Workload Deployment and Configuration Reconciliation at Scale in Kubernetes-Based Edge-Cloud Continuums

Daniel Hass
Endress+Hauser InfoServe
Weil am Rhein, Germany
daniel.hass@endress.com

Josef Spillner
Zurich University of Applied Sciences
Winterthur, Switzerland
josef.spillner@zhaw.ch

***Abstract*—Continuum computing promises the abstraction of physical node location and node platform stack in order to create a seamless application deployment and execution across edges and cloud data centres. For industrial IoT applications, the demand to generate data insights in conjunction with an installed base of increasingly capable edge devices is calling for appropriate continuum computing interfaces. Derived from a case study in industrial water flow monitoring and based on the industry's de-facto standard Kubernetes to deploy complex containerised workloads, we present an appropriate continuum deployment mechanism based on custom Kubernetes controllers and CI/CD, called Kontinuum Controller. Through synthetic experiments and a holistic cross-provider deployment, we investigate its scalability with emphasis on reconciling adjusted configuration per application and per node, a critical requirement by industrial customers. Our findings convey that Kubernetes by default would enter undesirable oscillation already for modestly sized deployments. Thus, we also discuss possible solutions.**

***Index Terms*—Continuum computing, scalability, configuration management**

I. Introduction

Field monitoring of physical phenomena is a growing source of distributed data streams that are collected, processed and used for generation of insights and inference of knowledge about the phenomena. Such phenomena include wildlife behaviour, earthquakes, human traffic and water flows. Apart from challenges relating to the data volume and velocity, a key challenge is the underlying network topology. Data collection is performed at extreme network edges while data processing including cross-correlation is typically performed at central locations such as cloud services. Consequently, edge-cloud and fog-cloud trends that have led to the notion of computing continuums over the past years are now seeing an increased deployment in the field to support field monitoring with simplified application models. This shift is further supported by the widespread availability of commodity cloud stacks such as Kubernetes shipping with various implementations that adjust to a hardware range from embedded edge systems to data centre equipment. In such deployments, software only needs to be written and packaged once and can then be flexibly deployed to where it fits best from an operational perspective.

Deployment is, however, a complex process, especially at scale. Industrial field monitoring deployments achieve scale by a large number of heterogeneous devices, a large number of applications, and a large number of customers with specific preferences. Hence, from a Kubernetes perspective, it is not sufficient to deploy static deployment descriptors or application charts. Instead, per-{application, device, customer} configuration adjustments need to be applied in the right order at deployment time, centered around resource constraints such as the allocation of virtual CPU cores and main memory.

In this paper, we first give a background on continuum computing, industrial IoT and Kubernetes workloads. Then, we introduce in depth a case study in water flow monitoring that is representative of field monitoring in an industrial IoT/edge-to-cloud transmission context. Our first contribution is then a custom Kubernetes controller to support workload deployment across the resulting continuum. This is followed by our second contribution, the execution of sufficiently scaled synthetic deployments. While discussing the results, we highlight problems with configuration reconciliation and present our third contribution, the discussion of possible solutions.

II. Background

A. IIoT and Continuum Computing

The concept of IoT is meant to tackle automation challenges through increased digitalisation and connectivity of plants and installed devices. Aside from the intrinsic endeavour of the businesses that are part of the respective application domain, such as water sensing, to digitalise their processes to enhance productivity, suppliers in this industry sector are also offering solutions based on the concept of IoT. Those solutions shall provide the end user as well as the supplier with new ways for value generation that are linked to increased productivity and efficiency, cost reduction, better customer/citizen experience, faster innovation and new revenue streams [1], [2].

The industrial application of the broad IoT concept has formed a subgroup over the last couple of years, called Industrial Internet of Things (IIoT). As Boyes et al. [3] state, the application of IoT in the industrial domain is a quite disruptive action as historically, industrial automation and

control systems were largely isolated from enterprise ICT environments [3]. This strict isolation between the industrial field and other networks is more and more softened as automation and control systems are equipped with additional connectivity options that make IIoT feasible.

The initial and often non-industrial IoT application architectures often leverage the cloud as their central hub for communication and computation purposes. This direct communication of devices with the cloud introduces several challenges [4]. It adds additional latency to the overall data processing [5], is hard to implement in bandwidth-restricted environments, introduces a central point for possible failures in an otherwise distributed system [6] and requires strong data confidentiality mechanisms. Additionally, industrial applications require some application functionalities to be available in offline scenarios. Bonomi et al. argued that a new platform is needed to meet these requirements [7]. They introduced the term fog computing [8] which is not a substitute for edge-cloud computing but a powerful complement [9]. The main idea of the fog architecture is the extension of a traditional cloud computing architecture to the edge of the network. The literature proposes several definitions for fog computing and the associated IIoT-edge-cloud continuum as application field [10]–[12].

The problem space around the edge-cloud continuum is composed of but not limited to security and privacy concerns [13], [14], data management issues [13], and resource constraints [13]–[15]. Especially important for this work is the statement of Chiang et al. [13] which explicitly mention the need for a unified fog-cloud platform that amongst others should manage the life cycle of applications, as well as Atlam et al. [15] that list fog computing deployment as an open issue that needs to be solved. Both sources especially mention the actual deployment and management of applications on the continuum as a challenge. However, existing works do not provide acceptable industry-grade solutions.

