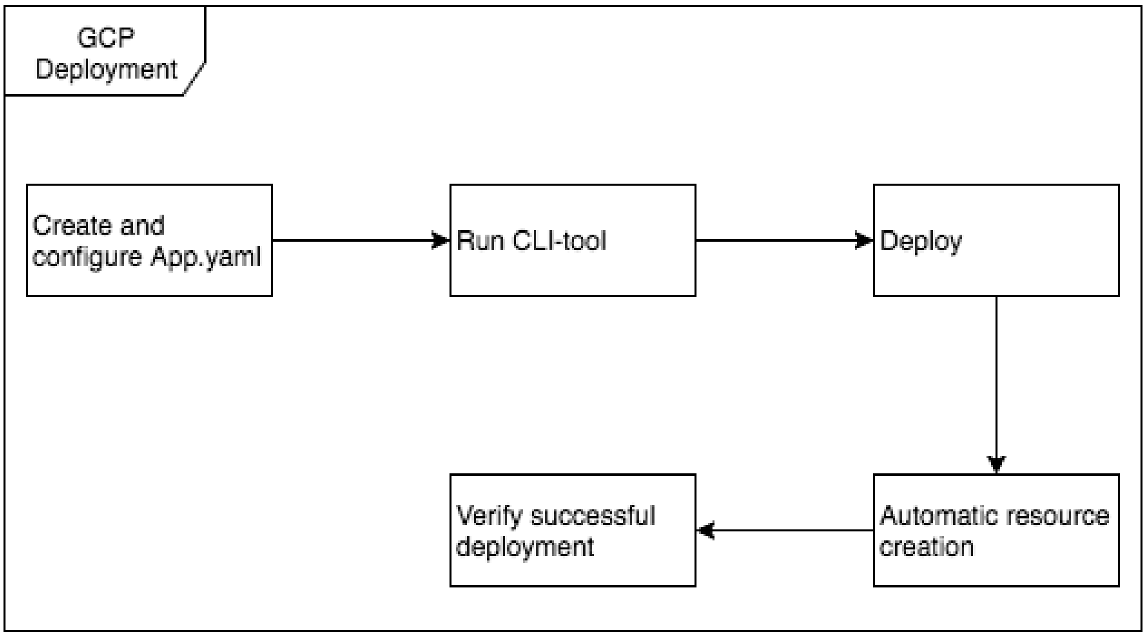
3.3.1 GCP and Resources

As far as the resource catalogue on GCP goes, the app engine is the most versatile and also the most applicable for this use. GCP's App engine has been well designed to resemble the competitors where it matters. The Cli-tool for the service works as expected and was simple to use for the deployment and management of the resources. Only thing going against the app engine is the actual console view in browser, which's resource naming convention is hard to follow when changing into from a different provider's platform.

3.3.2 Deployment to GCP

Due to our team not having used GCP before this assignment, the process to setup the environment on Google's end took a bit longer than expected. Much of this was due to corporate bureaucracy, and the fact that we did not have any experience on GCP nor did we have the environment fully setup unlike with the other two providers. Once everything was sorted out, the actual deployment was relatively effortless.

Google has a stake in the JavaScript and NodeJS environments so the deployment for the backend went mostly without a hitch, since the technologies are well supported and GCP resources are well optimized for these technologies. The code base was still in the condition left by the Azure deployment, so some refactoring in package.json and the build – script therein was required in order to comply with how GCP's deployment works. This represents the industry standard better than that of Azure's. After modifying the start script for the node-server the deployment (Picture 8) went through without a hitch.


GCP Deployment
Create and configure App.yaml
Run CLI-tool
Deploy
Automatic resource creation
Verify successful deployment

Picture 8. GCP Deployment

The frontend was deployed to a Google Cloud Bucket to be comparable to AWS and since it is the recommended way inside Google Cloud to host a static website (such as plain HTML5 or ReactJS). According to the tutorial on GCP documentation on how to host a static website on the app engine, the process is exactly the same as with the backend. Due to the process being exactly the same and requiring the same steps and effort, I did not deem it necessary to recreate the deployment.

