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*—Text Analytica is a cloud-based web service for document analysis. The prototype introduced and discussed in this report currently accepts txt file uploads and extracts the three most common words. The web service is hosted on *Microsoft Azure* and makes use of serveral cloud computing technologies. The functionality of Text Analytica is containerised into Docker containers orchestrated by a Kubernetes cluster. The data generated from the text analysis is stored in an *Azure Cosmos DB*. All technologies used were chosen for their positive impact on scalabitity. The scalability of Text Analytica was examined by the means of a load test. 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 web service aiming to support the analysis of text documents. For the purpose of this coursework a prototype has been developed, deployed and evaluated using different cloud computing technologies. In the remainder of this section, the concept of Text Analytica is discussed as well as the limitations of the implemented prototype. Section II explores the reasons for 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. Different aspects of the service's scalability are addressed in Section IV, including evidence of the performance of the service under load. Lastly future improvements of Text Analytica are proposed in Section V.

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 familiarise 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 many scenarios in which people are confronted with a large number of 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, while remaining within the proposed time frame. Therefore, the functionality of the submitted 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 parsed and analysed. At this point the analysis merely tags the system entries with the three most common words in the text. The content of the text file is stored as a string as part of the analysis, therefore currently there is no reason to store the actual files within the application and they are discarded after being processed. Additionally, no user account and login functionality has not yet been implemented.

II. Platform choice

The considerations detailed below eventually led to choosing *Microsoft Azure* instead of the services introduced in the lectures by Oracle to remain within the proposed time scope for the coursework.

A. Setup

Given the tutorials in class as well as the $£3,500$ in credits granted to us for *Oracle Cloud*, we initially intended to build our application based on their services. We started trying to set up a Kubernetes cluster with the *Terraform* command line tool, as was recommended to us. However, after having spent a significant amount of time on installing all necessary tools, e.g. *Terraform* itself, the *Terraform Provider OCI* [1] and the *Terraform Kubernetes Installer for Oracle Cloud* [2], we failed to set up an easy example cluster due to reaching service limits. These limits restrict how many instances of a specific resource in a zone one can use. To increase the limit a ticket with Oracle support must be opened, a process which is very slow and inconvenient. While *Microsoft Azure* does also have such limits [3], they are far more generous and therefore allowed us to work more efficiently, without depending on the reply of third parties.

