An architectural view on the Compute Continuum: challenges and technologies

G. J. Hu[a], T. Vardanega[a]

[a]*Department of Mathematics, University of Padova, Italy*

Abstract

Dramatic improvements in mobile connectivity as well as in the productivity of Web-level programming have caused the Internet of the past to turn into a ubiquitous platform where a multitude of web services are developed, deployed and employed dynamically at various locations. The resulting multitude of heterogeneous compute nodes, clusters, consumer devices, industrial sensors and actuators, all equally "connectable", compels envisioning a seamless integration of the Cloud and the Internet of Things into a single Continuum.

Although already spoken of in some vision statements, the Continuum still is a very novel notion for technology. Its concrete realisation demands adapting both Cloud-native technologies like Kubernetes and cutting-edge solutions such as WebAssembly. This paper presents a high-level reference architecture for the Continuum accompanied with a Proof of Concept implementation of an exemplary infrastructure layer for it. This effort helps gauge the distance between various state-of-the-art technologies with the proposed vision.

The findings presented in this work show that several technology solutions, organically sprouted in recent years, very well fit the Continuum, suggesting their natural convergence towards it. However, a substantial gap in terms of viability still remains, whose bridging requires serious community effort.

Keywords: Continuum of Computing, Edge Computing, Cloud Computing, WebAssembly

Email addresses: jiayi.ghu@gmail.com (G. J. Hu), tullio.vardanega@unipd.it (T. Vardanega)

1. Introduction

The Internet has evolved enormously since its inception. From just a simple communication layer for information sharing between researchers, it has grown into a ubiquitous platform for every user and any use. A flurry of organic changes to its infrastructure and interfaces has accompanied this vast transformation.

In a little over a decade of existence, the Cloud has earned web users tremendous benefits by rendering virtually unlimited quantities of infrastructure- and service-level resources available in an affordable and adaptively scalable manner.

Similarly, dramatic improvements in mobile connectivity occurred in the last two decades in terms of ubiquity, reliability, and affordability, have allowed anyone to access the Internet, and effectively the Web, from anywhere and at any time. Commercial forecasts predict that by the end of 2028, over 5 billion people, over 60/% per cent of the world population, will have a 5G coverage subscription [1], with everyday objects connected to the Internet and to each other. Such "things" comprise a growing multitude of heterogeneous devices ranging from consumer products, like mobile phones and wearables, to industrial sensors and actuators [2]. The result of this evolution is an exponential growth of Internet of Things (IoT) aggregations, beyond smart transportation systems, grids [3], and cities [4].

This arrangement intrinsically needs a decentralised, federated yet seamless organization. The Continuum is a collective infrastructure where data processing may take place dynamically where it is deemed most convenient under any of the criteria of interest (latency, privacy, energy, etc.). The concept enables the traditional Internet and the Internet of Things to integrate into a seamless Continuum, where a multitude of as-a-service applications may be developed, deployed, and employed regardless of location [5]. The Cloud and the Edge can both benefit from forming the Continuum together, allowing Cloud-like virtualised access to the physical world to occur in a more distributed and dynamic manner, and favouring the creation of numerous novel latency-free, private and secure, energy-savvy services.

This vision of seamless integration extends the literature view, which regards the Cloud and the IoT as distinct spaces, with the latter sending data and offloading computation to the former but not vice versa. Likewise, the concept goes beyond merely connecting network nodes to allow computation to happen at predetermined locations in the computing space. For instance, highly dynamic scenarios comprised of mobile users and diverse applications require flexible placement of data and computing for each mobile user. Figure 1 attempts to capture said vision pictorially.

The foundation of the Continuum is made up of pervasive service platforms located anywhere the user is, and a multitude of services, with different granularity, available over the Internet and composed opportunistically according to user needs.

Contribution. The concept of Continuum is not entirely novel, as other authors have already described a similar vision in the past [6, 5]. Other researchers,

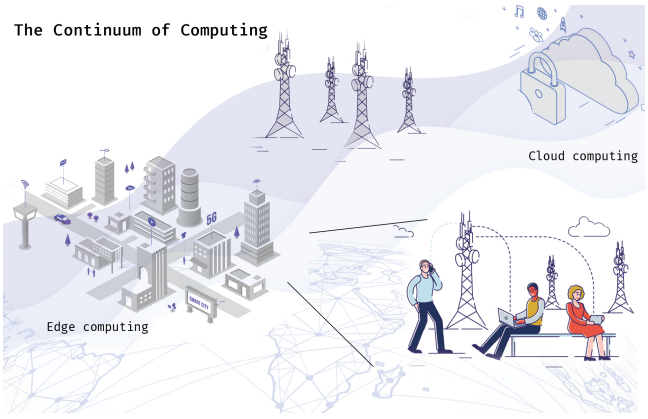

Figure 1: Pictorial view of the Continuum of Computing comprising Edge and Cloud computing. The Edge of the network is where physical reality begins, in contrast with the Cloud where all is digital.

such as [7] and [6], have also proposed analogous visions and architectures for the Continuum, although limited to the innovation of data-driven applications. Even more interestingly, the authors of [8] have argued the practicality of adopting the Serverless paradigm [9] for the Continuum. All the mentioned authors fall short in supporting a wide variety of applications, e.g. like long-running industrial control loops or even just mainstream Web app servers. Enabling the Continuum requires design-level (as oppose to merely implementation) considerations as many operational aspects of the system, for example orchestration, are highly sensitive to the specific characteristics of the hosted applications as well as of the hosting infrastructure. Our effort instead explored the use of state-of-the-art open-source technologies for building a proof of concept for the Continuum that can be application-agnostic and capable of supporting data locality and computational mobility.

