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

Vienna, March 19, 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.“

Vienna, March 19, 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-knife for orchestrating container-platforms. In addition, Kubernetes can also be used 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 be 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, geo-distribution, proof-of-concept

Contents

1 Introdutcion

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 methodology 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 two different approaches regrading the architecture of how to build an edge-computing environment making use of K8s:

- **Default**: A centralized K8s Cluster controlling many leaf-devices (workers) on the Edge.
- **Distributed**: Small and distributed K8s Clusters running independent on the Edge controlled by a centralized Master-Instance.

Another upcoming approach of solving that issue is making use of the service mesh [2]. This ultimately uses or builds on both of the aforementioned technologies. However, since this thesis concentrates on the two main architectures and their differentiation, the service mesh is not the main focus and just mentioned for the sake of completeness.

1.1 Problem area

Problems arise when trying to find the proper architecture for a specific use-case. There is no clear winner when comparing the above-mentioned different variants. Each of them has 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 a very specific use case, however the chance is slim low this findings fit your goals respectively enlighten the decisions. 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 in chapter 1 mentioned architectures regarding functionality, scalability, costs and security?
2. Which decision criteria must be defined respectively examined to create a catalog capable of choosing 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 be mainly on the geo-distribution aspect. Although 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 recommendations 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 main 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. The goal is to create a catalog with main criteria necessary for decision-making. Part of this catalog will also be a decision-tree, mentioned in the previous chapter, to easily find the proper architecture. The design science research (DSR) method serves as a scientific method and to test the characteristics recorded in the catalogue. This chapter is given the most attention, it is the area where new techniques or architectural decisions are finally verified and the proof is given whether the catalogue works as expected or not. In the latter case, the catalog will be revised to reflect the findings of the last step and re-examined again using DSR.

2 State of the Art

2.1 Technology

The present chapter provides an introduction to the general thematic. The main components and objects of K8s are explained aswell as the layers of edge-computing are highlighted. If anyone is already familiar with the subjects, may you jump over to the section 2.2 "architecture" to read further.

2.1.1 Kubernetes

To promote modern development and be able to implement continuous deployment pipelines cumbersome monolithic applications are divided into many smaller units. Each of these units provides only one function. In order to establish the overall functionality, these units are communicating with each other and thus provide services or make use 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 [3]. K8s was initially developed by Google and later donated to the opensource community. Over the course of time, a broad community has developed around K8s and a number of additional tools and extensions have emerged as a result. The most promising solutions regrading geo-distribution respectively edge-computing are highlighted in the subsequent section 2.2 "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.

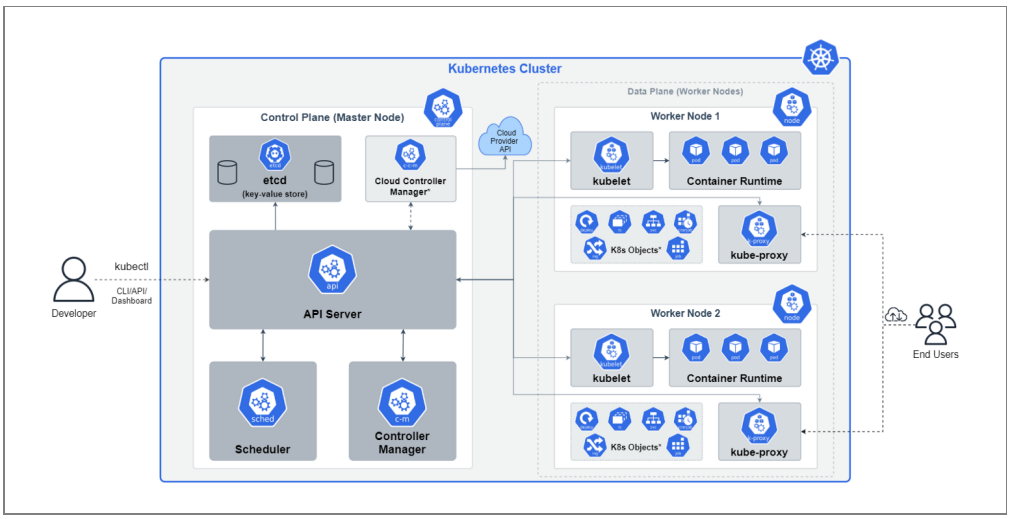

Figure 1: K8s architeture overview [4]

Master Nodes run the so-called *Control Plane* which is responsible for controlling the cluster itself and all the ressources within. The Control Plane consist of the following components [5].