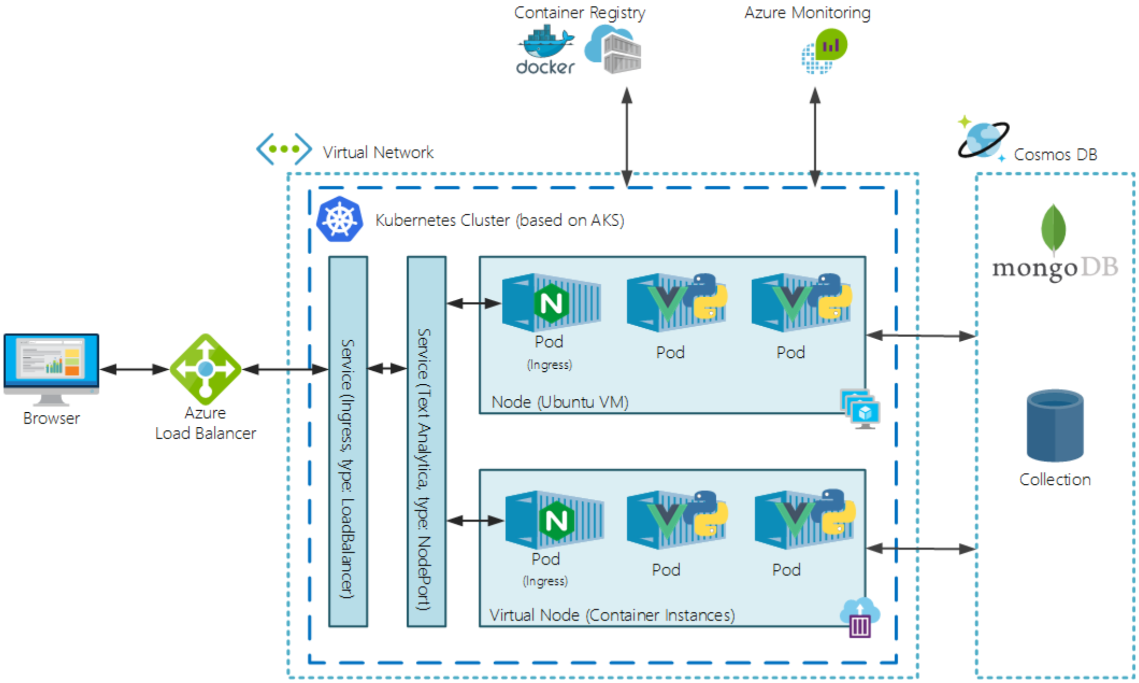

Fig. 1. Text Analytica's system architecture: The main components of the system are a Kubernetes cluster based on AKS, orchestrating the containers running Text Analytica and a *Microsoft Cosmos DB*, storing the results of the analysis. Each container is encapsulated by its own Kubernetes Pod. The images to build new containers can be pulled from the *Docker Hub* container registry. Azure monitoring is used to monitor the health of the cluster and its components. The Kubernetes cluster is exposed for external access through an *Azure Load Balancer*.

Another inconvenience of *Oracle Cloud* is the unstructured and inconsistent user interface of the control panel. This became especially apparent when comparing it to the user interfaces of Amazon, as demonstrated in the lecture, and Microsoft, which was eventually used to configure the cluster and database for Text Analytica. In addition to this, Azure made it very easy to work with third party SDKs as it provides good integration of these services within their own ecosystem.

B. Documentation and support

Working with one of *"the big three"* providers of cloud computing, that is *Google Cloud*, *Amazon Web Services* or *Microsoft Azure* comes with considerable advantages, when it comes to documentation and support. Not only are there large numbers of tutorials, answered questions on *Stackoverflow* and blog articles on their technologies, but they also come with very comprehensive documentations. The number of tutorials using Oracle's cloud services, however, was relatively small, making it difficult for beginners to get started. When running example code we encountered quite a few errors and bugs, but searching for solutions to these issues for example with *Terraform* and *Oracle Cloud* was often unsuccessful.

C. Scalability

In addition to the advantages mentioned in Section II-A and Section II-B there is one more reason why we decided to migrate the project to *Azure*. It offers the possibility to create *virtual nodes* based on *Virtual Kubelets* [4], enabling faster scaling. A more detailed explanation of this technology is given in Section III.

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) [5] and an *Azure Cosmos DB* [6], storing the results of the analysis. In the last part of this section, the implementation of the service and the continuous integration pipeline used to deploy the application are presented.

A. Infrastructure

Text Analytica's 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) [8] and the *Virtual Kubelet* open source project [4], [9]. Kubernetes treats the ACIs compromising the