In addition to the review of related works (Section 2), this paper presents a reference architecture for the Continuum (Section 3), an in-depth analysis of the problem space (Section 4), and a Proof of Concept that adapts and integrates the selected technologies (Section 5). We adapted well-known tools in the Cloud community and technologies still at an early stage of development to gauge the gap between our Continuum vision and its actual fruition. The result of our work, for concept and tools, are open-sourced and vendor-neutral, to favor further community efforts towards the Continuum.

To the best of our knowledge, the adaptation and integration of Continuum-oriented open-source tools have only been assumed to be possible [10], without exploring how far this was actually true in practice.

We first explored the feasibility of adapting mature Cloud-native tools like the Kubernetes [11] orchestrator to span over the whole Cloud-Edge Continuum. Notably, we explored the discoverability of Edge nodes in a Kubernetes cluster and the ubiquitous interoperability of web services. We also thoroughly exper-

imented with integrating the nascent sandboxing standard WebAssembly [12] to bring container-like virtualisation and portability on comparatively powerful machines and constrained devices.

Our experiments show that many of today's technologies fit rather well, in principle, our vision of the Continuum. We take this as a sign of natural convergence in the organic evolution trends of Cloud, Edge and Web. There is still, however, a substantial gap in terms of viability. On the one hand, existing production-grade tools are still figuring out how to align their Cloud-centric interface to modern use cases like Edge computing. On the other hand, nascent solutions like WebAssembly are up-and-coming on paper in terms of designs and features but still in a state of infancy in terms of practicality. (We shall return to this discussion in Section 6.)

2. Related Work

2.1. Continuum of Computing

The work in [13] provides a more comprehensive view of the trend towards integrating Cloud and IoT in a Continuum, and an articulate discussion of architecture, orchestration, privacy and business-value issues. The authors of [14] and [15] offer a similar broad literature review regarding the integration of Edge and Cloud in a Continuum. Other authors have also proposed architectures for the Continuum, but their efforts were concentrated on supporting data-driven applications [8, 6, 7]. Our work is more limited in terms of analysis of the challenges, as we restricted ourselves to those related to the presented architecture and technologies. As part of that, we propose a reference architecture and a technology selection that are missing in today's literature.

While the magnitude of data produced by the Edge is a major driving factor behind data-driven large-scale workflows, the Continuum vision should extend to a broader range of application types. Besides, while the cited research works achieve some level of integration of Edge and Cloud, they typically fail to consider the need for service composition, uniform interfaces and portable execution throughout the Continuum. Addressing these challenges is crucial to enabling pervasive applications with greater context awareness and mobility.

In passing, it should also be noted that the Continuum of Computing is recognized as an emerging paradigm by the HiPEAC (High-Performance Embedded Architecture and Compilation) network of excellence, sponsored by European Commission [16].

2.2. Osmotic Computing

Back in 2014, the authors of [17] described the concept of Fluid Internet. This novel paradigm would seamlessly provision virtualised infrastructure capabilities based on the requirements of services and users, much like a fluid adapting to fit its surroundings. In a similar chemistry analogy, a few years later, in 2016, the authors of [18] presented the vision for Osmotic Computing. Their work describes a paradigm that enables the automatic deployment

of (micro)services composed and interconnected over both edge and cloud infrastructures.

Both paradigms present strong affinities to the goals and challenges of the Continuum. Notably, the Osmotic paradigm envisions the same bidirectional flow of microservices from the Cloud to Edge - and vice versa - depending on the application configuration. The difference between Osmotic Computing and the Continuum of Computing is subtle but critical in terms of the novelty of the final applications they enable, respectively.

First, Osmotic Computing involves deploying microservices, a mere evolution of today's practice of building software in silos. Instead of running the entire application in the Cloud, Osmotic Computing decomposes it into microservices and deploys the latter across cloud and edge data centres. However, such microservices are not composed of services provided by a ubiquitous intermediary service platform. Osmotic services are thus limited in their context awareness, as services like city sensors are unavailable, decreasing business opportunities. Applications are built instead, at best, in numerous silos [19]. Indeed, the main types of microservices that the osmotic computing framework orchestrates are general-purpose microservices [18].

Second, there is a difference in semantics. Osmotic computing envisions an opportunistic *balancing* of microservices between the Cloud and the Edge, whereas the Continuum emphasises a wider *continuity* in terms of computing. Such continuity spans from the Cloud to the extreme Edge with highly constrained devices, all seamlessly integrated into the Continuum service platform. In contrast, Osmotic Computing limits itself to comparatively powerful machines such as Raspberry Pi. For such reasons, we emphasise the importance of exploring virtualisation technologies to truly include constrained IoT nodes as active players in the Continuum. Conversely, the Osmotic Computing literature focuses on more resource-demanding container-based approaches.

Third, once deployed, the Osmotic microservices are relatively stationary to the deployment location, whereas the Continuum exhibits greater extents of (potential) mobility. In case of unavailability of resources at edge/cloud, Osmotic Computing relies on solutions like message brokers (e.g. Apache Kafka) to store messages temporarily in ad-hoc queues, awaiting to resume services when resources are available [20]. Conversely, the Continuum paradigm expects the computation to migrate to the closest available location temporarily. Additionally, applications in the Continuum can move geographically to accommodate the user's movement, thanks to the seamless and ubiquitous service platform.

