MASTER THESIS

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Smart City

Kubernetes on the Edge

By: Bernd KLAUS, BA

Student Number: 2010303012

Supervisors: Dipl.-Ing. Hubert Kraut
Dipl.-Ing. Andreas Happe

Wien, January 29, 2022

FH University of Applied Sciences TECHNIKUM WIEN

Declaration

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz /Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.“

Wien, January 29, 2022 Signature

Kurzfassung

Kubernetes wird als Schweizer Armemesser der Container-Orchestrierung bezeichnet. Auch im Bereich Edge-Computing bietet der Dienst eine Vielzahl an unterschiedlichen Werkzeugen und Tools an, welche Teils unterschiedliche Strategien und Ansätze verfolgen. Die Auswahl reicht von einem zentralen Kubernetes-Cluster der verteilte Geräte, sogenannte „Leafs“, steuert bis hin zu vielen einzelnen und verteilten kleinen Clustern an der Edge, welche zentral gesteuert werden. Entscheidend ist es den richtigen Anwendungsfall zu erheben, um sich für die optimale Lösung entscheiden zu können. Ebenfalls spielen sicherheitstechnische Aspekte bei derart komplexen Umgebungen eine wichtige Rolle. Die vorliegende Arbeit gibt Einblicke und Entscheidungsgrundlagen sowie Empfehlungen hinsichtlich der IT-Security. Belegt werden die Angaben durch Implementierung eines Proof-of-Concepts

Schlagworte: Kubernetes, Edge-Computing, distributed System, Proof-of-Concept

Abstract

Kubernetes is the de facto swiss-army-knfife for orchestrating conatiner-platforms. In addition, Kubernetes can also be faciliated for deploying devices as well as applications on top of it on the edge of the network. However, there are different methods for archiving comparable results. On the one hand a possible solution is to build a central instance managing small distributed and independent clusters, on the other hand a centralized cluster with just leafs on the edge may is a better fit. This results in the challenge to find the best solution for the desired environment respectively use-case. The following thesis is making use of "Design Science Research" to give introductions on how to choose the proper architecture for the aimed environment.

Keywords: Kubernetes, Edge-Computing, distributed System, Proof-of-Concept

Contents

1 Introcution

Because of Internet-of-Things (IoT) Devices becoming more and more common, the number of devices capable of communicating with the world wide web (WWW) increases rapidly. Consequently, also the overall traffic generated as well the amount of data which must be processed increases accordingly. Regarding this development Edge-Computing is the rising start trying to solve that issues. Thereby data is not processed centrally like in traditional datacenters, but it is tried to handle those data close to the user within several distributed systems. Because of this methodlogy only really necessary data is transmitted to a central instance for further treatment and those the processing-power as well as the bandwidth necessary for processing required data is reduced significant.

It is expected that the number of IoT devices will continue to grow fast[1] over the coming years. Concomitant Edge-Computing also will become more important in the future and become an important role in modern Information Technology (IT) architectures.

To be able to control distributed systems effectively Kubernetes (K8S) is providing a lot of useful tools and functions. Fundamentally there are three different approaches regrading the architecture of how to build an Edge-Computing environment making use of K8S:

- A centralized K8S Cluster controlling many Leaf-Devices (Workers) on the Edge.
- Small and distributed K8S Clusters running independent on the Edge controlled by a centralized Master-Instance.
- A Service-Mesh expanding the functionality of the K8S networking stack.

1.1 Problem area

Problems arise when trying to find the proper architecture for a specific use-cases. There is no clear winner when comparing the above-mentioned different variants. Each of them have their own pros and cons and may decide whether a project is successful or not. It is therefore all the more important to choose the proper architecture right before starting, changing the strategy in retrospect would take a lot of time and effort. However, there is no clear guidance on how to find the proper target environment, at least none which apply in general. Occasionally one finds recommendations for very specific use case, however the chance is slim low this findings fit your goals respectively enlighten the architecture decision. This leads us to the following research question.