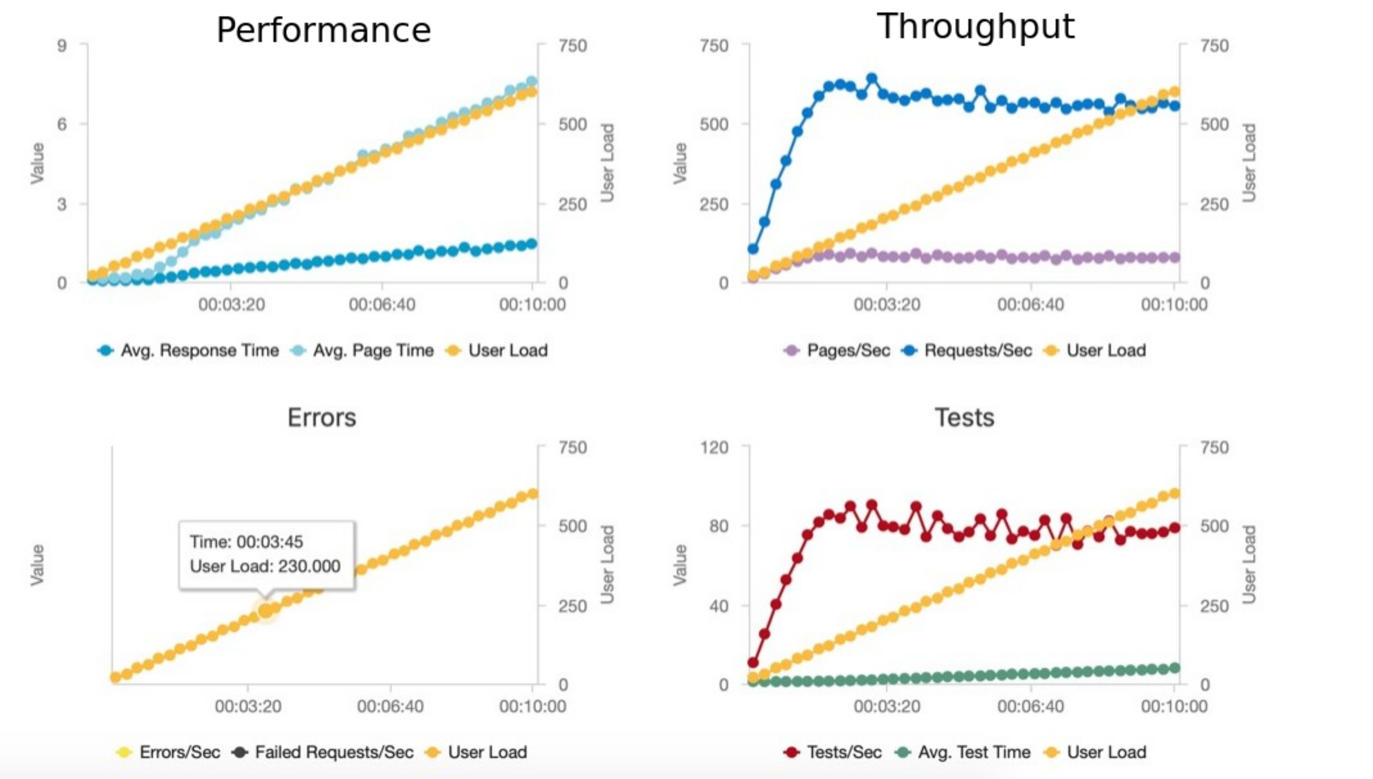

Fig. 2. Results of a load test of the Kubernetes cluster. Most importantly from the *Performance* chart it can be seen that Text Analytica's performance increases linearly when the number of users is increased. There is no significant loss in performance either.

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 must be started up and managed by the user. In terms of service level, they could be described as containers-as-a-service.

The cluster is monitored using *Azure Monitor* [10], 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 containerised application available to the public several network measures are in place [11], [12]. 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 I-B currently the only user data stored by Text Analytica is 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 [6]. 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 [13].

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

Text Analytica's back-end and the analysis functionality are programmed in Python. The back-end uses the Flask framework, which is a web development microframework [14]. While the implementation of the back-end is based on several online resources such as tutorials and videos [15]–[17], the front-end is based on a template using Vue.js and Bootstrap [18].

To enhance the deployment of the webservice, a continuous delivery pipeline, in this case an *Azure Pipeline* [19], was created. Whenever new code is pushed to Text Analytica's GitHub repository, a new Docker container image is built automatically using the Dockerfile and pushed to the Docker

Hub registry. These new images are then pulled by the AKS cluster to start updated containers to run the service. The service used to build this *Azure Pipeline* is part of *Azure DevOps*.

IV. Scalability

In a production environment, the number of Text Analytica users is expected to grow. As discussed in the lecture, sudden bursts in usage can sometimes not be foreseen [20]. Therefore, it was crucial to pay attention to the scalability of the application, to ensure a smooth and fast user experience independent of the number of parallel accesses. The application was built to handle an unknown number of users uploading and analysing files in parallel without running into errors or significant performance bottlenecks. In the remainder of this section it is discussed how this issue was addressed and results of one load test is shown.

A. Infrastructure

As mentioned in Section III the web service is managed by a Kubernetes cluster. The main advantages of using a Kubernetes cluster are easy scalability and load balancing. More specifically, if usage of existing resources is high, the cluster will automatically scale [21], [22]. In Text Analytica's case this might happen at peak times when a lot of people are using the Text Analytica web service. The cluster will then spawn more *Pods* on the existing *Azure VM Node*, to handle the additional workload without sacrificing performance [23].