2.3. Serverless Computing

The Serverless paradigm [9], which focuses on the provision of computational functions, may seem to fit well with the premises of the Continuum. First, the Serverless programming model makes developing, deploying, and managing applications dramatically less burdensome than conventional styles. Second, individual functions may flexibly and equally run on the Edge or the Cloud, thus earning much portability. Furthermore, the current state of technologies we later present, like WebAssembly, plays well with the premises of Serverless,

with limited resource access. Several works from the research and industry communities are actively exploring the combination of WebAssembly and Serverless Edge functions with notable results [21, 22, 23]. Their work, combined with the definition of serverless workflows described in [8], offer an appealing proposition for the Continuum.

However, while well suited for event-driven and request-reply applications, the Serverless computing model falls short for long-running services that must feature high availability and low latency, such as industrial monitoring control loops. Provisioning and instantiating a Serverless function inevitably incur additional latency due to cold start and package download, even more in the face of unpredictable mobile user patterns and distributed networks. Other authors have also proved that it can be challenging to modify stateful applications to the Serverless paradigm, e.g. conventional web servers, since the state is not easily shared among functions [24].

Finally, the Serverless paradigm typically requires limited execution time, limited resource access and limited specialised hardware. While these restrictions allow greater scalability and mobility, they greatly reduce the scope of applications that can be deployed in them. While recent works are reducing such limitations by allowing functions to be quickly co-located in the same machine and memory regions to be safely shared using WebAssembly sandboxing [23], we deem those limitations intrinsic to the nature of the Serverless model. The Continuum and the Serverless models are not to be regarded as one form of computing supplanting the other. Analogously to how the growth of general-purpose container orchestration platforms like Kubernetes was necessary to pave the way for implementing Serverless platforms, we expect a similar direction for the Continuum and the Serverless ways. As the Edge and the Cloud become increasingly integrated, the Serverless paradigm will likely act as the dominant service delivery model *within* the Continuum.

3. A system-level view of the Continuum

3.1. Preamble

Highly distributed networks are the most effective architecture for the Continuum, particularly as services become more complex and bandwidth-hungry. Although often perceived as a single entity, the Internet is actually composed of a variety of different networks. The net result (pun intended) of such articulation is that content generated at the Edge may have to traverse multiple networks, crossing peering points before reaching its destination, at the centre of the Cloud, as depicted in Figure 2.

For the Continuum, the throughput of the entire communication path, from IoT devices to data centres back to end users, is a paramount concern. Such realisation suggests preferring processing at the Edge than causing network pressure. Offloading some compute tasks from IoT sensors or actuator nodes to the Edge is likely to be more energy-efficient. This strategy may not always be as convenient, though. Response time is the sum of two components: compute

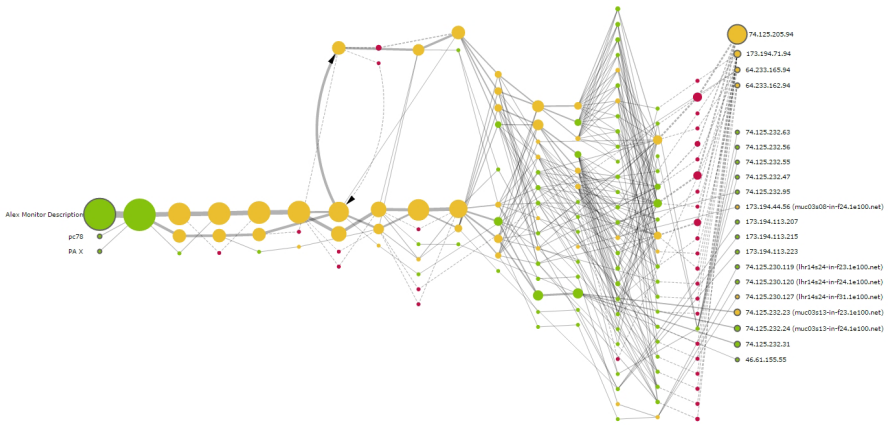

Figure 2: Traceroute virtualisation of an IP packet reaching google.com. The left green nodes are the source nodes, while packets travel across to the extreme right to servers located in data centers. Source [25].

latency and transmission latency. High compute latency can outweigh transmission efficiency. Hence, the Continuum computing is responsible for determining the preferable trade-off between the two, leveraging resources across the whole path to achieve the best optimisation on a case-by-case basis and adapting dynamically according to availability. Determining the best location for the computation to happen dynamically requires, in turn, seamless movement of data *and* computation.

3.2. Cluster federation

Figure 3 depicts the basic building blocks of the system, as we envision it to attain the sought dynamism of computation. *Cluster nodes* allow forming flexible, agile, and geographically bound aggregates, called *cluster zones*. Each such zone federates the resources collectively available within its nodes, and orchestrates their deployment.

The federation is achieved via a dedicated *infrastructure layer*, which discovers and aggregates services, data and compute resources transparently across cluster nodes in a manner that meets end-to-end QoS requirements.

As we envision it, the system dynamically instantiates and schedules services along the path from source to destination, based on application-specific requirements and constraints. If a single cluster zone lacks hardware, software or data resources to meet the user needs, it will propagate the corresponding requests outside of its federation to cluster zones within an acceptable geographical distance that have the required capabilities.