- *kube-apiserver* acts as frontend web-interface responsible for controlling the K8s cluster as well as the objects inside the cluster. Tools like kubectl abstract the *OpenAPI v2* endpoint and provide access in form of a 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 following chapters, this appears to be an important variable for edge-deployments. However, some available white-papers already try to address those issues and show possible solutions by adopting a custom scheduler taking care of those values. More details on this can be found in the chapter 5 "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 e.g. minikube [6] provide the *cloud-controller-manager* becomes an optional component and is not required. The same may apply to edge-locations as those areas are outside the cloud most of the time.

Worker Nodes manage the workload, i.e. run the actual application(s). These nodes are composed of the following, see list below, parts [5]. It should be mentioned, that also the described *Master Nodes* are executing those components because some 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 [7].

Kubernetes Objects are persistent properties inside the K8s ecosystem representing the state of a cluster. The most important feature of those objects is to describe the target environment in a declarative way. For this purpose, most of the time, YAML files are used. Kubernetes now ensures that the desired state of the environment is actually achieved and continuously monitors the required objects to meet those defined requirements. This mechanism is also ideal for distributed systems, such as edge computing, as availability can be monitored at any time and a an action can be taken if necessary. Subsequent the main objects are cited starting with the smallest unit [8].

- *Containers* decouple the actual application and its dependencies from the underlying infrastructure. The main properties of those containers are there immutability and repeatability. This means that the container can be rebuilt at anytime resulting in an identical clone. Likewise, the code of a running container cannot be modified subsequently.

- *Pods* include at least one or more *Containers*. In the most scenarios a single pod consists of a single container, in some cases a so-called sidecar container is used increasing the number of containers inside a pod. Containers which are in the same *Pod* share the same local Socks as well as volumes mounted.

- *Deployments*, *Statefulsets* and *Daemonsets* are responsible for ensuring the actual workload is provided, to achieve this they control and scale the assigned *Pods*. When creating an application for K8s, it is most likely to create one of those objects. The *Pods* and *Containers* are merely an end product that is derived these objects.

- *Services* provide an abstract way to make a set of *Pods* available on the network via a single endpoint. Additional deployed pods will automatically be added to the responsible *Services*. Thereby K8s is an excellent choice when it comes to scaling applications without any manual intervention. This also applies for deploying applications to the edge of the network, as illuminated in the course of this thesis. Closely related to the *Services* is the *Ingress* resource, which is taking care of making the aforementioned objects available outside the cluster. An optional reverse-proxy (*Ingress-Controller*) must be installed in order to make use of the latter.
  A new feature, which is of relevance regarding edge-computing, currently in beta phase, is the so-called *Topology Aware Hint*. Basically its meta-data added to the endpoints defined previously suggesting the connection client on how to reach the destination efficiently (e.g. zones aware of different locations can be defined)

- *ConfigMaps* and *Secrets* are pieces of information which can be mounted into to *Container* to adjust the configuration inside at runtime. Even whole files can be replaced

using on of them. *Secrets* are only different in the sense that they decode the content, however technically they are the same.

- *Volumes* provide persistent storage which extends beyond the life cycle of the pods. Volumes can be mounted at any defined position inside the pods. The disadvantage is that the data written to those *Volumes* resides out of the K8s ecosystem and therefore the operator must take care of data security and replication. This becomes even more complicated in an edge-computing environment where nodes have higher latency between them.

2.1.2 Edge-Computing

Edge-computing is the model that extends cloud services to the edge of the network. The computing resources on the edge act as a layer between the user, who provides or wants to process data, and the centralized datacenter (e.g. the cloud). Because data can be processed earlier respective closer to the user, latency and amount of data transferred can be reduced [9]. Also, the required computing-ressources in the datacenter can be minimized because data can be processed at the edge. A major driver of the subsequent s technology is IoT. The amount of devices and resulting data volume, which must be processed, is increasing exponentially [1]. Another technology which depends on it are low-latency applications like e.g. video-streaming.

Hierarchy descripes the layers of the architecture. The following list enumerates the most important layers from top to bottom [9].

1. *Cloud* - centralized datacenter
2. *Fog* - distributed "smaller" datacenters
3. *Edge* - the closest unit to the end-user
4. *IoT* - device at the edge put into use

The main focus of this work is to efficiently combine the two layers *Cloud* and *Edge* and orchestrate between them using K8s. The layer *Fog* is skipped because it is often seen to be "the same" as the *Edge*. Also, current K8s based solutions do not make use of it.

Geo-Distribution characterises the aspect of the geographical propagation of the edge ressources. The goal is to provide computing power over wide areas, each close to the users. By establishing many of these locations in different regions, network latency can be significantly reduced from the user's perspective. However, the latency between the edge-nodes and the centralized cloud still remain.

