Text Analytica: cloud-based document analysis

Florian Bauer
Department of Computer Science
University of Bristol
Bristol, United Kingdom
ya18048@bristol.ac.uk

Nathalie Pett
Department of Computer Science
University of Bristol
Bristol, United Kingdom
aq18034@bristol.ac.uk

***Abstract*—The source code is available at https://github.com/darkcookie298/CloudComputing. The application can be run online at http://textanalytica.lukaspman.io/.**

I. Introduction

Text Analytica is a cloud-based application which aims to support the analysis of text documents. For the purpose of this coursework a prototype has been developed and deployed using different cloud computing technologies. In the remainder of this Section I, the general concept behind Text Analytica is discussed as well as the limitations of the implemented prototype. The Section II of this report explores the reasons behind choosing *Microsoft's cloud services* over *Oracle's*. Afterwards in Section III the system architecture and technologies used for the project are explained in more detail. The scalability of the developed solution is addressed in Section IV, including evidence of the performance of the service under load. Lastly, in **??**, future improvements of the application are proposed.

A. Vision

University students are often faced with an abundance of resources regarding specific units or even certain topics within a unit. These range from lecture notes or slides to personal notes and additional scientific papers as well as e-books or extracts thereof. The first step in the exploration process is for students to familiarize themselves with these materials by identifying the documents key aspects and discovering links between different sources. This is what Text Analytica ultimately aims to facilitate.

More specifically the functionalities of Text Analytica could include tagging, keyword search, suggestions of related documents based on textual analysis and eventually the generation of short summaries, all based on user-supplied PDF documents. These functionalities render Text Analytica a useful tool for a large number of scenarios in which people are confronted with many different and possibly complex text sources, e.g. in the context of management decisions in industry or business / commerce.

B. Limitations of the submitted prototype

The focus of this coursework assignment was to deploy an application using different cloud services and explore its scalability. Therefore the functionality of the submitted Text Analytica prototype was stripped down to a minimum.

To skip the step of extracting machine readable text from PDFs by applying OCR techniques, currently users are only able to upload simple .txt documents. These files are then parsed and analyzed. At this point the analysis merely tags the system entries with the three most used nouns in the text. As the current state of analysis functionality does not require the files to be stored within the application (TODO: actually at the moment the whole text is stored in the db), they are discarded after being processed. Additionally the user account and login functionality has not yet been implemented.

- explain what we actually implemented and how it is different from the proposed application in FA2 (point to section about improvements)

II. Platform choice

The considerations detailed below led to choosing to work with *Microsoft Azure* to remain within the given time scope for the coursework.

A. Setup

At first we tried to use the *Oracle Cloud* and to build and maintain a Kubernetes cluster we wanted to use the *Terraform* command line tool. This was the recommended way we got from the Oracle's Team tutorials in class and for this we had an amount of 3500 free credits for the *Oracle Cloud*. After a long period of installing all needed tools e.g. *Terraform* itself, *Terraform Provider OCI* [11], *Terraform Kubernetes Installer for Oracle Cloud* [8], we tried to create an easy example cluster. But already at this point we run into service limits. These limits are restrictions on how many instances of a specific resource in a zone you are allowed to use. To increase the limit you have to open a ticket request and after a few days there will be an Oracle support guy helping you. As the ticket request system is very inconvenient and complicated, and the whole process takes several days this is a very strong disadvantage of the *Oracle Cloud* in comparison to *Microsoft's Azure*.

The second point of inconvenience in using the *Oracle Cloud* is the unstructured and inconsistent UI of the website, where you can control and maintain all Oracle services. Especially in comparison to other UIs, e.g. Amazon's, which one we discovered in the lecture, and Microsoft's, which we used finally to configure the cluster and database for our web application.