Collaboration among cluster zones is essential to support user mobility across

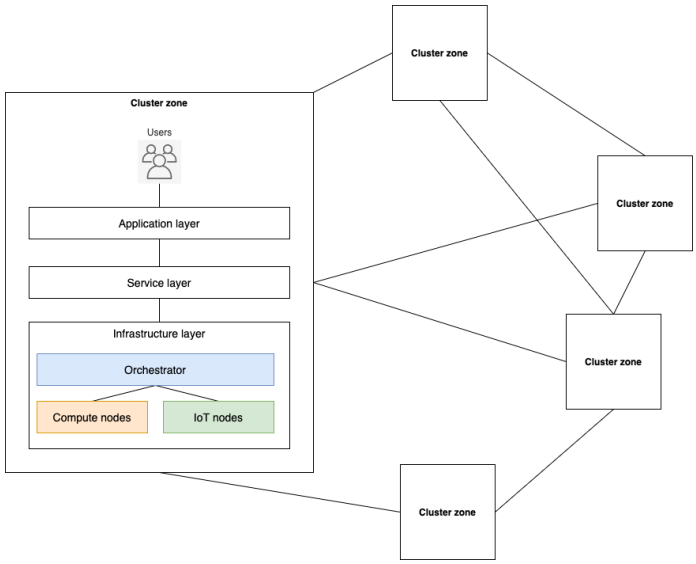

Figure 3: A high-level view of a federated set of cluster nodes.

neighbouring regions. In the Continuum, services should follow the user movements without significant outage or perturbation.

User applications running on a single cluster node are given access to requested resources thanks to the intermediation of the *service layer*. Applications intending to run on a cluster zone specify their service requirements and constraints, namely the type of resource (e.g., expected performance, pricing), without needing detailed knowledge of the underlying infrastructure. The *orchestrator* receives the requirements from the *service layer* intermediary and provisions resources and services as required, assigning them to *compute nodes* in the target cluster zone. While geographically distant, such nodes form an interconnected cluster that logically aggregates the available resources.

Services capture common dependencies like a database and persistent storage for data sources, along with pertinent constraints on them, such as latency limits and subscription plans.

We leave the federation architecture as an open research question for the future of the Continuum, owing to the comparatively early stage of maturation of our concept, and the broad and challenging scope of the topic. In the following, we limit ourselves to studying the infrastructure architecture, which is a fundamental enabler to the federation layer.

3.3. Infrastructure layer architecture

The infrastructure layer comprises a set of service providers that offer data and computation resources. The data can be generated by streaming IoT devices, for example cameras, smartwatches, and other data sources typical of

"smart things" environments. The computation resources can be heterogeneous and distributed across the infrastructure, from the Cloud to the Edge.

Figure 4 portrays the reference architecture of the Continuum infrastructure we envision, whose elements we discuss next in this section.

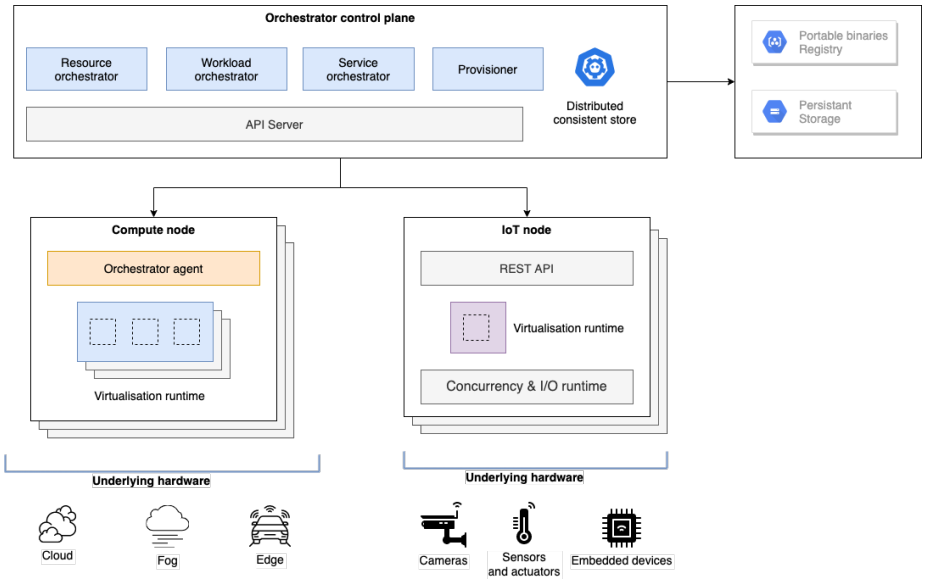

Figure 4: Reference architecture for the infrastructure.

3.3.1. Orchestrator control plane

The orchestrator control plane is the core of the orchestration system. It has a resource monitor module responsible for keeping track of real-time resource consumption metrics for each node in the compute cluster. The scheduler usually accesses this information to make better optimisation decisions. The scheduler is responsible for determining whether the Continuum has enough resources and services to execute the submitted application. If resources are insufficient, applications can be rejected or put on wait until the resources are freed. Another possible solution is to increase the number of cluster nodes to host the incoming application. Such nodes can be provisioned from local machines or anywhere in the network, preferably close to the cluster. After determining if requirements can be satisfied, the scheduler maps application components onto the cluster resources. This deployment is done by considering the application requirements, e.g. latency, geographical constraints, availability or utilisation.