2.2 Architecture

This chapter focuses on the two different architecture approaches which can be used for edge computing. After an overview the advantages and disadvantages as well as possible solutions are examinated.

2.2.1 Default

In order to be able to manage resources at the edge, a traditional architecture can be used. This is subdivided into a centralized control plane hosted in the cloud and distributed worker nodes near the edge. The same K8s architeture is commonly used when deploying to a single location aswell inside the cloud. The following graphic illustrates the architecture.

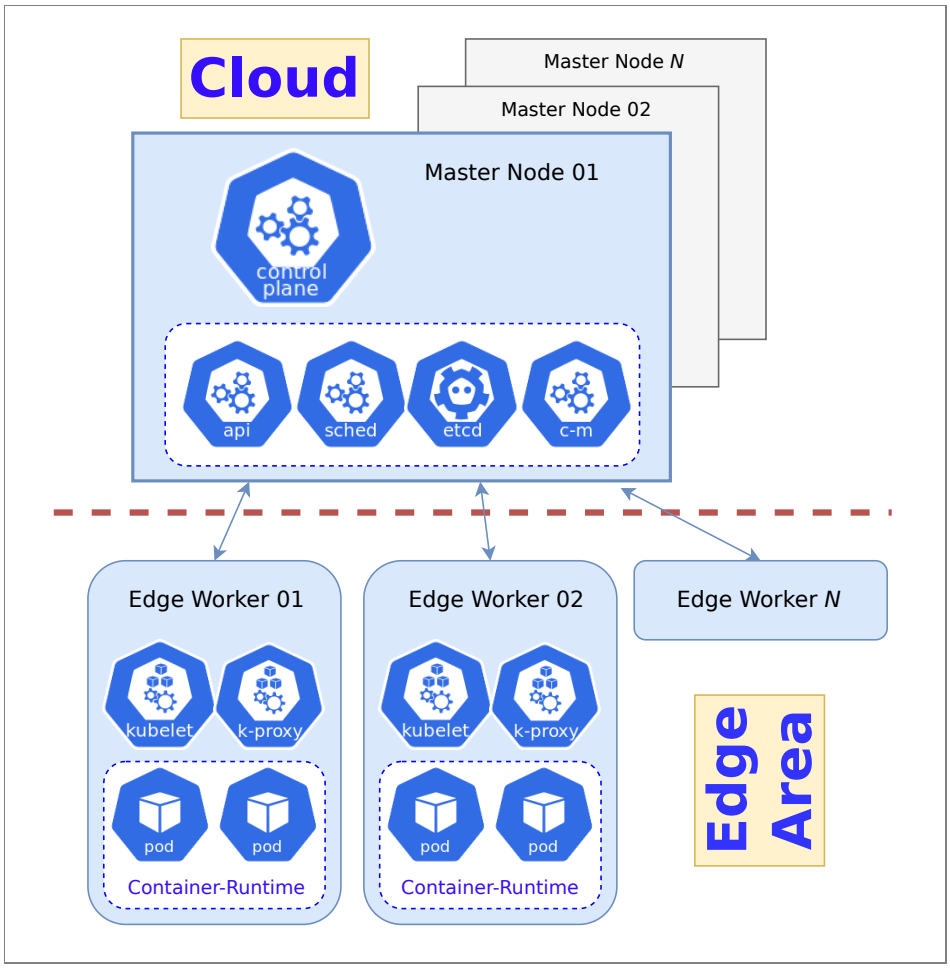

Figure 2: Default K8s architeture

Although the conventional components can also communicate with each other over long distances, there are still some challenges that need to be taken into account. An important role in this context is played by the container network interface (CNI). This part of K8s, which can be

selected in the form of a plugin, is responsible for the cluster-internal network traffic. Most of these plugin providers offer advanced features that facilitate geographical distribution. Some of the most used CNI providers are [10]:

- Calico
- Canal
- Cilium
- Flannel
- Weave

Other issues, which must be observed, are the replication of the storage and metadata hold by the control planes. Because of the higher latency and-or unstable connection at the edge site data may cannot always be reliably retrieved. To circumvent this problem, asynchronous replication can be used for storage replication. Another solution is to make services at the edge stateless, thereby no big data chunks are requested at all. The important part of the metadata store are domain name system (DNS) entries, because K8s heavily relies on them for service discovery. NodeLocal DNS is the recommand way [11] to hold a copy on the worker nodes. In contrast to this forwarding data to the cloud is not an issue in most of the times when using suitable message queues.