B. Kubernetes Workloads and Edge-Cloud Implementations

With the advent of cloud computing and the subsequent refinement of methodology and architectures into what is today known as cloud-native, developers have been able to to react to new market requirements more rapidly [16] and allow them to generate enormous business growth and value [17]. Kratzke and Quint [18] offer a distilled set of properties that describe the characteristics of cloud-native applications [16]. One key essence in the context of this work is given in the following summary by Kratzke and Quint that defines that the application and each self-contained deployment unit of that application is designed according to cloud-focused design patterns and operated on a self-service platform [18]. Wurster et al. [16] composed the following list of properties from Kratzke's and Quint's [18] work: Service-based architectures, API-based interactions, state isolation, self-contained service deployment, disposability, fault-resilience, infrastructure abstraction, infrastructure-as-code, policy-driven elasticity, CI/CD compliance. For the context of this work, deployment and platform related properties are described in further detail, because even though the infrastructure (i.e. edge and cloud) abstraction layers have become sophisticated, developers and operators still run into situations where the underlying heterogeneity of computing resources causes problems.

- Self-contained service deployment: Each deployable service should be packaged in a standardised and self-contained format that includes the application code as well as necessary libraries and tools. A prominent method of achieving this packaging are container images that can be deployed on every compute unit that offers a compatible container engine.
- Infrastructure abstraction: The service execution and deployment should be abstracted from the underlying infrastructure. This allows applications to be portable across a variety of cloud providers, deployment targets and hardware types. Together with the self-contained service deployment, abstraction is a key enabler for flexible application placement across the edge-cloud continuum.
- CI/CD compliance: To build cloud-native applications, developers started to embrace the development and operations (DevOps) paradigm by using the concepts of continuous integration (CI) and continuous deployment (CD). Each separate service should be equipped with its own pipeline, which allows individual teams to deliver new versions of their services independent from others.

To overcome the risk of a serious vendor lock-in, several methodologies have become popular that use different tools and abstraction layers to reduce the impact of certain cloud provider specifics on the actual application and deployment architecture. Orchestration frameworks deal with the end-to-end process of a deployment and offer additional functionalities to manage related services [19]. Orchestrators offer APIs that allow users to define a desired state [20] that is afterwards implemented onto the set of resources that are assigned to the orchestrator's control domain. The API abstracts implementation details of the underlying infrastructure to a common set of resources that can be used regardless of the backing cloud provider. A state of the art orchestrator that falls into this category, also called cloud stack, is Kubernetes.

The efforts to extend Kubernetes into IIoT/edge-cloud continuums can be categorised into two groups. The first one focuses on the adaptation of the originally cloud-focused Kubernetes to be deployable in a resource-constrained edge scenario and offers users to run standalone clusters at the edge of the network. The second group is built around the concept of extending a cloud-based cluster towards edge nodes.

Representatives of the first group are K3s and MicroK8s [21]. Both distributions tuned their component resource footprint to a minimum which reduces the overhead introduced by the orchestration and leaves more of the constrained edge resources for actual user deployed workloads. In a default installation, K3s replaces the clustered storage mechanism 'etcd' [22] with the small footprint database engine SQLite. Besides this resource tuning, these distributions offer the standard Kubernetes API and run all necessary cluster components

on the edge resources themselves.

The second group extends a cloud-based control cluster towards edge nodes, with the notion of edge resources as single Kubernetes worker nodes rather than standalone and self-contained clusters. Furthermore, some of the projects in this group offer additional services and connectivity options that should solve the challenges of edge-deployed resources. KubeEdge [23] (presumably related to the original industrial research prototype KubeEdge by Xiong et al. [24]), FLEDGE [25], OpenYurt [26] and SuperEdge [27] can be mentioned as representatives of this group. However, all designs from the second group provide limited scalability due to the tight integration between edge and cloud, and only support push-based deployments, i.e. are not suitable for IIoT deployment.

III. Case Study: Water Flow Sensing

The application domain of water flow sensing spans over many different industries and businesses, e.g. municipal utilities as well as food and beverages. Especially water networks are highly distributed systems that belong to the most critical infrastructures of the modern civilised world. In order to run these networks, process automation technology is essential to ensure a continuous and faultless operation. A key factor for the automation is the continuous sensing of metrics that allow human operators or the automation infrastructure to monitor the state of the system in order to take action if necessary. Therefore, large-scale IIoT/edge-clouds emerge and need to be maintained through software deployments, i.e. Kubernetes workloads.

For better illustration of the work required to maintain a typical water network, the number of sensors and measurement locations is given based on a case study of Endress+Hauser, which is a process instrumentation supplier. The mentioned case study was conducted with the city of Oberzent in Germany that has 10'248 citizens and requires 19 decentralised sites equipped with a total of 60 sensors [28]. Hence, one edge node per 500 inhabitants is a reasonable scalability target.

Due to the mentioned criticality of the water networks, these industries are highly regulated by authorities. New regulatory requirements put a strain on existing manual processes that involve steps performed by trained valuable personnel. Some of these repetitive tasks offer the potential for further digitalisation which would free up capacity of the qualified personnel to work on their core assignments. Such repetitive tasks might comprise the manual read-out of measurements or general device validation checks that need to be carried out regularly. Current solutions introduce avoidable effort on the personnel side through time-consuming travels and work on-site of possibly remote plants.