3.3.2. Compute nodes

Each machine in the cluster that is available for hosting services and applications is a compute node. Each of these nodes implements the orchestrator agent runtime with various responsibilities. First, it collects local information,

such as resource consumption metrics periodically reported to the control plane. Second, it starts and stops service instances and manages local resources via a virtualisation runtime. Finally, it monitors the instances deployed on the node, sending periodic status reports to the control plane.

A central responsibility for the virtualisation runtime on the Compute nodes is to offer a consistent execution platform independent of any underlying infrastructure to allow applications to run across all software and hardware types with the same behaviour. This capability is a fundamental enabler owing to the extreme heterogeneity of the devices in the Continuum.

3.3.3. IoT nodes

IoT nodes are embedded devices that act as sensors or actuators, provided as services to the cluster (more on this to follow). The IoT nodes are heterogeneous in runtime implementation and communication protocols. Applications in the cluster interface with them via brokers provisioned by the cluster, as discussed in Section §5.3. Besides, the embedded devices support dynamic configuration by running arbitrary virtualisation modules in a lightweight runtime.

In addition to warranting interoperability among Compute nodes, the IoT runtime must also be compatible with the application format accepted by Compute nodes, when the module size and the hardware requirements can be satisfied by the target device. Such extended service interoperability enables greater flexibility and novelty in deciding where some aspects of IoT computing, such as controlling and preprocessing, happens.

Allowing arbitrary computation to run safely on microcontrollers effectively opens the embedded world to the Continuum as an additional place of intelligent computing, rather than only as a mere data collector and dummy actuator.

3.3.4. Underlying infrastructure

One of the main requirements of the infrastructure architecture is to allow deployment on a large variety of platforms. The cluster machines can be either VMs on public or private Cloud infrastructures, physical machines on a cluster, or even mobile or Edge devices, among others. Such extreme diversity requires rethinking mainstream virtualization technologies in a form that does not require the application programmer to have prior knownledge of the eventual execution contexts.

3.4. Use case: Weather-based services

As a practical example to guide the architecture's implementation, we apply the Continuum system design to weather-based services. The emergence of efficient sensing methods and IoT technologies are giving the opportunity to record and analyse possible impact of weather factors on both mainstream areas like flood warning [26] and novel fields like electrical load forecasting [27] and precision agriculture [28].

For instance, weather-relevant attributes are of great significance for electrical load forecasting and include values like temperature, air pressure, vapour

pressure, precipitation, evaporation, wind speed, and sunshine duration. Interestingly, detailed weather condition data sometimes may be captured solely by household sensors, such as the indoor temperature, sunshine duration, and indoor air quality, which differentiate in every house but have a strong effect on energy consumption. These data are also typically preprocessed to anonymise and normalise the final inputs. Gathering and preprocessing this data showcases how essential it is, for many services in the Continuum, that arbitrary code may execute safely and swiftly.

Additionally, typical parameters for precision agriculture are soil moisture content, soil temperature, surrounding temperature, humidity level, CO2 level of air, and sunlight intensity level. The sharing of weather parameters between electrical load forecasting and precision agriculture is, thus, an additional point in favour of sharing the data and computation on the Edge.

Finally, in a flood warning system, the telemetry stations acquire data (e.g. air humidity, soil moisture) from wireless sensor networks. Besides the opportunity of sharing this data with the above applications, the sensor networks also process the data in a distributed manner, and locally determine potential levee breaking. The geographical distance between the networks, the volume of data, and the relatively low interest (when no significant event is happening) make a centralised vertical solution undesired. In this case, the distributed architecture of the Continuum is a viable architecture for advanced telemetry services with distributed intelligence.

To meet the requirements of the cited types of services, we implemented a Proof of Concept system based on the architecture proposed in Figure 5:

- Sensor nodes are composed of sensor devices that collect data, preprocess it and transmit it to the Edge cluster for further processing. One challenging task of this layer is implementing the dynamic configuration of internal application-specific logic, as preprocessing is a necessary step presented in the case of electrical load forecasting;

- Broker nodes expose the sensor nodes behind a common interface. The broker subscribes to the IoT data and the device periodically sends updates, which are forwarded to the cluster. The broker ensures that both parts, IoT nodes and services nodes, are independent as far as they agree to communicate following the same API interface;

- Service nodes implement the services needed by the system, allowing them to be reused across different clusters. Internally each service can be composed of stand-alone services. We show the example of levee monitoring, which needs a streaming service to aggregate the data from the broker nodes, a local database to store the information for the analysis, and a flood prediction service to analyse the information and provide insight.

The service nodes are deployed at multiple Edge clusters, corresponding to different stations, and at a Cloud cluster.