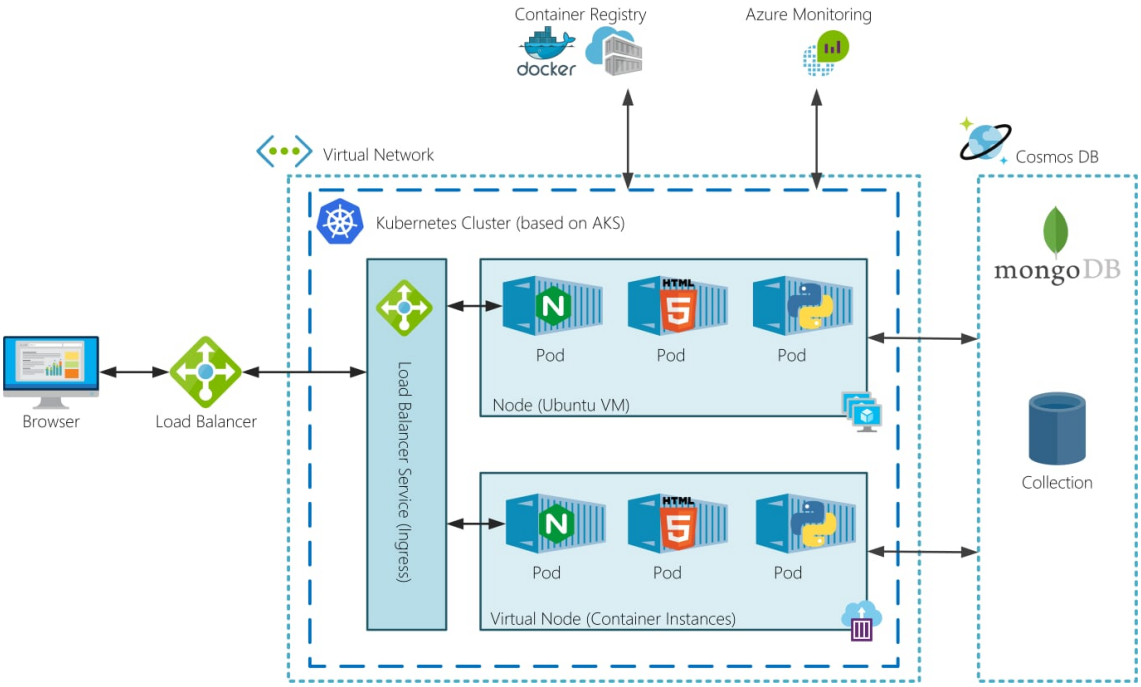

Fig. 1. System architecture of our Kubernetes cluster.

B. Documentation and support

An even bigger advantage of using one of *"the big three"* in the cloud business, so *Google Cloud*, *Amazon AWS* or *Microsoft Azure*, is there are a lot of tutorials, answered questions on *Stackoverflow*,etc and better documentations. Respectively using Oracle's cloud leads to a very small number of tutorials, which is one of the reasons why it is more difficult for beginners. There were a lot of errors or bugs when running there example code, too, but searching for solutions of the problems with *Terraform* and *Oracle Cloud* was very unsuccessful and often ends in no results or other users questions having the same problems. So after increasing the service limits and running in errors again and again, we got to a point, where we decided to switch to *Microsoft Azure* and deploy the cloud application Text Analytica there. In retrospect this was one of our best decisions of the whole project.

C. More Advantages

In addition to the advantages mentioned in Section II-A and Section II-B, there are some more reasons why we decided to migrate the project to Azure: There is an option to use so called *Virtual Nodes* respectively *Virtual Kubelets*, which fasten scaling times. The functionality and how this is used in the Kubernetes cluster is explained in Section III. Finally there is a bigger community and probably more jobs for the Azure Cloud Services, so it is more meaningful for us to gain experience with this cloud provider instead of Oracle Cloud.

III. System Architecture and Service Implementation

Figure 1 shows the architecture of the implemented system, which is discussed in the first two parts of this section in more detail. Text Analytica's two main components are a Kubernetes cluster, managed by the *Azure Kubernetes Service (AKS)* [1] and an *Azure Cosmos DB* [4], storing the results of the analysis. In the last part of this section, the service, its underlying code and the continuous integration pipeline used to deploy the application are presented.

A. Infrastructure

Text Analyticas front-end, back-end and analysis service are currently run within a single Docker container. Container orchestration is handled by a Kubernetes cluster managed by AKS, where each container is encapsulated by a Kubernetes pod. When a new pod is started up the corresponding container is created from a container image pulled from the container registry *Docker Hub* [7].