In addition there is no awarness of where workloads are running and the level of latency between nodes when using vanilla kubernetes. There are some developments in this area, but they have not really caught on yet [12][13][14].

KubeEdge is an open-source Cloud Native Computing Foundation (CNCF) project [15] that already comes with many of these functions respectively requirements pre-charged. Likewise, this tool was explicit developt for edge-computing. Worker nodes can therefore be distributed across the hole globe without any major adjustments. To make this possible, some components were added or exchanged [16].

- *KubeBus* - a custom network plugin working in private internet procotcol (IP) address ranges behind network address translation (NAT). Thereby no dedicated public ip is necessary for edge locations making the architecture more flexible.
- *SyncService* - Another important part of KubeEdge is the metadata sync service running on each worker node. This extension cyclically synchronises the data of the master. Thus, the quantity of data to be called up can be minimized and workload can continue to run in offline scenarios.
- *EdgeCore* - Kubelet is replaced with a lightweight custom agent, the EdgeCore. Thereby also devices with very limited ressources, e.g. a Raspberry Pi, can be used for running workloads.

In addition a component for device management and seemles integration with message queuing telemetry transport (MQTT) is built-in to KubeEdge[15]. However, because this thesis focuses on the geo-distribution aspect, this part is not researched further.

2.2.2 Distributed K8s

In this case, a different approach is chosen and, in contrast to the previous architecture, fully-fledged clusters are also operated at the edge. The obvious advantage is that they can be operated autonomously. There is basically no dependence on the other nodes.

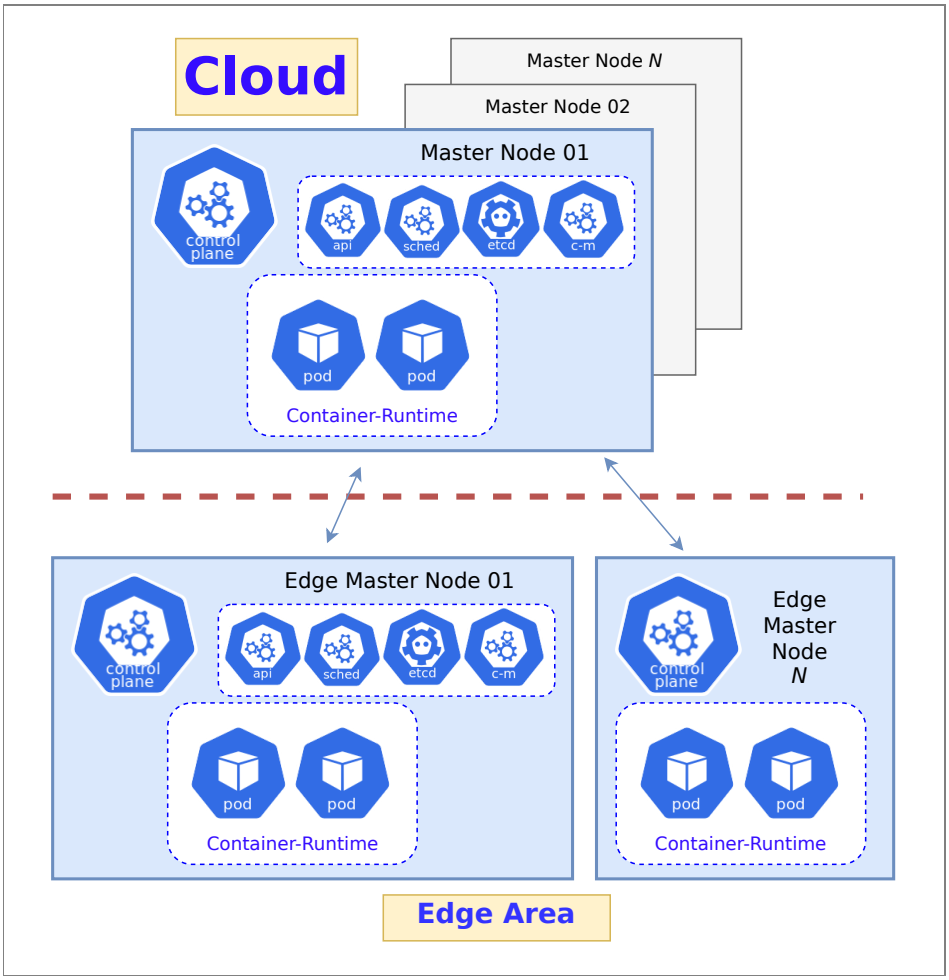

Figure 3: Distributed K8s architeture

However, this also creates new challenges such as the distribution and coordination of workloads. Without additional functionality added, the workload running on the clusters does not know about their neighbours. To overcome this gap, the division "multi-cluster deployment" has excelled in recent years. Even Google, the original developer of K8s, offers such a service [17] in their public cloud. Due to the increased demand, the Kubernetes cluster federation (KubeFed) tool is officially being maintained and further developed by the K8s project[18]. Although the

tool is still in beta, it is a very good choice for our field of application. One possible alternative would the K8s manager by Rancher Labs.