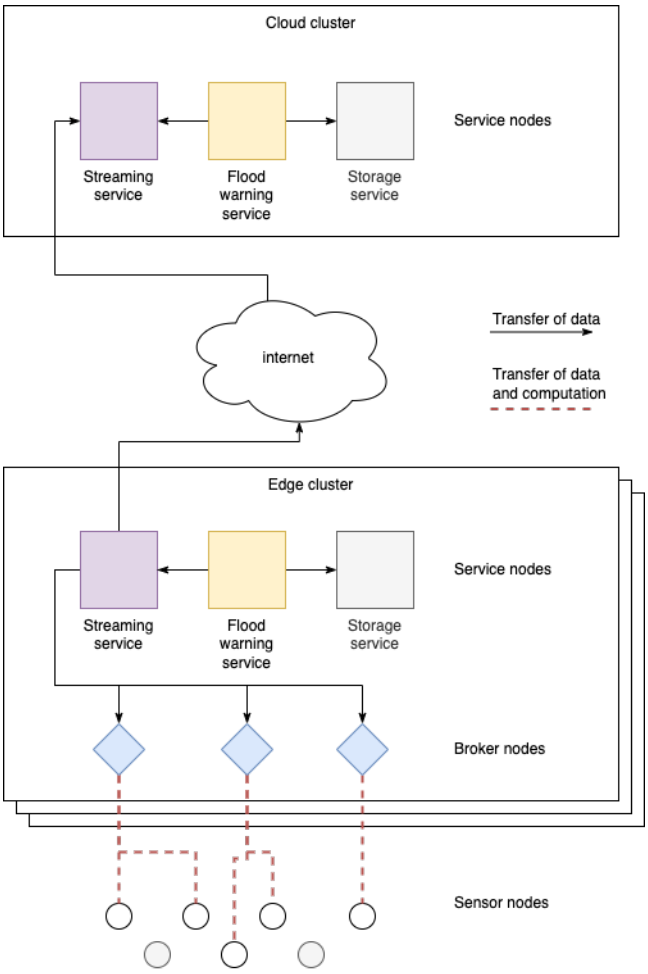

Figure 5: Architecture for the flood warning system.

The prediction model benefits from more knowledge derived from multiple streams geographically distributed. A flood risk assessment model running in the Cloud can achieve a globally optimal solution, whereas Edge services can output only locally optimal results. On the other hand, the communication channels may become unavailable during flood threat scenarios, so the system must perform a localised assessment. Unfortunately, the loss of communication is unpredictable, but the system must quickly adapt to that eventuality. These instances of demand for computing dynamism show how preferable the Continuum is in comparison to relatively static Cloud-only, Edge-only, or Cloud+Edge architectures.

Table 1: Preliminary overview of the challenges and candidate technologies. WebAssembly is a key enabling technology in our research.

Requirement / Challenge	**Candidate Technology**
Service orientation	RESTful web services Open Service Broker CoAP
Orchestration	Kubernetes, Akri
Virtualisation	WebAssembly
Dynamic configuration	WebAssembly
Interoperability	WebAssembly
Portability and Programmability	WebAssembly, Rust

4. The challenges ahead

Several challenges need to be addressed in the realisation of the Continuum infrastructure. Besides featuring extreme heterogeneity, current Edge technology most notably lacks support for service orientation, interoperability, orchestration, reliability, efficiency, availability, and security. We now briefly discuss each problem in isolation, and propose a candidate technology for each of them, which are enumerated in Table 1.

Service orientation. We argue that service orientation is fundamental to organising and utilising distributed capabilities that may lie under the control of different ownership domains.

A service-oriented model is centred around a service provider that publishes its service interface (i.e., how users may access the corresponding functionalities) via a service registry where consumers may locate it and use it to bind to the desired service provider [29].

The prime virtue of such a model is the loose coupling it earns for services, which are solely responsible for the logic and information that they encapsulate, agnostic of the composition in which they can be aggregated by higher-level providers, and placed behind well-defined interfaces and service contracts with corresponding constraints and policies. This design is in stark contrast with the dominant practice of the present day, where a multitude of ad-hoc programs are developed that are confined to single places of the network and permanently cement the behaviour of the associated devices [5].

However, major limitations have to be overcome before services can be operated seamlessly and maintained nimbly.

First and foremost, there is a lack of vendor-neutral, trustworthy and widely accepted service intermediaries. Their availability is critical to enable efficient retrieval of services that meet given user needs and warrant agreed levels of quality. Unfortunately, to date, interoperability is not dear to the main actors in the field [30].

A second critical limitation is the lack of inter-operable support for composing higher-order services from lower-level ones. Individual providers adopt their own conventions for interfaces and communication protocols: for example, Google Cloud Platform services heavily use Protocol Buffers [31], a Google technology for serialising structured data, in their service APIs. A plausible implementation of the Continuum should map high-level descriptions (e.g. key-value stores) to vendor-specific implementations.

Moreover, whereas services on the Internet of today are mute and unresponsive, future services should be communicative and reactive to their respective environments [29]. The current service interfaces in fact are ostensibly designed with human interaction in mind, thus being scarcely suited for machine-to-machine (M2M) discovery and interaction. Effective M2M communication is paramout in our vision of the Continuum, as binding a consumer to a particular service interface should entail minimal direct interaction with the provider's infrastructure.

Finally, services fit for the Continuum, hence deployable at the Edge, are sensitive to the context of the environment in which they operate. The context awareness we envision is necessary to implement local control loops and trigger specific actions on local events (e.g. sensor readings in our PoC).

Orchestration. The transition to the Continuum will require coordinating and scheduling the operation of multiple distributed service components. The complexity of that endeavour makes orchestration essential, over and above the rating it enjoys from DevOps adopters.

Orchestrating in the Continuum is especially challenging owing to the scale, heterogeneity and diversity of resource types, and the uncertainties of the underlying environments for resource capacity (e.g. bandwidth and memory), network failures, user access pattern (e.g., for quantity and location), and service life cycle [15]. Extreme heterogeneity also hinders the devising of sound pricing models that reflect account locations, resource types, transport volumes, and service latency.