The Kubernetes master node, responsible for the cluster operation, is fully managed by AKS. In addition to the master node the cluster consists of two worker nodes. The first one is based on a general purpose Linux VM, while the second is implemented as a virtual node based on *Azure Container Instances (ACIs)* [3] and the *Virtual Kubelet open source project* [12] [13]. Kubernetes treats the ACIs compromising the virtual node like standard nodes, so new pods can simply be provisioned on them. Based on container images, ACIs are ready to use in a few seconds as no virtual machines have to be started up and managed by the user. In terms of service level they could be described as *containers-as-a-service*.

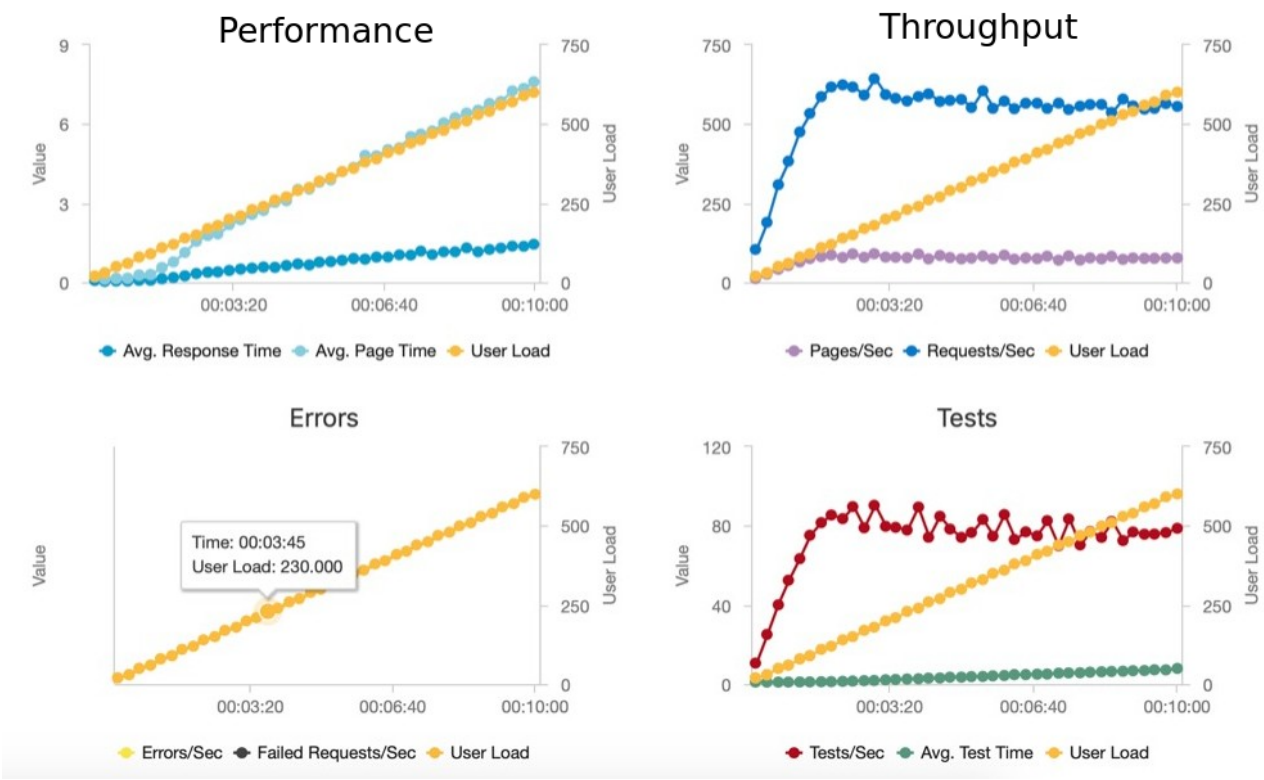

Fig. 2. Load tests of the Kubernetes cluster.

The cluster is monitored using Azure Monitor [6], a collection of tools to monitor, query and log services and infrastructure running on Azure. It monitors the health of the cluster itself, its nodes and the running services and containers.

To make the pods containing the containerized application available to the public several network measures are in place [2] [5]. First, a Kubernetes service of type NodePort has been created to allow access to the pods via IP address or DNS name and port. To expose the services of Text Analytica for external access an ingress service was used. Next to application level load balancing which at this point is not necessary for Text Analytica, as its services still all run within a single container ingress can for example be used for SSL / TLS termination. Next to the ingress service an NGINX ingress controller is deployed as a pod to each node. The ingress service is of type LoadBalancer, which leads Azure to create and configure an Azure Load Balancer resource with a corresponding external IP address.