1.2 Research question

This paper is going to answer the subsequent research questions:

1. What are the main differences of the above mentiones architectures regarding functionality, scalability, costs and security?
2. Which decision criteria must be defined respectively examined to create a catalog capable of making choose the proper architecture easier for IT managers as well as administrators?
3. Is there a trend in which technology is most likely to be used?

1.3 Goal

The main goal of this thesis is to highlight the pros and cons for each of the architectures defined in the Introduction. The focus will mainly be on the geo-distribution aspect. Altough IoT is playing a major role in pushing the development forward, however it is not considered further in the present work. To find the proper architecture, or at least recommandations what could fit best for different desired use-cases, a catalog will be defined. An important part will become the decision tree helping people making comprehensible decisions based on scientific research. The meain characteristics which are taken into account are scalability, state-of-the-art, handling, costs as well as security.

1.4 Methodology

In the first part of the present work existing literature will be inspected. Related and relevant work will be examined accordingly and linked in the document. Also results will be incorporated to get out the most of it. In the second part a catalog with main criteria necessary for decision-making is defined. Part of this catalog will also be a decisicon-tree, mentioned in the previous chapter, to easily find the proper architecture. The last chapter deals with testing the defined criteria against real world examples making use of the Design Science Research (DSR) methodology. The last chapter is getting the most focus because it is the area where new technics respectively architecture decision are finally verified and those proofs if the catalog is working as expected or not. In the latter case, the catalog will be revised to reflect the findings of the last step and re-examinated again using DSR.

2 State of the Art

The first chapter gives a brief introduction to the main technologies used respectively examinated in the later part of the document. If anyone is already familiar with the subject may you jump over to the Architecture chapter.

2.1 Technology

2.1.1 Kubernetes

To promote mondern development and be able to implement continious deployment pipelines cumbersome monolytic applications are divied into many smaller units. Each of this units provides only one function. In order to establish the overall functionality, these units are communicate with eacher other and thus provide services or make us of other ones. This new method of delivering applications brings many advantages in terms of development but also introduce some new challenges and complexities regarding operation. To simplify the tasks around the management of this architecture, K8S has established itself as the de facto standard [2]. Over the course of time, a broad community has developed around kubernetes and a number of additional tools and extensions have emerged as a result. The mopst promising solutions regrading geo-distribution respectively edge-computing are highlighted in the subsequent section Architecture. In order to be able to interpret the results of the use-cases, as well as building the necessary basic understanding, the following functionalities and components of K8S are of relevance.

Master Nodes run the so called *Control Plane* which is resposible for controlling the cluster itself and all the ressoruces within. The Control Plane consist of the following components [3].

- *kube-apiserver* acts as frontend webinterface responsible for controlling the K8S cluster. Tools like kubectl abstract the interface and provide access in form of an simply understandable and usable Command Line Interface (CLI).
- *etcd* represents a high-available and consistent key value store responsible for storing the actual state as well as the desired configuration of the cluster.
- *kube-scheduler* is responsible for scheduling pods on the available worker nodes. Decision variables such as available ressources, affinity-rules and constraints are taken into account. However the default *kube-scheduler* ist not aware of any latency between the worker nodes nor the pods communicating with each other. As discovered in the hereafter chapters, this appears to be an important variable for edge-deployments. However, some available white-papers already try to address those issues and showed possible solution

by adopting a custom scheduler taking care of those values. More details on this can be found in the chapter Related Work.

- *kube-controller-manager* consists of a single compiled binary controling the status of nodes, jobs, service-accounts and endpoints as well as creating or removing them.
- *cloud-controller-manager* represents the interface to the underlying cloud-platform. This allows kubernetes to create and/or configure load-balancers, routes and persistent-volumes on the underlying cloud-infrastructure. In a local environment such as e.g. minikube provide the *cloud-controller-manager* becomes an optional component and is not required. The same may applies to edge-locations as those areas are outside the cloud most of the times.

Worker Nodes manage the workload, i.e. run the actual application(s). These nodes are composed of the following, see list below, parts [3]. It should be mentioned, that also the previously descripte *Master Nodes* are executing those components because some of the core-componentes are containerized (pods) itself.