Orchestrating services in the Continuum is a remarkable challenge, encompassing technologies from various fields, including wireless cellular networks, distributed systems, virtualisation, and platform management. Additionally, it requires mobility handover and service migration at local and global scales [32].

Virtualisation. The rapid pace of innovation in data centres and application platforms has transformed how organisations build, deploy, and manage services. Container-based virtualisation, owing to its natural versatility and light unitary weight, has become the dominant solution for all seekers of elastic scalability. Thousands of containers can be stored on a physical Cloud host in contrast with just very few traditional heavy-weight Virtual Machines. A natural near-future direction is an Edge-friendly containerisation that allows users to deploy services and applications on heterogeneous Edge nodes with minimal effort. Several works (e.g. [33] and [34]) argue the feasibility of container virtualisation applied to cheap low-powered devices, such as the Raspberry Pi.

Thanks to the underlying Docker image technology [35], containers provide resource isolation, self-contained packaging, anywhere-deployment, and ease of orchestration, very fitting features for the Continuum. Nonetheless, we reckon that the current state of containerisation technology still comes at too great expense in terms of memory overhead and system requirements. A typical state-of-the-art Edge runtime for containers requires at least half a Gigabyte of memory even when idle [36]. Besides, containers incur latency between hundreds of milliseconds and seconds, wholly unaffordable for latency-sensitive services that operate at the Edge. To achieve better efficiency, some platforms cache and reuse containers across multiple function calls within given time windows, typically 5 minutes. In the Edge, however, long-lived and over-provisioned containers can quickly exhaust local resource capacity, and become impractical for serving multiple IoT devices.

In the way of hard security, containers also offer weak isolation. To achieve stronger guarantees, they are often run in per-tenant VMs, too heavy for Edge-type nodes like the Raspberry Pi. A lightweight yet robust isolation solution thus is a critical step in the quest for the Continuum.

Dynamic configuration. IoT nodes must be capable of prompt reaction to context changes in the environment where they operate. Such reactions are critical to applications like video analysis [37] that are natural candidates for deployment at the extreme edge of the Continuum. Enabling dynamic configuration on constrained devices would enable swift adaptation to environmental events in accordance with application requirements. This goal can be achieved by running an application-specific computation on the node itself, earning a considerable improvement in task accuracy (owing to physical vicinity), privacy preservation, network bandwidth, and response time.

Opening devices to arbitrary code execution, though, exposes the system to malicious acts, with compromising breaches that can exploit the slightest code weakness. Current software isolation stacks like containers can hardly be used in trustworthy embedded systems as the latter typically lack the necessary storage capacity or Operating System components.

A further challenge of dynamic configuration is striking an acceptable compromise between warranting isolated execution and containing the corresponding loss of efficiency and increased energy consumption. A common memory-safe execution technique is to adopt interpreted languages that provide type and memory safety [38, 26].

Interoperability. Many technologies are available for connecting and integrating all kinds of "things" into the Continuum. ZigBee, IPv6 over Low-Power Wireless Area Networks (6LoWPAN), MQTT, and CoAP are popular in the wireless sensor networking area, while OPC has a good take-up in factory automation [39]. The fact is, though, that such technologies are too numerous and varied for any single standard to be able to accommodate all of them.

For this reason, building the Edge infrastructure of the Continuum requires coping with extreme heterogeneity, which standards will hardly be able to tame.

Best is to separate functionality from implementation, seeking interoperability in lieu of standardisation. Service-oriented architectures are ideal in this regard as they encapsulate functionality in services that can expose a common interface, abstracting away inner idiosyncrasies.

An infrastructure that allows connecting and integrating diverse technologies is not just a "necessary evil" but rather a strength that earns two key benefits. Firstly, it allows applying different solutions to different applications, in a best-fit logic. Secondly, an infrastructure where diverse technologies can easily be integrated into will be more future-resistant. Such flexibility is crucial for the Edge and IoT, which will undoubtedly see new developments for technologies and protocols. An infrastructure built with technology diversity in mind will allow interoperability with existing and already deployed devices and networks.

Portability and Programmability. In Cloud-native models of computing, users of containerisation are free to select the programming language of choice, with the sole proviso of ensuring that the corresponding executable image, which embeds all the necessary package libraries and configurations, can be deployed on the target platform. Such images can be constructed from minimal file system layers, sharing read-only parts (e.g. base OS) with other containers, thus shedding a considerable fraction of their footprint.

Conversely, in the Continuum, the compute nodes are vastly diverse for CPU (e.g., x86_64, ARM32, ARM64, and RISC-V) and runtime, making it much harder for programmers to make native application development and deployment decently portable.

Docker images (and related tools like Docker buildx [40]) attempt to overcome this challenge by defining multiple variants (usually referred to as tags) of the same image, to target multiple architectures, for processor or OS. However, this nice feature does not alleviate the pain of configuring and building each application image for all the target platforms and it is impractical to preemptively determine them in the Continuum.

5. Review of candidate technologies

5.1. Rationale for using open-source tools

In order to address the described problems in our Proof of Concept (PoC), we adapted and integrated a variety of technologies from the industry. The resulting selection includes a mixture of well-known mainstream and experimental tools still under initial development.