As a reference, if the environment is already up and running the whole process should take no more than a business day, which also includes completing the deployment tutorial provided by Google, which covers the very basics of the platform. Without the tutorial, the process should be done in less than two hours. This naturally is highly dependent on the resources used, but at least both the app engine deployment and usage have been sufficiently optimised.

4 Discussion

With the deployment process complete, it is time to think which platform makes the most sense for a chatbot, and by conjunction development use for Liquid Studio Helsinki, especially for deploying a prototype or a demo. I will also give consideration as to which platform would be the one that we would recommend to a customer for something like a chatbot, or a comparable product.

As a client-driven business client's wishes will naturally be respected, but if a platform makes more sense in regard to the project without a doubt, it would be beneficial for all parts to give consideration for the other platform. Towards this end, after this thesis Liquid Studio will have a better point of reference to give to a customer to support their decision, whether it is to continue with their original platform or to adopt a new one.

4.1 Azure and AWS

When you consider the purely technical part of the deployment process it's arguable that AWS makes more sense, especially if you use a highly optimized service like EBS for the actual deployment of your app. Since EBS provides such seamless deployment for a web app, it is a clear winner in that regard, at least when compared to a service such as Azure's Web Apps.

What Azure benefits the most is their infrastructure in general and the backing of the Microsoft technology stack. This technology stack especially prevalent inside Finland, where most big companies are already rather tied to Microsoft, due to both Azure and then the most common OS platform currently, Windows [10]. In addition to this, Azure does make the resource creation rather easy when using their interface and since the rest of the technologies used in the chatbot are Microsoft based and provided, it does make sense in the general usability and functionality of the app to use Azure.

I would still continue to deploy Kiana and other projects like it to Azure, since the web apps seem well suited for the purpose. On a general level, my recommendation falls upon AWS since they have a more wholesome service to provide and actually setting up the environment, and getting the deployment to run through with Liquid Studio's current development setup is a lot more streamlined. It is also worth noting, that no changes are required in the current setup to create a fully automated deployment pipeline when using AWS, which would also allow the environment and setup to remain fully compliant with Accenture's own policies and practices. A noteworthy caveat here, is that a lot of the times a customer has a different setup completely from that of Liquid Studio's, in which case things like CI-pipelines are not a problem and in that case Azure is definitely a viable option.

4.2 Azure and Google Cloud

The comparison between GCP and Azure is a fair one since they hold about the same position in regard to both AWS, and the cloud provider market in general, which is a combined 20% of the market with IBM included vs. the 40% that is estimated for AWS (Synergy Research Group, 2017). When you consider this, it makes sense to compare Azure and Google to see which one would make more sense to invest more time into.

Both of the services offer many of the same functionalities and capabilities, with focuses on different key aspects. Azure shines in their support for hybrid cloud solutions, where a local or a private cloud is coupled with a public one. Some of the more enterprise oriented configurations, such as compliance policy configurations are advertised as strongpoints for Azure.

4.3 AWS and Google Cloud

As far as resources provided and available on the platform, both AWS and GCP are much alike. GCP's app engine is familiar to use for someone who comes from an AWS background and vice versa. There are extensive cli-tools for both platforms and services with which both creation and management of resources is done through. Alternatively, the user can opt to use the console in a browser, though deployment is best done via the terminal/cli.

When comparing the actual deployment processes, it is a lot easier to compare that of GCP's to AWS' since their processes are rather similar. To initiate the process, you would use a CLI-tool on both services on your local machine, which then in turn initialises the project directory for the appropriate deployment. Since the deployment works so similarly on each platform the comparison is easy to make but the following recommendation is more difficult. Nonetheless I would still choose AWS over GCP, since I personally found AWS easier to manage when compared to GCP. This is the most apparent when it comes to IAM (Identity Access Management) inside the platform. Google's IAM was only recently implemented and as such is still cumbersome to use and definitely requires some fine tuning on Google's end. Managing related resources such as SSL-certificates and billing is also a lot easier on Amazon, where it can be done through Route 53 whereas GCP's equivalent was difficult to find and even more difficult to access, due to GCP's IAM configurations and the challenges therein.