KubeFed consists of a central hosting cluster which controls subordinate clusters via application programming interface (API) delegation. Within KubeFed you can define which configuration should be managed by the hosting cluster. The methodes used are intentionally low-level and can thus be expanded well for different edge-deployment variants. Two different types of information is used to configure KubeFed[18]:

- *Type configuration* - determines which API types are handeld.
- *Cluster configuration* - determines which clusters should be targeted.

The type configuration itself consist of three main parts:

- *Temaplates* - defines a reprenentation of common ressources across the clusters.
- *Placement* - defines which destination-cluster the workload should run on.
- *Overrides* - defines varation of the templates on a per cluster level.

2.3 General Challenges

Challenges that appear when creating respectively operating such infrastructures are diverse, as can be seen in the following list [9].

- *Variety* - a lot of different locations, technologies as well as methods on how to control devices on the edge is challenging for both development and operation. The better these different factors can be abstracted and simplified, the more effectively the infrastructure can be used.
- *Integration* - edge-computing evolves very quickly, thereby things could change quickly. The more important it is to keep the provided interfaces extensible. This way, new devices or application can be put in use swiftly.
- *Awareness* - the devices and/or end-user do not care about how their traffic is routed, or their data is processed. However, the architecture needs to take care of that to use the topology in the best possible way.
- *Resources* - scaling Resources like central processing unit (CPU), random access memory (RAM) and disk space at the edge is by far more elaborate than in a datacenter or in the cloud.

- *quality of service (QoS)* - The service provided at the edge should be reliable and provide a good user experience. Availability and performance play a central role in this context. As the availability can not be guaranteed to some extent, an appropriate failover mechanism should be in place.
- *Security* - physical access control as well as isolating applications from each other is a difficult task. Also, the data traffic must be separated accordingly. In general, IT security is a hot topic, and especially at the edge, it requires appropriate consideration for hardening the environment.
- *Monitoring* - another important factor is how to capture metrics and events (logs) from the edge. They need to be indexed on a centralized instance in order to get a general overview on what's happening. Because of the dynamic and rapid changes some kind of automatic discovery should be used for that purpose.
- *Environment* - Some locations may have to deal with difficult conditions regarding their surroundings. Increased dust exposure, poor internet connections or recurring power outages can be some of these factors. The system must be able to cushion or parry such failures accordingly.

The above-mentioned challenges provide a good starting point for defining the necessary tests to find the matching target architecture. Details about this test can be found in the section 3.1 "methodlogy".

3 Design Science Research

"A common topic when performing research in technical disciplines is to design some kind of artefact, such as a model, an information system, or a method for performing a certain task. To address this topic in a systematic and scientific way, Design Science Research (DSR) has stablished itself as an appropriate research method." [19]

3.1 Methodology

According to the description in the introduction before, DSR is used as a scientific method to construct the proof-of-concept (PoC). For this purpose two real world exmaples are build, which are then gradually refined. Those exmaples are descripte in detail in the following section 3.2 "environment". The same are then deployed onto each of these environments. Afterwards, Tests are then defined and carried out on the basis of the issues listed in section section 2.3

"genereal challenges". Based on the results, a preferred environment per property can finally be determined.

3.2 Environment

3.2.1 Generel

The main goal is to create both of the aimed architectures in a similar environment. The target environment should provide good coverage of the diversity encountered in edge environments. For that poropse the PoC is build across Hetzner Cloud and a local site connected via 4G mobile internet.

Coverage This combination allows a wide range of possibilities to be achieved:

- *Geographical distribution* - Hetzner Cloud offers location in Germany, Finland and the U.S for deployments. This allows communication to take place over long distances.
- *Scalability* - On the Hetzner Cloud scaling nodes can be realised quickly via API.
- *NAT* - At the local site, nearby vienna, no public IP is availabe for each of the nodes. Therefore, NAT is used to map a dynamic changing IP to the nodes downstream.
- *advanced RISC machines (ARM)* - Also at the local site, a Raspberry Pi is used as an edge-node to emulate devices with minial ressources.
- *Unstable* - The connection at the local site may be unstable from time to time because the connection is established over the public 4G network using mobile technology.