With the advent of the IIoT concept, novel ICT architectures emerged that solve some of the problems mentioned above. In the water flow sensing domain, such architectures typically deal with the task of gathering, transforming and shipping of sensor data to other connected systems and the cloud as well as additional knowledge generation based on the processed

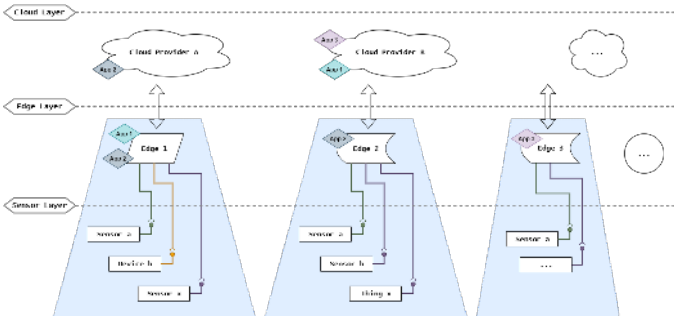

Fig. 1: Cloud-to-edge/things continuum model applicable to the water flow domain

measurements. Examples for use cases that are present in the water flow sensing domain include but are not limited to:

1) Inference of additional metrics based on the observation of existing sensor values
2) Remote validation and report generation for measurement equipment
3) Correlation of sensor metrics with environmental data (e.g. weather forecasts) to support decision processes or generate alarms

Furthermore, maintenance and operational tasks can be carried out digitally without the need for on-site interaction with the device. The infrastructure for such applications in the water flow sensing domain fits the notion of edge-cloud continuums and typically consist of three layers as depicted in Fig. 1. The domain knowledge is dispersed across teams, leading up to 10 concurrent applications per edge node. Hence, our work targets among other application fields the deployment of water sensing applications across the installed field edges and cloud resources and across concurrent applications.

IV. Deployment Mechanism – Kontinuum Controller

A. Preliminaries

This work strives to provide an industrially validated concept for a holistic application deployment and configuration management framework that is able to deliver applications across the whole edge-cloud continuum, based on Kubernetes. Applications or other software assets shall be deployable across the continuum if they are packaged and described in a format that is supported by the concept. The deployment targets in scope consist of cloud providers that offer IaaS or a higher level service offering, and constrained edge devices that are located in close proximity to the water flow sensors. A common interface for deployment and configuration management of this diverse group of deployment targets shall be defined. Through such an interface, a holistic deployment and management process shall be made possible at scale.

In the application domain of water flow sensing and the process industry in general, the operational technology is often shielded and secured from the outside world to minimise the chance of an attack or general influence of outside factors

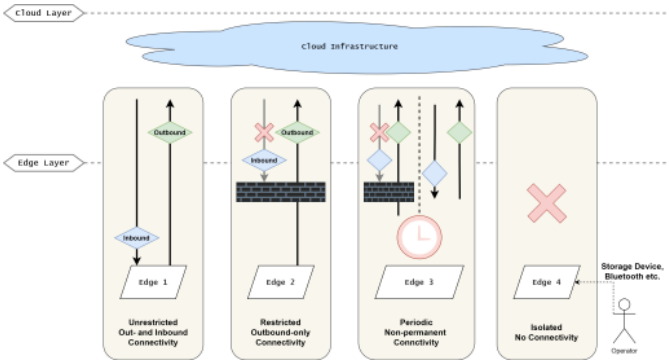

Fig. 2: Edge connectivity variants

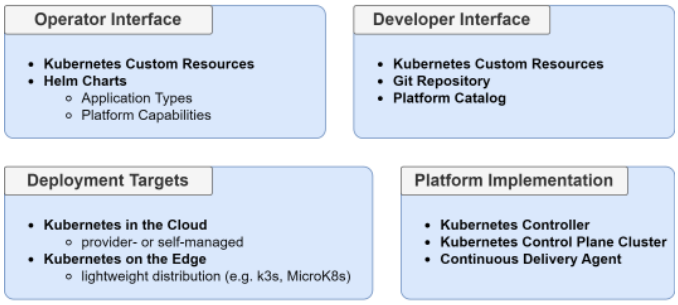

Fig. 3: Concept component overview

on the critical industrial process. Additionally, the quality of network connection can largely differ between various edge deployments. While industrial plants might be equipped with high bandwidth network connections, remote sites or standalone sensor deployments often have only access to a limited or even no connectivity option at all. These factors result in different possible network topologies that a continuum application platform has to support. Fig. 2 illustrates different variants of edge connectivity that evidently call for a pull-based deployment option.

According to our assessment, an autonomous Kubernetes controller variant offers the best fit for a platform concept on the continuum compared to other approaches such as the Open Application Model (OAM) or the construction of a platform-as-a-service (PaaS) atop Kubernetes. Due to its usage of basic Kubernetes principles, it fits optimally onto the holistic deployment and management concept that uses standalone Kubernetes clusters. The high resource consumption and limited extensibility of existing PaaS solutions make them the worst fit for deployments along the continuum. The result of the assessment supports the statements made by Pahl et al. [29] that another evolutionary step for the PaaS systems is necessary to properly support such deployment scenarios. KubeVela as a reference project falls in-between the other variants as it provides a solid foundation for platform features on the continuum but its push-only multi-cluster functionality limits its usage to only a partial set of connectivity options. However, the underlying mechanisms of the OAM are a solid baseline for a platform concept and should be (at least partially) taken into account during the fine-grade design for a pull-based controller variant and its implementation.