AWS is recommended due to the simplicity of its pricing and the wholesomeness of the service. It is worth noting here though, that GCP is relatively new to the market, and a lot of its features are either in beta or carry a disclaimer, that the service is expected to change and it might break your product/service. AWS on the other hand is considered extremely stable and as such is better suited for production. The rest of the platform is generally extremely stable and changes that are implemented within there are applied over a long period of time, giving developers ample time for implementing and refactoring the necessary changes.

For fairness' sake, I could recommend Google to a customer or to an internal project that is all about scalability on the cheap, since GCP has extensive options for different configurations and sizes for the resources. Google also has extensive tooling for handling big data and otherwise analysing it, so in some cases a mixture of AWS and GCP could be the perfect answer.

4.4 Cross-comparison of all three

When all three are considered a clear winner is hard to find, since all of the three platforms offer a lot to the table. If a winner must be picked, for chatbot deployments I would go with Azure. Azure's web apps are extremely simple to use once the deployment process is setup properly and they offer the best compatibility and usability when you consider the price. Azure also offers many of the necessary services to pair up with the chatbot, such as LUIS.

In the case of Liquid Studio in general my recommendation falls on AWS due to the wholesome service it offers and the ease of use of all of the resources. During the deployments it became obvious, that AWS has the most extensive documentation about both the basic workflows of their services, and also the more complicated concepts such as CloudFormation. This is further helped by Liquid Studio's already existing AWS capabilities and know-how.

All of the three platforms use a form of Identity Access Manager (IAM) for internal security, and it is considered a best practice to utilize these as much as possible. Out of the three AWS has the most robust and extensive one and GCP has the worst. GCP's IAM and its difficulty is much due to its relatively young age as a service, having only come to the market quite recently (Butler 2017).

4.4.1 Why Azure?

When developing on Windows or for a Microsoft-bound customer it is arguable that choosing Microsoft Azure makes sense, especially if the company has an added interest in a hybrid cloud solution, where data is divided between a private cloud/local server and a public/external server. This coupled with the general trust that especially Finnish customers have in Microsoft as a partner and a provider, the sales opportunities might be easier.

The actual development setup would require some effort, but in the end, it is still possible to establish a proper development pipeline for Azure development. This coupled with Microsoft’s extensive and easy-to-couple AI and ML –services I would recommend Azure for the AI-development team and field of Liquid Studio and by extension Accenture. This is also further supported with the fact that, in comparison to the biggest competitors’ offerings, all of the AI-tools are well in their development and easier to expand upon and customize to your project’s needs.

What became apparent during this exercise, is that Azure is the least optimized for hosting a standard ReactJS –based webpage, for it had a tenuous deployment process with many a hiccup along the way, such as project Kudu not handling multipart NPM-commands. During the research for this thesis I discovered that I could also host the frontend on Azure Blob Storage. I deployed the website unto blob storage too to justify straight comparison of all services, but will still focus on web apps due to them being the recommended way inside of Azure for this task. It is also worth noting that the information about Azures Blob Storage hosting is extremely difficult to find and as such I do not personally recommend it either.

My recommendation for the web apps for chatbot deployment actually comes from the default provided SSL-certified endpoint for the resource. Azure provides the web app resource with an “azurewebsites” domain which has a Microsoft owned and provided certificate. The SSL-certificate needs to be considered because Directline requires all of the traffic between front- and backend to be encrypted sufficiently.

For Kiana specifically, I would choose to continue using Azure if no pressing reason enforces a change. With the software architecture and Azure services, and now the ready-to-use deployment scripts, the software is simply just better optimized for Azure versus the competition.

4.4.2 Why AWS?

AWS has strived to streamline a lot of the processes, such as deployment and management of a project through Elastic Beanstalk which is nigh on effortless. Using EBS makes especially sense in internal projects and small MVP or PoC-types of projects, such as our chatbot Kiana due to the ease of the actual deployment, so more time can be dedicated for the development instead. AWS also provides training by them for the customer at the customer’s location so starting development on the platform is very well supported.