Locations Subsequent, a tabular lists all of the nodes that are used for each of the both test environments, followed by a graphic showing the locations on a map.

Node	Location	IP	CPU	Cores	RAM	Latency
Master	Nürnberg, DE	dedicated	AMD64	2	4GB	0ms
Edge-1	Ashburn, US	dedicated	AMD64	2	2GB	95ms
Edge-2	Helsinki, FI	dedicated	AMD64	2	2GB	24ms
Raspberry	Kirchberg, AT	dynamic	ARM64	4	4GB	40ms

Table 1: PoC nodes specification

Figure 4: Map of the PoC[20]

OS The operating system (OS) used on all cloud nodes is Ubuntu 20.04.3 LTS. For the local site on the Raspberry Pi the OS Raspberry Pi Lite, without graphical user interface (GUI), is used.

K8s For simplicity, the tool of choice for installing K8s is K3s. This is a lightweight K8s distribution optimized for IoT and deployments at the edge. All necessary features are supported, only the dependencies have been reduced to the essentials. In actual use, insofar as resources in the cloud only play a subordinate role, the standard K8s can also be used analogously. However, at the edge K3s if a perfect match. To illustrate the simplicity, the installation command used is shown below.

```
curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="--disable traefik
    --disable-cloud-controller" sh -s -
```

Code 1: K3s installation

VPN No virtual private network (VPN) is used for either environment. Although a VPN can increase security accordingly and services can communicate more easily, its use in a widely distributed edge environment with many nodes is problematic. VPNs are simply not designed for unstable connections with high latency and possibly rapidly changing IP addresses. The two tools used therefore use alternative approaches via HTTPs and RPC respectively.

3.2.2 KubeEdge

For the default architecture resrespectively KubeEdge variant, only a K8s installation on the master node is required. Edge nodes just need be to equipped with a supported containter runtime of choice. In our specific PoC Docker is used. KubeEdge takes over the roles of

the kublete and the k-proxy as shown in the figure "default K8s architecture". A single binary (keadm) is used to initialise KubeEdge. A token and the public IP are specified as parameters on the master. On the edge nodes, those information is used to join the master.

Challenges The most important steps and challenges encountered during the installation are listed below.

- *CloudStream* - While the standard installation can be done easily, activating logs is much more challenging. Activities such as generating certificates, setting environment vairables and adapting configuration files must be carried out manually.

Installation Steps for installation can be founf in the Github repository: Berndinox/K8sEdge

3.2.3 KubeFed

In contrast to KubeEdge, as also noted in subsection 2.2.2, KubeFed relies on individually acting clusters that are controlled by a central instance. Because of this, each node must be equipped with a fully functional K8s cluster. Usually, the installation is associated with greater effort, but K3s simplifies this process enormously, as descripted in section 3.2.

Challenges The most important steps and challenges encountered during the installation are listed below.

- *Hosting Cluster* - The challenges arise when it comes to connecting all instances from the central cluster. The kubeconfig for each of them must be modified, to include the puclic ip or full qualified domain name (FQDN) of the node, and transmitted to the central hosting cluster. There, the configuration must be added as an additional context. Finally, a custom binary is used to add each of the defined contexts to KubeFed, as shown below.

```
kubefedctl join edge1 --cluster-context edge1 --host-cluster-context
    default --v=2
```

Code 2: KubeFed join context

- *Dynamic IP* - Because the central hosting cluster is initialising the connection to the nodes at the edge, these nodes must be reachable over a static adress. In most cases the use of static ips is not a problem. However, especially in edge-environments, the use of dynamic ip-addresses may be necessary. To circumvent this problem, a so-called dynamic DNS service is used to establish the connection. Although such a service can be set up relatively quickly, it does mean that an additional component has to be installed and maintained. In contrast, KubeEdge works without such a workaround.

- *dNAT* - As long as the target node is behind a router and connected with a private ip, destination NAT must be configured to forward the necessary port. Alternatively, a reverse-proxy can be used to publish the internal service. This step is also only required for the KubeFed variant.

Installation Steps for installation can be founf in the Github repository: Berndinox/K8sEdge

3.3 Use-Cases

3.3.1 Web-Application

3.3.2 Enterprise VPN

3.3.3 Distributed Database

3.4 Performed Tests

3.5 Analysis

3.5.1 Relevant Magnitudes

3.5.2 Outcome

3.5.3 Paraphrase

4 Catalog

4.1 Decision Variables

4.1.1 Installation complexity

The first decision variable that is examined is the effort required for the installation and the associated complexity. According to the KISS principle[21], those solutions with less complexity should be preferred. One investigates the installation routine, descripted in the section 3.2 "environment", very different steps and necessary tools were discovered.