When the *Azure VM Node* is used to capacity new Pods are provisioned based on *Virtual Kubelets* on a *virtual node*. On the one hand this allows the application to start up new Pods faster, as there is no need to wait on new VMs being booted up as already explained in Section III-A, on the other hand this enables infinite scaling as there is no capacity limit for a virtual node based on ACIs. Due to higher prices this option should, however, not be used permanently and in case of observing lasting high usage, more traditional VM-based nodes should be provisioned.

B. Data storage

Text Analytica uses an instance of Azure's *Cosmos DB* for storage. The following part of this section discusses how scalability is addressed in terms of the data storage.

Cosmos DB supports two kinds of partitioning: logical and physical partitions. Logical partition are such partitions, where all datasets or elements in a container share a property, which can serve as a partition key. In this case the data itself and the data throughput is automatically, horizontally partitioned and scaled by *Cosmos DB*. Then there are the physical partitions. A number of logical partitions are assigned to one physical partition, which guarantees persistence and consistency of the data [24]. However, currently Text Analyticas database entries are not configured to have a partition key for logical partitioning. This is an improvement that eventually should be made.

C. Monitoring and Load Testing

To measure how well Text Analytica scales with an increasing workload respectively more parallel users we conducted a load test with the *Azure DevOps* load testing solution [25].

Figure 2 displays the results of one of our load tests. The four diagrams show data for *Performance*, *Throughput*, *Errors* and *Tests*. The test lasted ten minutes and involved seven hundred virtual users at peak time, accessing the website. At this point we have not yet implemented a test case where actual files are uploaded to the application. From the *Errors* diagram we observe that no errors occurred, which means our application did not fail even under high usage. The *Performance* part contrasts the increasing user load, the increasing average response time as well as average page time. Interpreting this result, we conclude Text Analytica has a good scaling rate, as the response time scales nearly linear with no major increase in latency. The *Throughput* figure shows the pages/sec and requests/sec with increasing user load, and *Tests* simply shows the tests/sec and average test time.

A useful tool for monitoring the Kubernetes cluster is the *Kubernetes Dashboard*. It shows all important resources, e.g. the number of Pods running, CPU and memory usage. This tool was especially helpful for debugging.

V. Future improvements

As mentioned before, because of the time frame proposed for the development of this coursework project and especially due to initial issues with Oracle's cloud services, the submitted version of Text Analytica is just a prototype. Therefore there are some improvements on different levels, which are discussed in this section.

A. Infrastructure

At the moment, as Text Analytica has not much functionality, all of it is run on one container. To allow more specific scaling of certain parts of the application, the services should be split up into microservices, each running on its own container. Based on the current implementation the functionality could be split up into back-end, front-end and analysis. Not only would this separation allow for a more specific scaling of the services, but it would also allow to implement ingress rules, so that the ingress can be used as an application layer load balancer.

Another way to optimise Text Analytica would be improved management of the Kubernetes cluster itself. There are several popular open source projects, such as Istio, which can be used to efficiently manage a mesh of microservices [26].

A more drastic change to the system architecture would be the shift to serverless, implementing the different services as *Azure functions* for example [27]. Instead of the current IaaS approach, where some set up and maintenance has to be done by the user, this would mean using Azure's services on a PaaS level, allowing developers to focus on the application's functionality, while provisioning of resources and scaling is handled completely automatically by Azure.

B. Service implementation

As the focus of this coursework assignment was the application of cloud technologies and optimising the application for scaling, there are quite a few improvements on the service implementation level to be made in the future.

Firstly, there are some small bugs in the current implementation, such as the missing creation of an actual ID for the uploaded and analysed documents, which would need to be fixed. Secondly there is still some functionality missing to make Text Analytica an useful tool. Such functionality includes but is not limited to login and user account management, PDF uploads and processing, more advanced analysis as well as the option to store and retrieve files within or from the application.

To turn Text Analytica into a production level application it must comply to certain security standards. While developing the prototype there was no focus on security and therefore only minimal / default security measures are currently in place.

Lastly the existing continuous delivery pipeline could be extended for example by adding testing stages.

VI. Conclusion

conclusion...