B. Platform components

In the first part of the implementation, a coarse-grained overview of the different components that are used in the two concept layers is given. Fig. 3 shows a graphical summary of the different components used in the design. The overview is grouped into two interface groups for the platform operators and developers that deploy applications, as well as the group of deployment targets and components used for the platform implementation. The following listing describes each group and their respective components in detail:

- Operator interface: The operator interface is based on Kubernetes custom resource objects. These Kubernetes API objects are used to interact with the platform and are processed by the Kubernetes controller that implements the platform functionality. Helm charts are the second components that platform operators interact with. With 63% (as shown by the CNCF survey 2020 [30]), Helm, at the time of writing, is the most popular packaging format for Kubernetes. For platform operators Helm charts are used for two purposes: to provide simple to use blueprints for often used application types (e.g. web applications, batch jobs etc.) that abstract Kubernetes complexity away from the users and to packages managed services onto the platform that can later be consumed by application developers.
- Developer interface: Similar as already described for the operator interface, developers also use Kubernetes custom resource objects to interact with the platform and describe their applications. The interfaces for interaction with the platform should hide platform internals and therefore a direct interaction with Kubernetes is not desirable. Git as a commonly used version control system is used to provide an interface to publish platform definitions in a way that most developers are already familiar with. Those textual definitions are then applied by a CD agent to the respective Kubernetes cluster, to minimise the direct interaction of developers with the Kubernetes API. The platform catalogue allows developers to browse available platform capabilities and applications types. The catalogue shall provide an overview of the platform operator maintained Helm charts and their detailed settings.
- Deployment targets: The deployment targets need to run Kubernetes in order to be deployable and managed by the platform. The types of clusters are grouped into cloud and edge for illustration purposes as technically no difference from a platform deployment perspective exists based on the locality of the clusters. In the cloud, the usage of managed Kubernetes services offered by cloud providers as well as the deployment to self-managed clusters is supported. Outside the boundaries of the cloud in closer proximity to the things layer often lightweight Kubernetes distributions, like K3s or MicroK8s, are typically used

due to restricted resource availability. However, any other Kubernetes variant would also be supported.

- Platform implementation: The actual platform functionality on the continuum and connection between the standalone deployment target clusters is carried out by a Kubernetes controller running on the so-called Control Plane Cluster (CPC). This cluster acts as a central hub for application descriptions and deployment assignments for all managed deployment targets. The actual platform logic is computed on the control plane cluster to save resources on the deployment targets. The aforementioned CD agent is in charge of pulling the developer definitions onto the CPC, as well as distributing the resulting Kubernetes deployment manifests to the actual target clusters.

C. Custom resource interface

Platform users and operators will use Kubernetes CRs to interact with the controller running on the control plane cluster which implements the platform logic. Since Kubernetes version 1.7, the API server has been supporting CRDs [31] that allow the expansion of the Kubernetes API with custom objects and resources. This functionality will be used to save definitions and application descriptions for the continuum-wide platform functionality on the CPC. The Kubernetes API of the CPC will be extended with the following three CRDs:

- Workload: Similar to the Application object defined in the OAM, the workload object holds a list of components that shall be deployed together. Each component references a Helm chart and includes a set of values entered by the user to configure the specific deployment details. A second list of so-called managed components describes the dependency of the respective workload to a managed and shared resource that must be deployed on the target cluster. For the assignment of the workload to one or many deployment targets, the object includes a Kubernetes label selector that is matched against the target labels.
- Target: The target object represents one Kubernetes cluster that is designated to be a deployable target on the platform. It includes a set of labels that are used for the aforementioned assignment of deployments. To deliver the resulting Kubernetes manifests to the represented target cluster, the target object includes a reference to a storage location where those deployment manifests shall be stored. Additionally, specific targets include a paused flag that will prevent the controller from shipping new deployments onto the deployment target. This functionality can support an attended update process.
- Overlay: The overlay object is a way for platform operators to specify default settings for certain components that are not included in the default Helm chart values. It can also be used to enforce certain values that should not be changeable by a platform user. The third use case is the insertion of target-specific settings for certain component types.