Dependencies The first important variable we look at in this appendix are the dependencies.

- *Connectivity* - While KubeEdge works with literaly any connection, even behind carrier-grade NAT[22], important conditions must be checked for KubeFed, as descripted in subsection 3.2.3 under the paragraph "challenges". For the sake of completeness, the requirements for KubeFed are listed subsequent.
  - Static IP or dyn DNS agent if dynamic assigned
  - NAT rule if the device is located behind a router
- *Device Support* - Both solutions offer a wide selection of different types of devices supported, like: ARM64, ARMv7, x86, or x64 based systems. However, because KubeFed relies on a full functional K8s cluster those extensive support is provided by the use of K3s and the container-runtime packaged within. In contrast, KubeEdge only requires on one of the supported container-runtimes as well as the KubeEdge binary. Because KubeEdge is taking special care of implementing lightweight agents compatible with container runtime interface (CRI)[23][24], an even wider range of devices can be supported. Due to the design, even edge-devices with fewer resources can be used effectively. In summary, both solutions support a wide range of devices. KubeEdge, however, goes one step further and is therefore slightly in the lead.
- *Other dependencies* - Other requirements such as bandwidth, connection quality and e.g. storage space have little or no influence on the installation itself and are therefore only considered in the following operational part.

Installation routine The second step is to examine the installation routine itself.

- *Required Tools* - One important part is the number of tools and binaries used respectivley required to be able to run the environments. The fewer components are used, the less complex the installation and operation. The fewer components used, the greater the tendency to keep the environment simple. Although individual components can also involve quite a high degree of complexity, this variable must not be disregarded. The individual configuration steps are evaluated in the following sections. All tools are listed for each environment subsequent.
  - *KubeEdge*
    - CRI
    - KubeEdge binaries
    - K3s (Master only)
  - *KubeFed*
    - K3s
    - KubeFed binaries (Master only)

    * Helm (Master only)

  It should be noted that K3s includes or builds on CRI.

- *Configuration steps* - This property describes the necessary steps that were required during the installation without additional tools. For KubeEdge the basic installation, without any logging capability, is only done by using the binary. However, if logging is necessary (in almost all cases it should be), some adaptations must be made. For Kubefed, in contrast, the respective kubeconfig must be transferred to the master and added there as a custom context. Minor adaptation with little effort is necessary for both environments, yet the comparison is not balanced because KubeFed does not offer out-of-the-box the possibility to collect all logs of the edge devices.
- *Documentation* - Both documentations appear well maintained and kept up to date. The KubeEdge updates seem a little more recent according to the Github history. The KubeFed documentation can only be read in the form of markdown files within Github, but there is a separate web page for KubeEdge.

Node	Location	IP	CPU	Cores	RAM	Latency
Master	Nürnberg, DE	dedicated	AMD64	2	4GB	0ms
Edge-1	Ashburn, US	dedicated	AMD64	2	2GB	95ms
Edge-2	Helsinki, FI	dedicated	AMD64	2	2GB	24ms
Raspberry	Kirchberg, AT	dynamic	ARM64	4	4GB	40ms

Table 2: PoC nodes specification

4.2 Decision Tree

4.3 Exclusions and Special Cases

5 Related Work

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

6.1 Findings

6.2 Conclusio

6.3 Discussion and further research

Bibliography

[1] Fay Arjomandi, Matt Trifiro, and Jacob Smith. *State of the Edge 2021. A Market and Ecosystem Report for Edge Computing*. Research rep. The Linud Foundation, Aug. 1, 2020. URL: https://stateoftheedge.com/reports/state-of-the-edge-report-2021/.

[2] CNCF. *Service Mesh - from technical selection to best practice*. Ed. by Malphi. Mar. 15, 2018. URL: https://www.cncf.io/wp-content/uploads/2020/08/rfma-servicemesh-cncf.pdf.

[3] Portworx. *Kubernetes Adoption Survey*. Research rep. PureStorage, Mar. 26, 2021. URL: https://www.purestorage.com/content/dam/pdf/en/analyst-reports/ar-portworx-pure-storage-2021-kubernetes-adoption-survey.pdf.

[4] Patel Ashish. *Kubernetes — Architecture Overview*. Aug. 12, 2021. URL: https://bit.ly/3sMQyEE.