As far as chatbot hosting is considered, it can be argued that actually creating and hosting your own chatbot unto AWS is redundant since AWS has extensive internal services and tooling for such. AWS provides a service, Lex coupled with Polly and their own machine learning platform, to create and host chatbots, with the provided AWS generated mobile app. With these tools in mind it does not make a lot of sense to actually develop an external chatbot design solely AWS hosting in mind, but rather develop independently and assess when the time comes whether or not hosting on AWS makes sense.

AWS is also considerably easy to sell as a solution, due to AWS actually having a huge market share of up to 45% and more (Miller 2017) with very little to show any slowing down in growth. A lot of big companies already have some or many of their services and products running on the platform, so they have their own practices and policies already figured out. This, coupled with the safety of familiarity, makes it an optimal sales opportunity and in general an easy tool to use for chatbots and other web apps.

For hosting Kiana AWS brings to the table easy integration with the rest of Liquid Studio’s development setup. In addition to this, easy-to-use services and a wide user base on Stackoverflow and among other communities. Major flaw of the platform is, that the AI- and ML-tools are highly tied to the AWS-ecosystem and as such bring little benefit to Kiana’s development.

4.4.3 Why Google Cloud Platform?

One of the benefits that Google itself is driving its foray into the world of cloud PaaS is the scalability and simplicity of their basic instance’s pricing. Unlike the competition Google charges by the minute, whereas comparable competition charges by the starting hour. This makes it an ideal platform for a software that is expected to have a lot of light to medium and inconsistent traffic, such as perhaps a proof-of-concept demo or an internal demo or beta. Google also brings to the table expandable capabilities in the AI- and Chatbot-fields, but lacks the possibilities and opportunities for customization when compared to Azure. If the project allows to be tied to Google's platform in entirety then the Google AI-tooling is extremely capable, especially when it comes to translation and speech synthesis.

The frontend was hosted on GCP's bucket service due to its simplicity and comparability with others. The biggest advantage that GCP brings to the table beyond their more than capable pricing policy is Google's extensive support and knowledge of open source technologies and analytics. Even though Google's documentation of the service is still much under development, it has already matured better than that of Azure's, which is hard to find and decipher.

Google's App engine provides a default domain and SSL with the service, but as for the resource's URL naming it is not something that many customers would approve of. In case you do not purchase your own certificate, and set it up correctly, you are left with a combination of your project name, app name and then the appspot subdomain, which altogether does not make for an optimized or reasonable URL (Google 2017a).

Google has a well-documented CI-pipeline support, which makes GCP a valid option for a customer project (Google 2017b). Since a properly set up CI-pipeline is important and also widely considered a best practice this speaks well on behalf of GCP and should warrant further investigation in to the matter, especially since a proper integration with Azure has proven troublesome so far within our internal projects and assignments.

In general, I could with good conscience recommend GCP for hosting Kiana. The deployment is extremely painless and management in general seems straightforward enough. GCP also brings to the table Google's voice and speech API's which have proven troublesome up until now, but would be beneficial when using the platform, and more than likely easier to take into use.

5 Introspection

Researching the policies and workings behind the biggest cloud providers gave me an interesting insight into the whole topic and really helped me to understand the magnitude of the market and paradigm shift. There is a lot of buzz around “The Cloud”, but actually researching it and comparing options gave me a whole new look at the topic and understanding of its complexity. I among others have used cloud based solutions for a long time now without actually knowing what’s happening behind the scenes.

With the actual practical part of the thesis, I came to realise that during my employment here at Liquid Studio I’ve learned quite a lot about scripting and different cloud platforms, mainly about AWS and Azure, with the added learning on GCP during this project. As for what I could definitely improve upon, plan the project even better and have a concrete step-by-step understanding of what is required for each part, both with the unexpected such as how complicated the starting process for GCP was and how I overestimated how much work the AWS deployment would be.