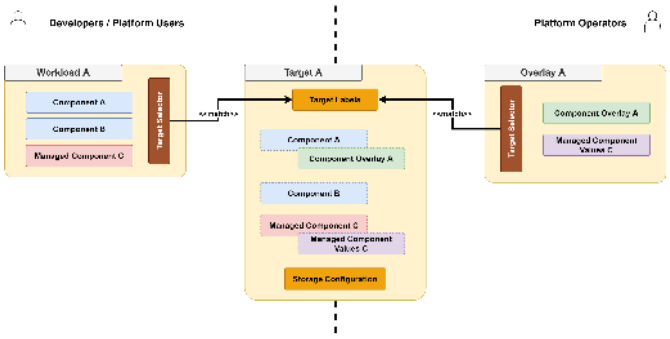

Fig. 4: Kubernetes custom resources interaction

The interaction between the three CR types is an essential part of the concept. Fig. 4 shows a graphical overview of how one set of workload, target and overlay CRs interact with each other. In this example, the workload resource contains two components with their respective user configuration. Additionally, the workload requires one platform managed component to be available on the respective deployment targets. A single overlay takes care of some specific configuration for component A and is maintained primarily by the platform operators. It also includes the platform managed configuration values for the managed component C. The managed component C object in workload A does not carry any user-specific configuration, as they are fully managed by the platform operators. Kubernetes labels and selectors [32] are a common mean in Kubernetes to build references between Kubernetes API objects. This mechanism is also used to assign workloads and overlays to one or many targets. The workload and overlay resources include a target selector that should match the labels assigned to one or more target resources. Kubernetes supports equality-based and set-based selectors. For simplicity, this concept will focus on the simpler variant of equality-based matching, where matching objects must satisfy all of the specified label constraints, though they may have additional labels as well.

D. Reconciliation process

Controllers react to changes on Kubernetes objects they watch and try to reflect the changes made to those objects in- and outside of the cluster. The process of comparing the current state with the desired state and taking the appropriate actions to achieve the desired state is called reconciliation [33]. To implement this reconciliation, the Kubebuilder SDK [34] offers a function called Reconcile for every type the controller should work on. It takes context information and the name and namespace of the object that shall be reconciled as inputs. This function needs to be implemented by the developer of the controller and therefore, besides the above-mentioned type definitions, is the main area of work to implementing the Kontinuum Controller. Kubebuilder offers configuration options to let the developer filter the objects on which the reconcile function should be called if changes are made to objects on the cluster. As each reconcile function is built for a

specific object type, Kubebuilder will by default automatically subscribe to all changes for objects of the respective type on the cluster. Watching and filtering object changes as well as queuing requests for reconciliations is abstracted away from the developer by the Kubebuilder SDK. A high-level algorithm that is implemented by the majority of reconcile functions consists of the following four steps:

1) Query the Kubernetes API to get the full object for which the reconcile request has been filed.
2) Compare the state described in the objects specification section ('spec:') with the current state on the cluster and the real world.
3) If necessary, make changes to other objects on the cluster and/or the real world to bring both closer to the desired state.
4) Update the status section ('status:') of the object under reconciliation to reflect the new state.

This control loop concept offers the possibility to implement processes and workflows on top of the Kubernetes API. The implementation is based on an asynchronous control flow, where each step is based on a change of a certain Kubernetes object. This change might be observed asynchronously by a controller running on the cluster that itself might change other objects respectively. That asynchronous chain of object changes propagates through the cluster until a steady state is reached that comes as close as possible to the desired one, described by the different objects involved. For the Kontinuum Controller, this algorithm needs to be implemented specifically for the three Kubernetes object types mentioned above: workload, target and overlay. Fig. 5 provides an overview of the interaction between the different components in the form of a sequence diagram. It focuses on the most common use case on the platform: a user changing a workload definition that is afterwards propagated through the different objects and results in an upload of updated deployment definitions to cloud storage services, for instance in a respective AWS S3 bucket.

The user configuration traverses the different Kubernetes objects and reaches the storage layer. Fig. 6 shows the case of a user editing a Workload definition. The subsequent traversal of the configuration entries through the different Kubernetes CRs is depicted. The first change of the Workload specification causes the controller to reconcile the changed object. During this reconciliation operation, as already described above, the configuration values are copied to the specification section of the associated target object. As this operation is a change to the Target object, another reconciliation is triggered. This time the changed Target object is reconciled by the controller. The second reconciliation operation processes and merges all configuration from the Workloads and Overlays that are found in the Targets specification and uploads the results to a pull storage service, for instance an S3 bucket. Additionally, the status section of the object is propagated to represent the current state of the storage layer and the associated deployment target.