[5] The Kubernetes Authors. *Kubernetes Components*. Ed. by Tim Bannister. Oct. 17, 2021. URL: https://kubernetes.io/docs/concepts/overview/components/.

[6] The Kubernetes Authors. *Minikube Start*. Ed. by Tian Yang. Nov. 19, 2021. URL: https://minikube.sigs.k8s.io/docs/start/.

[7] The Kubernetes Authors. *Container Runtime Interface (CRI). a plugin interface which enables kubelet to use a wide variety of container runtimes.* Ed. by Tim Bannister. Jan. 28, 2022. URL: https://github.com/kubernetes/cri-api.

[8] The Kubernetes Authors. *Kubernetes Concepts*. Ed. by Tim Bannister. June 22, 2020. URL: https://kubernetes.io/docs/concepts/.

[9] Al-Dulaimy Auday et al. *Introduction to edge computing*. Sept. 1, 2020. DOI: 10.1049/PBPC033E_ch1.

[10] Mike Mackrory. *The Ultimate Guide To Using Calico, Flannel, Weave and Cilium*. June 7, 2021. URL: https://platform9.com/blog/the-ultimate-guide-to-using-calico-flannel-weave-and-cilium.

[11] The Kubernetes Authors. *Using NodeLocal DNSCache in Kubernetes clusters*. Ed. by Tim Bannister. Jan. 31, 2022. URL: https://kubernetes.io/docs/tasks/administer-cluster/nodelocaldns/.

[12] Haja David et al. *Sharpening Kubernetes for the Edge*. Research rep. MTA-BME Network SoftwarizationResearch Group, Mar. 17, 2020.

[13] Reincke Karsten and Steiner Lukas. *k8s-edge-scheduler*. Ed. by Reincke Karsten and Steiner Lukas. Mar. 31, 2019. URL: https://github.com/telekom/k8s-edge-scheduler.

[14] Chima Ogbuachi Michael et al. *Context-Aware Kubernetes Scheduler forEdge-native Applications on 5G*. Research rep. SoftCOM, Mar. 1, 2020.

[15] Manaouil Karim and Lebre Adrien. *Kubernetes and the Edge?* Research rep. Inria Rennes, Oct. 22, 2020.

[16] Xiong Ying et al. *Extend Cloud to Edge with KubeEdge*. Tech. rep. Seattle Cloud Lab, Oct. 27, 2018. URL: https://ieeexplore.ieee.org/document/8567693.

[17] Google Inc. *Multi-Cluster-Dienste*. Ed. by Google Inc. Feb. 15, 2022. URL: https://cloud.google.com/kubernetes-engine/docs/concepts/multi-cluster-services.

[18] The Kubernetes Authors. *Kubernetes Cluster Federation*. Ed. by The Kubernetes Authors. Mar. 10, 2022. URL: https://github.com/kubernetes-sigs/kubefed.

[19] Benner Wickner Marian, Kneuper Ralf, and Schlömer Inga. *Leitfaden für die Nutzung von Design Science Research in Abschlussarbeiten*. Research rep. IUBH Internationale Hochschule, Nov. 1, 2020. ISRN: 2512-319X.

[20] *Google Maps*. Mar. 12, 2022. URL: https://maps.google.com.

[21] Smith Byron. *How to KISS: The Art of Keeping it Simple, Stupid*. Research rep. 723 W Aspen Ave, Ste: X Double Dot LLC, Apr. 23, 2012.

[22] Richter Philipp et al. *A Multi-perspective Analysis ofCarrier-Grade NAT Deployment*. Research rep. TU Berlin, May 1, 2016. URL: https://www.researchgate.net/publication/303330116_A_Multi-perspective_Analysis_of_Carrier-Grade_NAT_Deployment.

[23] The Kubernetes Authors. *Container Runtime Interface (CRI)*. Ed. by The Kubernetes Authors. Jan. 10, 2022. URL: https://kubernetes.io/docs/concepts/architecture/cri/.

[24] Gpinaik. *CRI Support in edged*. Ed. by Dai Looong et al. Sept. 2, 2021. URL: https://github.com/kubeedge/kubeedge/blob/master/docs/proposals/cri.md.

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

QoS quality of service

CPU central processing unit

RAM random access memory

CNI container network interface

IP internet procotcol

NAT network address translation

DNS domain name system

MQTT message queuing telemetry transport

CNCF Cloud Native Computing Foundation

KubeFed Kubernetes cluster federation

API application programming interface

PoC proof-of-concept

ARM advanced RISC machines

OS operating system

GUI graphical user interface

VPN virtual private network

FQDN full qualified domain name

CRI container runtime interface

A Appendix