B. Data storage

As mentioned in [section limitations] currently the only user data stored by Text Analytica are the results of the analysis and related metadata. This data is combined into a json object and sent to the Cosmos DB.

Cosmos DB is a globally-distributed, multi-model database [https://docs.microsoft.com/de-de/azure/cosmos-db/introduction]. It was chosen for this project, because it is very easy to set up from the Azure portal, is fully managed and scaled by Azure and can be treated like a MongoDB in development using an API [https://docs.microsoft.com/de-de/azure/cosmos-db/mongodb-introduction].

To provide persistent storage not affected by dying and restarting containers, the database is decoupled from the Kubernetes cluster.

The data in the database is shielded from unwanted access, as it is only accessible from the virtual network in which the Kubernetes cluster is hosted.

C. Service implementation

Front-end built from template using vue.js and bootstrap [https://startbootstrap.com/template-overviews/freelancer/]

Backend built with Python using the Flask framework based on several tutorials

We used Azure Devops to setup a continuous delivery pipeline to adapt the concept of DevOps. If new code is pushed to the git repository of text analytica, then a new docker container is built automatically and pushed to the docker hub registry. Afterwards the updated containers are applied onto the AKS cluster. The described steps are performed by an Azure pipeline, a service of Azure devops to create build pipelines.

IV. Scalability

In case we will reach the point our web application gets more and more famous, and will be used more often the scalability of the application has a huge impact on the software

experience and loading times. To build a good cloud application it should handle an unknown number of users parallel uploading and analyzing files without running into errors or bigger performance bottlenecks. In the rest of this section we discuss how we created a scalable cloud application and show results of some load tests.

A. Infrastructure

As mentioned in Section III all services are managed by the Kubernetes cluster. The main advantage of using a Kubernetes cluster is scalability and loadbalancing. So if there is a high usage of the existing resources the cluster can scale up by itself. In our case this might happen at peak times when a lot of people are using the Text Analytica web service. Than the cluster creates more (TODO: describe more carefully what exactly happens here) [9].

As we are using *Virtual Kubelets*, we can achieve even faster up- and downscaling times than with the pure Kubernetes cluster itself. This is due to the fact we do not need to wait on new VMs or container booting up (TODO: check this). [10].

B. Data storage

- PaaS and other region

C. Service

- container and microservices -¿ why is this important for scalability?!

D. Monitoring and Load Testing

To know how good a cloud application scales with an increasing amount of work respectively more users a load test is perfect for. We used the *Azure Loadtest*, as this is an inbuild tool of the Azure Cloud Services and works efficiently for our usecase.

In Figure 2 are the results of one of our load tests. In the figure there are four diagrams, for *Performance*, *Throughput*, *Errors* and *Tests*. This test lasted ten minutes and involved 700 virtual users. In the *Errors* diagram can be seen there is no error, which means our application still works even under high usage. In the *Performance* part there is the increasing user load and the increasing average response time as well as average page time. It follows our application has a good scaling rate, as the response time scales nearly linear with a fast performance. The *Throughput* figure shows the pages/sec and requests/sec with increasing user load, and *Tests* simply shows the tests/sec andaverage test time.

A useful tool for monitoring the Kubernetes cluster is the *Kubernetes Dashboard*, which can be seen in Figure 3. Here all important ressources can be watched , e.g. number of working pods, CPU usage and memory usage. For debugging and in case of errors this tool can be really helpfull.

V. Future improvements

A. Infrastructure

Split up microservices, front-end, back-end and analysis into separate containers, this would also allow for ingress to act as an application layer load balancer directing incoming traffic to the right service

Another possibility would be to go serverless, that is using PaaS and tools like Azure functions

Improve management of the kubernetes cluster by using popular open source projects like Istio.

B. Service implementation

Solve small bugs like missing ID in display

Add functionality like login, pdf, text recognition, more advanced analysis, save files and retrieve them, ...

Add / consider security measures, as of now there has been no focus on this

Extend the existing Continuous Delivery Pipeline and add test and dev stages to it.

VI. Conclusion

conclusion...

References

[1] Aks. https://docs.microsoft.com/de-de/azure/aks/intro-kubernetes. Accessed: 2019-01-08.

[2] Aks networks. https://docs.microsoft.com/de-de/azure/aks/concepts-network. Accessed: 2019-01-10.