When doing the actual empirical part of the thesis it became apparent to me, that the scope could have been a bit wider and I would have still been able to achieve the end goal within the given time for the project. As it is I way overestimated how much time each deployment would take and what would be needed for them, since I did not have much experience in the matters when actually planning the thesis and the project as a whole. That said, I am still content with how I achieved my goals and achieving them ahead of time gave me the opportunity to properly focus on writing out the thesis and researching the differences between the given platforms.

Much to my personal surprise, as an end result I decided to side with Azure for the deployments. Coming in to the thesis project I expected to find AWS the better choice for this, much due to my earlier experiences working with both platforms. Nonetheless, I am pleased to be proven otherwise and will let it stand as a reminder for myself that choosing a technology stack requires many factors to be considered in.

6 Conclusions

In the end, even with all the flaws laid out in the open, I would recommend Azure for hosting Kiana (appendix 3). I base my recommendation on both the clear structure of pricing on the web apps service, and the fact that most of the AI-related services in the chatbot are still Microsoft and Azure based. Keeping the ecosystem fully on Azure makes it easier to manage the stack. Azure is relatively expensive when compared to AWS which does give consideration against using Azure (appendix 1) but especially for Kiana and the intended use the actual usage related costs are negligible on all services.

If you broaden the scope beyond the chatbot asset it gets more complicated, but in the end, I would mostly recommend AWS. AWS has a wide array of applicable services to offer to different kinds of applications and the technologies they support are wide and varied. AWS' pricing is also extremely competitive and the resources and documentation available are exhaustive to say the least. Due to its popularity AWS also has a massive information source in communities like Stackoverflow and other technology forums, which makes development and production on the platform a lot easier than say with Azure for which its rather difficult to find custom solutions to a developer's problems.

Out of the three I would recommend Google Cloud Platform the least, since it is still heavily under development in the most important and interesting parts when it comes to innovative and new technologies, such as their AI-tools. This fact coupled with the reality, that most of Google's AI-tools are really closed off and only available if you commit to the ecosystem, makes it a really hard platform to recommend over Azure or AWS. Google's machine learning platform has a lot of potential, and is considered rather robust, since Google has extensive experience in big data and predictions.

What Google does have going for it is the pricing policy, so if the cost of the resource is a big talking point then it can be considered, since the app engine service is charged by the minute instead of by the hour. The price point is particularly interesting when you consider how a chatbot works fundamentally, with small bits of traffic coming in at a time. GCP would most likely be a huge asset and a viable option later on, a year or two from now, once the platform has matured more and both the users and the provider have figured out how to best benefit from and use the platform.

Here is also worth noting, that many of the problems that a developer faces in a project are easiest to solve by choosing an amalgamation of two or in some cases even more platforms. A viable example would be to host the actual application backend on AWS, use Azure-based AI-tools such as LUIS and Azure data lake or GCP’s big data analytics tools. With a chatbot based on the Microsoft botframework SDK it is arguable that for latency and user experience reasons hosting on Azure makes the most sense since the Directline service used in between the frontend and the backend is hosted on Microsoft’s servers, effectively reducing the ping and round trip of requests in extreme cases considerably.

References

Accenture Technologies 2017. Liquid Studios. URL: https://www.accenture.com/us-en/capability-rapid-application-development-studio# Accessed: 7 August 2017.

Butler Brandon 2017. AWS vs Azure vs Google: Cloud platforms compared: Page 3 of 3. URL: https://www.networksasia.net/article/aws-vs-azure-vs-google-cloud-platforms-compared.1488273117/page/0/2 Accessed: 11 July 2017.

Dotnetfoundation 2017. Project Kudu. URL: https://dotnetfoundation.org/kudu Accessed: 29 June 2017.

Google 2017a. Static Website Examples, Troubleshooting and Tips. URL: https://cloud.google.com/storage/docs/static-website#https Accessed: 8 August 2017.

Google 2017b. Continuous Delivery Tool Integrations. URL: https://cloud.google.com/container-registry/docs/continuous-delivery Accessed: 8 August 2017.