- *kubelet* is an agent which assures that the container is executed properly inside their associated pods according to its specifications defined via *PodSpec*. Also *kubelet* is responsible for monitoring the healthy state of the containers.
- *kube-proxy* uses the packet filters of the operating system underneath to forward traffic to the desired destination. The resulting access points, also called *Services* in K8S-jargon, can be made available either internally or externally.
- *container runtime* is the part that finally executes the containers. The default runtime at time of writing is *containerd*, however any runtime is supported that complies with the CRI specification [4].

2.1.2 Edge-Computing

latency reliability and availability low ressources may bad network conditions
Target length: 1-2 S

Geo-Distribution short introduction - whats that! Target length: 0,5-1 S

2.2 Architecture

2.2.1 Default

Centralized Master and Workers at the Edge (Default!) K8s architecture why is it called Default challanges Target length: 2-4 S (incl. picture)

2.2.2 distributed K8s

Cluster running indepented Target length: 2-4 S (incl. picture)

2.2.3 Service Mesh

Using the Service-Mesh for Edge-Computing (Smar-Nic!) Target length: 2-4 S (incl. picture)

3 Design Science Research

3.1 Methodlogy

3.1.1 Performed Tests

Target length: 2 S

3.2 Environment

describe the test-environemtn Target length: 0,5-1 S

3.3 Architecture

3.3.1 Default

Target length: 3-4 S

3.3.2 distributed K8s

Target length: 3-4 S

3.3.3 Service Mesh

Target length: 3-4 S

3.4 Use-Cases

3.4.1 Web-Application

Target length: 1

3.4.2 Enterprise VPN

Target length: 1

3.4.3 Distributed Database

Target length: 1

3.5 Analysis

Target length: 4 S

3.5.1 Relevant Magnitudes

3.5.2 Performed Tests

3.5.3 Outcome

3.5.4 Paraphrase

4 Catalog

4.1 Decision Variables

Target length: 1-2 S

4.2 Decision Tree

Target length: 1 S

4.3 Exclusions and Special Cases

Target length: 1 S

5 Related Work

Target length: 3 S (all subsections)

5.1 Kubernetes and the Edge?

Some introdution to K8s at the Edge, highlighting the main Architectures.

5.2 Extend Cloud to Edge with KubeEdge

Descripes KubeEdge and its advantages

5.3 Sharpening Kubernetes for the Edge

Sharpening Kubernetes for the Edge Make Kubernetes aware of the latency between the nodes at the edge.

5.4 Ultra-Reliable and Low-Latency Computing in the Edge with Kubernetes

Similar to the paper before. Latecny awar pod deploymentm, but you also can deploy to regions and an custom re-scheduler is implementated taking care of redeploying when one node fails. Clustering node-groups based on latency.

6 Results

Target length: 3 S (all together)

6.1 Findings

6.2 Conclusio

6.3 Discussion and further research

Notes —to-be-removed— Sites: - longest: 50 (may i need even more) - shortest: 36 (zu wenig) ——————-

Bibliography

[1] F. Arjomandi, M. Trifiro, and J. Smith, “State of the edge 2021, A market and ecosystem report for edge computing,” The Linud Foundation, research rep., Aug. 1, 2020. [Online]. Available: https://stateoftheedge.com/reports/state-of-the-edge-report-2021/.

[2] Portworx, “Kubernetes adoption survey,” PureStorage, research rep., Mar. 26, 2021. [Online]. Available: https://www.purestorage.com/content/dam/pdf/en/analyst-reports/ar-portworx-pure-storage-2021-kubernetes-adoption-survey.pdf.

[3] T. K. Authors. “Kubernetes components.” (Oct. 17, 2021), [Online]. Available: https://kubernetes.io/docs/concepts/overview/components/.

[4] T. K. Authors, *Container runtime interface (cri), A plugin interface which enables kubelet to use a wide variety of container runtimes.* [Online]. Available: https://github.com/kubernetes/cri-api.

List of Figures

List of Tables

List of Code

List of Abbreviations

IT Information Technology

WWW world wide web

K8S Kubernetes

IoT Internet-of-Things

DSR Design Science Research

CLI Command Line Interface

A Anhang A

B Anhang B