[3] Azure container instances. https://docs.microsoft.com/en-us/azure/container-instances/container-instances-overview. Accessed: 2019-01-10.

[4] Azure cosmosdb. https://docs.microsoft.com/de-de/azure/cosmos-db/introduction. Accessed: 2019-01-08.

[5] Azure expose kubernetes. https://blog.jreypo.io/containers/microsoft/azure/cloud/cloud-native/how-to-expose-your-kubernetes-workloads-on-azure/. Accessed: 2019-01-10.

[6] Azure monitor. https://docs.microsoft.com/en-us/azure/azure-monitor/overview. Accessed: 2019-01-10.

[7] Docker hub container registry. https://hub.docker.com. Accessed: 2019-01-08.

[8] Github terraform kubernetes installer. https://github.com/oracle/terraform-kubernetes-installer. Accessed: 2019-01-07.

[9] Microsoft azure kubernetes service aks. https://docs.microsoft.com/en-us/azure/aks/. Accessed: 2019-01-08.

[10] Microsoft documentation virtual node. https://azure.microsoft.com/de-de/resources/samples/virtual-node-autoscale/. Accessed: 2019-01-08.

[11] Terraform provider for oracle cloud infrastructure github. https://github.com/terraform-providers/terraform-provider-oci. Accessed: 2019-01-10.

[12] Virtual kubelet. https://github.com/virtual-kubelet/virtual-kubelet. Accessed: 2019-01-10.

[13] Virtual kubelet github. https://github.com/virtual-kubelet/virtual-kubelet/blob/master/providers/azure/README.md. Accessed: 2019-01-10.

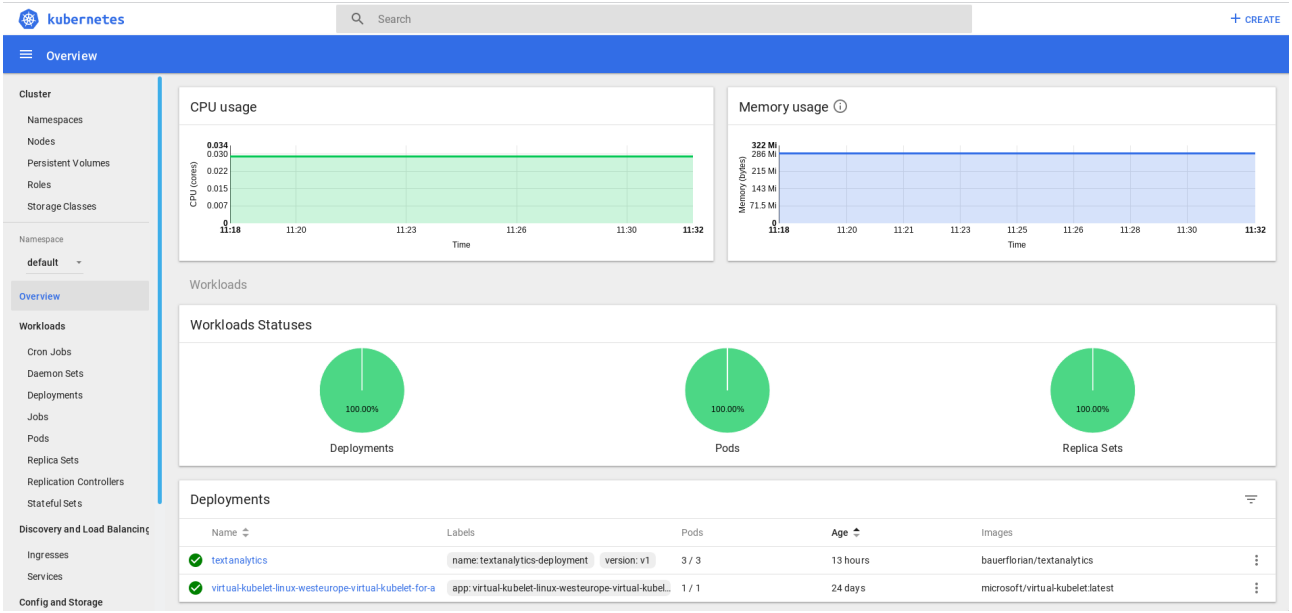

Fig. 3. Kubernetes Dashboard