References

[1] "Terraform provider for oracle cloud infrastructure," https://github.com/terraform-providers/terraform-provider-oci, accessed: 2019-01-10.

[2] "Terraform kubernetes installer for oracle cloud infrastructure," https://github.com/oracle/terraform-kubernetes-installer, accessed: 2019-01-07.

[3] "Azure subscription and service limits, quotas, and constraints," https://docs.microsoft.com/de-de/azure/azure-subscription-service-limits, accessed: 2019-01-12.

[4] "Virtual kubelet," https://github.com/virtual-kubelet/virtual-kubelet, accessed: 2019-01-10.

[5] "Azure kubernetes service (aks) ii," https://docs.microsoft.com/de-de/azure/aks/intro-kubernetes, accessed: 2019-01-08.

[6] "Welcome to azure cosmos db," https://docs.microsoft.com/de-de/azure/cosmos-db/introduction, accessed: 2019-01-10.

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

[8] "What is azure container instances," https://docs.microsoft.com/en-us/azure/container-instances/container-instances-overview, accessed: 2019-01-10.

[9] "Kubernetes virtual kubelet with aci," https://github.com/virtual-kubelet/virtual-kubelet/blob/master/providers/azure/README.md, accessed: 2019-01-10.

[10] "Azure monitor overview," https://docs.microsoft.com/en-us/azure/azure-monitor/overview, accessed: 2019-01-10.

[11] "Network concepts for applications in azure kubernetes service (aks)," https://docs.microsoft.com/de-de/azure/aks/concepts-network, accessed: 2019-01-10.

[12] "How to expose your kubernetes workloads on azure," https://blog.jreypo.io/containers/microsoft/azure/cloud/cloud-native/how-to-expose-your-kubernetes-workloads-on-azure/, accessed: 2019-01-10.

[13] "Azure cosmos dbs api for mongodb," https://docs.microsoft.com/de-de/azure/cosmos-db/mongodb-introduction, accessed: 2019-01-10.

[14] "Flask: web development, one drop at a time," http://flask.pocoo.org, accessed: 2019-01-12.

[15] "Running flask on kubernetes," https://testdriven.io/blog/running-flask-on-kubernetes/, accessed: 2019-01-12.

[16] "Python flask and vue tutorial - video 1," https://www.youtube.com/watch?v=iwhuPxJ0dig, accessed: 2019-01-12.

[17] "A mongodb driven flask application in docker from scratch," https://www.youtube.com/watch?v=AAPOCB1U1kg, accessed: 2019-01-12.

[18] "Freelancer: one page bootstrap portfolio theme," https://startbootstrap.com/template-overviews/freelancer/, accessed: 2019-01-10.

[19] "Azure pipelines," https://docs.microsoft.com/en-us/azure/devops/pipelines/?view=vsts, accessed: 2019-01-12.

[20] "Aws case study: Animoto," https://aws.amazon.com/de/solutions/case-studies/animoto/, accessed: 2019-01-12.

[21] "Scaling your containers with kubernetes," https://blog.codeship.com/scaling-your-containers-with-kubernetes/, accessed: 2019-01-10.

[22] "Kubernetes autoscaling 101: Cluster autoscaler, horizontal pod autoscaler, and vertical pod autoscaler," https://medium.com/magalix/kubernetes-autoscaling-101-cluster-autoscaler-horizontal-pod-autoscaler-and-vertical-pod-2a441d9ad231, accessed: 2019-01-10.

[23] "Azure kubernetes service (aks) i," https://docs.microsoft.com/en-us/azure/aks/, accessed: 2019-01-08.

[24] "Partitioning and horizontal scaling in azure cosmos db," https://docs.microsoft.com/de-de/azure/cosmos-db/partition-data, accessed: 2019-01-10.

[25] "Run url-based load tests with azure devops," https://docs.microsoft.com/en-us/azure/devops/test/load-test/get-started-simple-cloud-load-test?view=vsts, accessed: 2019-01-12.

[26] "What is istio," https://istio.io/docs/concepts/what-is-istio/, accessed: 2019-01-12.

[27] "Azure functions documentation," https://docs.microsoft.com/en-us/azure/azure-functions/, accessed: 2019-01-12.