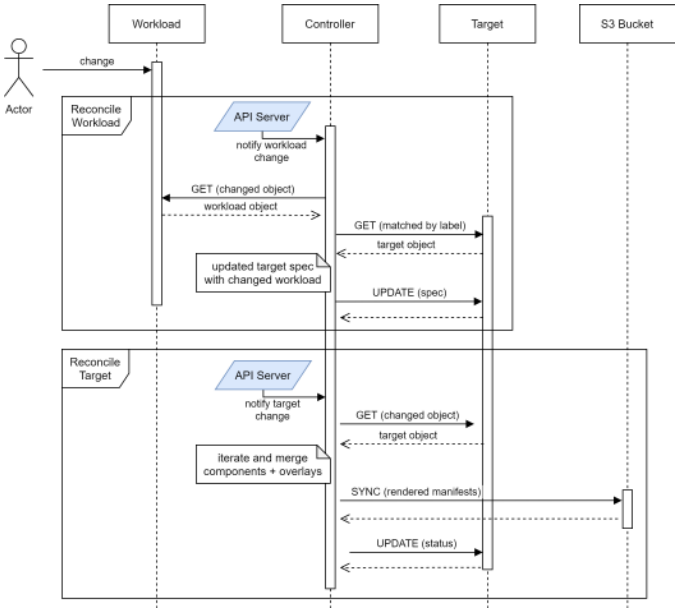

Fig. 5: Sequence diagram – Workload change

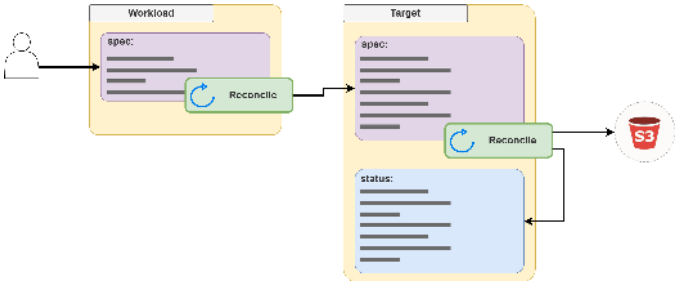

Fig. 6: Configuration flow and object traversal

V. Experimental Validation

A. Workload Deployment

To make the impact of the different platform layers on the device resource consumption visible, a multi-step testing procedure was carried out on two industrial devices, a Kunbus RevPi Connect+ with 1 GB RAM and a Wago Edge Controller with 2 GB RAM. Each edge node runs K3s and Flux to pull updated applications from S3. The four steps are 'idle' (no containers running), 'k3s startup', 'flux startup' and 'application deployment', first using NGinx as minimal deployment workload per edge, and then a more complex PostgreSQL-based water sensing application on an edge-cloud setup.

Fig. 7 shows the exemplary resource consumption on the Kunbus device. There is an evident exhaustion of memory in the NGinx deployment phase, an effect not visible with the other device, suggesting that 2 GB RAM are the minimum edge capacity for the Kontinuum Controller and any reasonable application.

In the holistic application test, a PostgreSQL-based Helm chart was deployed to Digital Ocean (via the Kubernetes API adapter Crossplane) and an edge node. The application of the workload resource on the CPC until the running container on

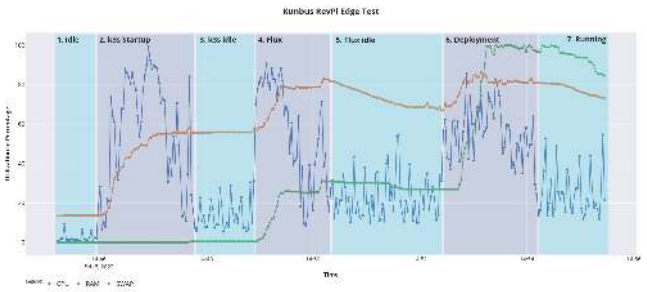

Fig. 7: Resource consumption of Kunbus RevPi

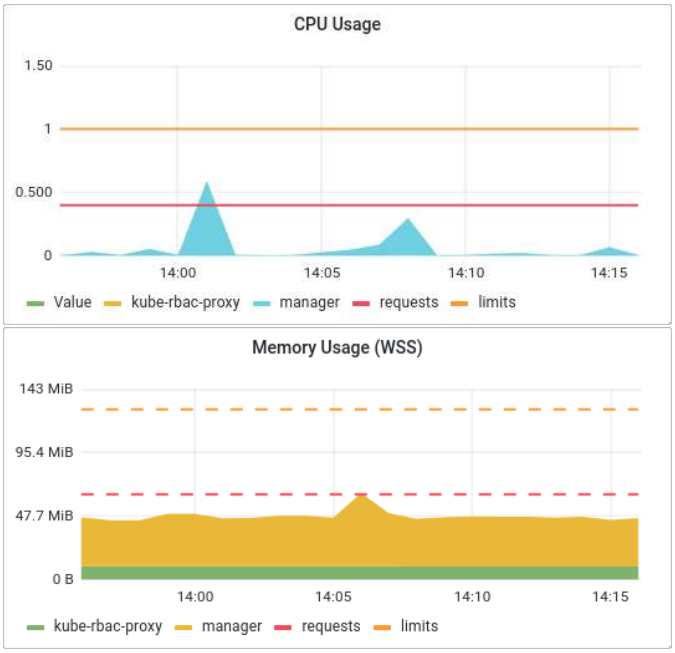

Fig. 8: Controller CPU and memory consumption

the target edge takes around one minute, which is suitable for sporadic edge connectivity during maintenance cycles. The waiting time can be attributed to I/O while the CPU load on the controller is insignificant. Only a fraction of the memory usage can be attributed to the controller itself, as evidenced by Fig. 8, measured using the B test series for reconciliation (described below).

B. Configuration Reconciliation

For the scalability tests, a large number of Kontinuum Controller CRDs needed to be applied to the cluster in order to generate the desired load. This task was automated via a script that performs the following steps with a configurable amount of objects per run: Create a configurable amount of Workload resources all assigned to the same (yet not existing) Target; Create a configurable amount of Overlay resources all assigned to the same (yet not existing) Target; Create a single Target that is referenced by the previously created resources; Delete all created resources after an interactive confirmation by the user.