Harms Joshua 2017. Everything that went wrong when deploying my Node + Hapi app to Microsoft Azure. URL: https://nozzlegear.com/blog/everything-that-went-wrong-when-deploying-my-node-hapi-app-to-azure Accessed: 29 June 2017.

IBM 2015. United States of Cloud saves states millions of dollars. URL: https://www.ibm.com/blogs/cloud-computing/2015/04/the-united-states-of-cloud-saves-state-and-local-governments-millions-of-dollars/ Accessed: 8 August 2017.

Laurinavicius Tomas 2017. Can Google Challenge Microsoft And Amazon For Cloud Supremacy? URL: https://www.forbes.com/sites/tomaslaurinavicius/2017/04/10/google-cloud-challenge/#403a9adb5fea Accessed: 8 August 2017.

Laurinavicius Tomas 2016. UX Trends 2017: Experts Bet On AI, Chatbots And VR. URL: https://www.forbes.com/sites/tomaslaurinavicius/2016/12/04/ux-trends-2017/#2aac90f11615 Accessed: 2 August 2017.

Marr Bernard 2017. Supervised V Unsupervised Machine Learning – What's the Difference? URL: https://www.forbes.com/sites/bernardmarr/2017/03/16/supervised-v-unsupervised-machine-learning-whats-the-difference/2/#1e671a822080 Accessed: 7 August 2017.

Miller Ron 2017. AWS still owns the cloud. URL: https://techcrunch.com/2017/02/02/aws-still-owns-the-cloud/ Accessed: 5 July 2017.

Oxford Living Dictionaries 2017. Chatbot. URL: https://en.oxforddictionaries.com/definition/chatbot Accessed: 4 August 2017.

Panettieri Joe 2017. Cloud Market Share 2017: Amazon AWS, Microsoft Azure, IBM, Google. URL: https://www.channele2e.com/channel-partners/csps/cloud-market-share-2017-amazon-microsoft-ibm-google/ Accessed: 29 June 2017.

Perez Sarah 2016. Microsoft silences its new A.I. bot Tay, after Twitter users teach it racism [Updated]. URL: https://techcrunch.com/2016/03/24/microsoft-silences-its-new-a-i-bot-tay-after-twitter-users-teach-it-racism/ Accessed: 6 July 2017.

Ray Andrew 2016. Webpack: When to use and why. URL: http://blog.andrewray.me/webpack-when-to-use-and-why/ Accessed: 10 July 2017.

Synergy Research Group 2017. Microsoft, Google and IBM Public Cloud Surge is at Expense of Smaller Providers. URL: https://www.srgresearch.com/articles/microsoft-google-and-ibm-charge-public-cloud-expense-smaller-providers Accessed: 4 July 2017.

Ylipaavalniemi Jarkko 2017. Kiana Virtual Assistant Demo. Intranet. Accessed: 7 August 2017.

Appendices

Appendix 1. Compute resources used for the thesis

Compute resource	Specifications	Price	Notes
AWS	1 Core, 1GB, NaN	$9,50	Beanstalk managed
Azure	1 Core, 1.75GB, 10GB	$55,80	Shared between two apps
GCP	Flex-instance	$49.16	Backend, estimate

All estimates are calculated with the used resources and estimated full load for the application with one instance.

Appendix 2. Resources used for the thesis

Resource	Specifications	Notes
AWS EC2	1 Core, 1GB, NaN	Backend hosting, Beanstalk managed
Azure Web App	1 Core, 1.75GB, 10GB	Backend hosting, Shared between two apps
GCP App Engine	Flex-instance	Backend hosting, Flex instance
AWS S3	Bucket	Frontend hosting
Azure Blob Storage	Bucket	Frontend hosting
Google Cloud Stor-age	Bucket	Frontend hosting

Appendix 3. Scoring chart

Platform	Resource sizing	Pricing	Compatibility	Workflow	Additional services
AWS	1	1	3	2	1
Azure	2	2	1	3	3
GCP	3	3	2	1	2

Scoring is arbitrary on a scale of 1-3 to create distinction and clarity. Based on the experience of working with the services during the thesis and research for it.