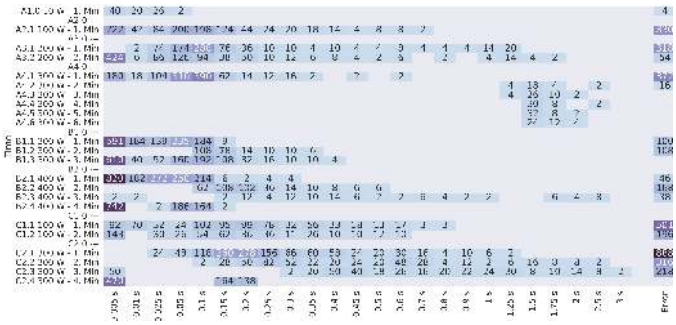

Fig. 9: Scalability heatmap – reconciliation durations

The scalability tests were performed according to the schema described above. The goal was to simulate the creation of a new Target resource on an already utilised platform control cluster. The number of assigned Workloads was chosen as the main test variable to check if the controller runs into scalability issues after a certain number of assigned objects. All tests were performed with 1 target, a set of 10 Overlays and a variable amount of Workloads. Performance metrics were gathered using Prometheus. The primary analysis is based on the Kubebuilder SDK metric 'controller_runtime_reconcile_time_seconds_bucket'. This metric provides an aggregated view of the durations each reconciliation request takes to complete due to Kubernetes' optimistic concurrency model.

Fig. 9 contains a heatmap that shows the aforementioned buckets grouped by time slots of one minute. The tests performed are named as follows and were executed with the given parameters: A tests 10–400 workloads without filtering; B tests 300–400 workloads with ignoring changes to the status object via event filtering; C tests 100–300 workloads with ignoring changes and up to three concurrent reconcile workers per CR. The results reveal that the B group achieved the best results. The expected improvement by the parallelism in the C group could not be achieved due to a high number of synchronisation-related errors (428% in C compared to B in the highest spike). The A group performance is in between, with expected issues due to status changes to configuration objects triggering another reconciliation in the absence of filtering such recursive behaviour, effectively leading to oscillation. Server-Side Apply (SSA) has been available since Kubernetes v1.22 and similarly helps avoiding oscillation, as evidenced by an early-stage kubectl-based prototype we make available along with other experiment scripts, but would require Kubebuilder SDK integration in order to be usable by the Kontinuum Controller.

VI. Discussion and Conclusion

Making large-scale industrial sensing approachable for software development and deployment is a major challenge. With the open source [1] Kontinuum Controller, we have created a framework based on Kubernetes to facilitate customised

[1]Kontinuum Controller website: https://kontinuum-controller.github.io/

deployments to heterogeneous edge-cloud environments offering computational resources for sensor data processing. It is the first approach for pull-based workload placement across edge nodes, applications and configurations. According to our findings, around 100 customised workloads per minute can be deployed on the continuum, making it a suitable approach for many industrial IoT applications with a careful controller design to avoid oscillation during reconciliation.

The work of Pahl et al. [29] describes several evolutionary steps of the wider PaaS concept towards a better fit for the edge-cloud continuum. We claim that the results of this work can be conceived as a first step towards the fourth stage as the edge-cloud PaaS, i.e. a meta-OS for the IIoT/edge-cloud continuum. Evidently, the focus of the presented concept is limited to application deployment and management. It offers potential for extensions towards additional PaaS functionality such as monitoring and automated software builds.

References

[1] T. Lynn, P. Rosati, and P. Endo, "Towards the Intelligent Internet of Everything: Observations on Multi-disciplinary Challenges in Intelligent Systems Research," 11 2018.

[2] J. Bradley, J. Barbier, and D. Handler, "Embracing the Internet of Everything To Capture Your Share of $14.4 Trillion," 2013, [6.9.2021]. [Online]. Available: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoE_Economy.pdf

[3] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial internet of things (IIoT): An analysis framework," *Computers in Industry*, vol. 101, pp. 1–12, Oct. 2018. [Online]. Available: https://doi.org/10.1016/j.compind.2018.04.015

[4] W. Masri, I. A. Ridhawi, N. Mostafa, and P. Pourghomi, "Minimizing delay in IoT systems through collaborative fog-to-fog (F2F) communication," in *2017 Ninth ICUFN*. IEEE, Jul. 2017. [Online]. Available: https://doi.org/10.1109/icufn.2017.7993950

[5] S. Mubeen, P. Nikolaidis, A. Didic, H. Pei-Breivold, K. Sandstrom, and M. Behnam, "Delay Mitigation in Offloaded Cloud Controllers in Industrial IoT," *IEEE Access*, vol. 5, pp. 4418–4430, 2017. [Online]. Available: https://doi.org/10.1109/access.2017.2682499

[6] *Leveraging Context-awareness to Better Support the IoT Cloud-Edge Continuum*. IEEE, Apr. 2020. [Online]. Available: https://doi.org/10.1109/fmec49853.2020.9144760

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12*. ACM Press, 2012. [Online]. Available: https://doi.org/10.1145/2342509.2342513

[8] F. Bonomi, "Connected vehicles, the internet of things, and fog computing," in *The eighth ACM international workshop on vehicular inter-networking (VANET), Las Vegas, USA*, 2011, pp. 13–15.

[9] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 416–464, Firstquarter 2018.

[10] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog Computing: Platform and Applications," in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*. IEEE, Nov. 2015. [Online]. Available: https://doi.org/10.1109/hotweb.2015.22

[11] L. M. Vaquero and L. Rodero-Merino, "Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, p. 27–32, Oct. 2014. [Online]. Available: https://doi.org/10.1145/2677046.2677052

[12] OpenFog Consortium Architecture Working Group, "OpenFog Reference Architecture for Fog Computing," OpenFog Consortium, Tech. Rep., 2017, [10.09.2021]. [Online]. Available: https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf

[13] M. Chiang, S. Ha, F. Risso, T. Zhang, and I. Chih-Lin, "Clarifying Fog Computing and Networking: 10 Questions and Answers," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 18–20, Apr. 2017. [Online]. Available: https://doi.org/10.1109/mcom.2017.7901470

[14] M. R. Raza, A. Varol, and N. Varol, "Cloud and Fog Computing: A Survey to the Concept and Challenges," in *2020 8th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE, Jun. 2020. [Online]. Available: https://doi.org/10.1109/isdfs49300.2020.9116360

[15] H. Atlam, R. Walters, and G. Wills, "Fog Computing and the Internet of Things: A Review," *Big Data and Cognitive Computing*, vol. 2, no. 2, p. 10, Apr. 2018. [Online]. Available: https://doi.org/10.3390/bdcc2020010

[16] M. Wurster, U. Breitenbücher, A. Brogi, F. Leymann, and J. Soldani, "Cloud-native Deploy-ability: An Analysis of Required Features of Deployment Technologies to Deploy Arbitrary Cloud-native Applications." in *CLOSER*, 2020, pp. 171–180.

[17] N. Kratzke and R. Peinl, "ClouNS - a Cloud-Native Application Reference Model for Enterprise Architects," in *20th International Enterprise Distributed Object Computing Workshop (EDOCW)*. IEEE, Sep. 2016. [Online]. Available: https://doi.org/10.1109/edocw.2016.7584353

[18] N. Kratzke and P.-C. Quint, "Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study," *Journal of Systems and Software*, vol. 126, pp. 1–16, Apr. 2017. [Online]. Available: https://doi.org/10.1016/j.jss.2017.01.001

[19] C. Surianarayanan and P. R. Chelliah, *Essentials of Cloud Computing - A Holistic Perspective*. Singapore: Springer Nature, 2019.

[20] E. Truyen, D. V. Landuyt, D. Preuveneers, B. Lagaisse, and W. Joosen, "A Comprehensive Feature Comparison Study of Open-Source Container Orchestration Frameworks," *Applied Sciences*, vol. 9, no. 5, p. 931, Mar. 2019. [Online]. Available: https://doi.org/10.3390/app9050931

[21] Rancher Labs, "K3s - Lightweight Kubernetes," 2021, [01.11.2021]. [Online]. Available: https://rancher.com/docs/k3s/latest/en/

[22] L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, and M. Kihl, "Impact of etcd deployment on Kubernetes, Istio, and application performance," *Softw. Pract. Exp.*, vol. 50, no. 10, pp. 1986–2007, 2020. [Online]. Available: https://doi.org/10.1002/spe.2885

[23] KubeEdge Project Authors, "KubeEdge Documentation - Why KubeEdge," 2021, [26.9.2021]. [Online]. Available: https://docs.kubeedge.io/en/docs/kubeedge/

[24] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend Cloud to Edge with KubeEdge," in *2018 IEEE/ACM Symposium on Edge Computing, SEC 2018, Seattle, WA, USA, October 25-27, 2018*. IEEE, 2018, pp. 373–377. [Online]. Available: https://doi.org/10.1109/SEC.2018.00048

[25] T. Goethals, F. D. Turck, and B. Volckaert, "FLEDGE: Kubernetes Compatible Container Orchestration on Low-Resource Edge Devices." Springer International Publishing, 2020, pp. 174–189. [Online]. Available: https://doi.org/10.1007/978-3-030-38651-1_16

[26] OpenYurt Contributors, "OpenYurt Documentation - Architecture," 2021, [27.12.2021]. [Online]. Available: https://openyurt.io/docs/core-concepts/architecture

[27] SuperEdge Contributors, "SuperEdge Documentation," 2021, [26.9.2021]. [Online]. Available: https://superedge.io/docs/

[28] Endress+Hauser, "Smart monitoring system for potable water treatment," 2020, [26.03.2022]. [Online]. Available: https://bdih-prod-assetcentralapi-assetcentral-rest-srv.cfapps.eu10.hana.ondemand.com/files/DLA/005056A500261EDB898273E0FAD8A4C8/CS01637X11EN.0120.pdf

[29] C. Pahl, S. Helmer, L. Miori, J. Sanin, and B. Lee, "A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters," in *4th Intl. Conf. Future IoT and Cloud Workshops (FiCloudW)*. IEEE, Aug. 2016. [Online]. Available: https://doi.org/10.1109/w-ficloud.2016.36

[30] Cloud Native Computing Foundation, "CNCF SURVEY 2020," 2020, [29.12.2021]. [Online]. Available: https://www.cncf.io/cncf-cloud-native-survey-2020

[31] The Kubernetes Authors, "Changelog 1.7," 2017, [06.01.2022]. [Online]. Available: https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.7.md

[32] ——, "Labels and Selectors," 2022, [09.01.2022]. [Online]. Available: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

[33] M. Hausenblas and S. Schimanski, *Programming Kubernetes - Developing Cloud-Native Applications*. Sebastopol: "O'Reilly Media, Inc.", 2019.

[34] The Kubernetes Authors, "Implementing a controller," 2022, [16.01.2022]. [Online]. Available: https://book.kubebuilder.io/cronjob-tutorial/controller-implementation.html