Using existing tools intentionally allows a realistic evaluation of the state of the art. As anticipated, what transpires from it shows an evident organic trend towards the Continuum. A further advantage of building over existing technologies is that their use is bound to facilitate acceptance and ease of development.

5.2. Service orientation

The web has become the world's most successful vendor-independent application platform and the dominant architectural style on it is Representational State Transfer (REST) [41] that makes information available as resources identified by URIs. The web is a loosely coupled architecture and applications communicate by exchanging representations of these resources using the HTTP protocol. HTTP is the most popular application protocol on the Internet and the pillar of the Web. However, new communication protocols (e.g. CoAP, which we discuss in Section 5.2) are emerging to extend the web to the Internet of Things and HTTP itself is undergoing revisions (e.g. HTTP/3 or QUIC [42]).

Our rationale for picking REST is threefold.

First, REST resources are an information abstraction that allows servers to make any information or service available, identified via Uniform Resource Identifiers (URIs). For example, this allows the sensor nodes in our PoC to act as a server and own the resource's original state. The client negotiates and accesses a representation of it. Such representation negotiation is suitable for interoperability, caching, proxying, and redirecting requests and responses. These features enable seamless inter-operation and better availability of any kind of service in the Continuum, especially IoT-involved services. Besides, under the REST architectural paradigm, IoT nodes can advertise web links to other resources creating a distributed discoverable IoT web and resulting in an even more scalable and flexible architecture.

Second, REST allows using a uniform interface across the Continuum: clients access server-controlled resources in a request-response fashion using a small set of methods with complementary semantics (GET, PUT, POST, DELETE). The requests are directed to resources that expose a generic interface with standard semantics that intermediaries can interpret. The result is an application that enables layers of transformation and indirection independent of data origin.

Third and last, REST enables high-level interoperability between RESTful protocols through proxies or, more generally, intermediaries that behave as server to a client and play as client with respect to another server. REST intermediaries fit well with the assumption that not every device must offer RESTful interfaces directly. Such flexibility suitably accommodates the diversity of communication protocols on the Edge.

We used these features to bring IoT nodes into the Continuum as any other service and to enable the coexistence of multiple equivalent services offered by different Cloud providers. We mapped provider-specific interfaces to uniform RESTful interfaces.

Open Service Broker. In our PoC, we realised a web-based service platform that implements the RESTful Open Service Broker (OSB) interface [43]. Components that implement the OSB REST endpoints are referred to as service brokers and can be hosted anywhere the application platform can reach them. Service brokers offer a catalogue of services, payment plans and user-facing metadata. The main components of the OSB architecture are depicted in Figure 6.

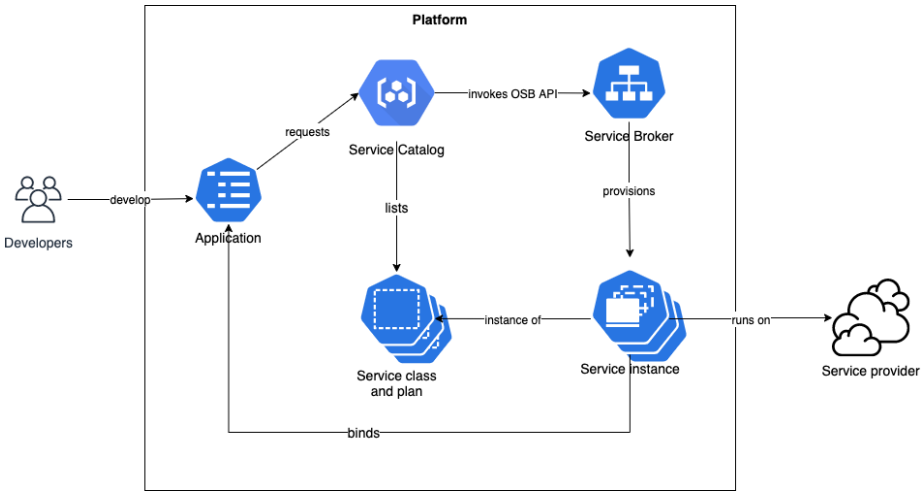

Figure 6: The Open Service Broker architecture.

In the Continuum platform, providers control access to services and payment plans but permit developers to add their own services to the catalogue. In this manner, we expect that a rich ecosystem of services may be developed over time, and tapped from simple well-documented RESTful interfaces.

As Cloud standards still struggle to gain traction, however, we need to bridge the heterogeneity gap between platforms. To this end, we used brokers to orchestrate resources at different levels within a provider. As the number of Cloud vendors is limited, building brokering layers that align access to different Clouds is an affordable endeavor. The service broker translates RESTful requests from the platform to service-specific operations such as creating, updating, deleting, and generating credentials to access the provisioned services from applications. Service brokers can offer as many services and plans as desired. Multiple service brokers can be registered with the service platform so that the final catalogue of services is the aggregate of all services. The platform is thus able to provide a rich catalogue and a consistent experience for application developers who consume these services.

Over the years, the API interface of the OSB has matured considerably, learning from the experience of a wide range of marketplace services and Cloud vendors, such as Microsoft Azure and Huawei Cloud. The current standard version 2.17 is entirely designed around asynchronously provisioned services and provides valuable guidance for challenging situations such as service failures. The OSB guidance ensures consistent semantics and interoperability across various service behaviours. Sadly though, service dependency remains a pain point that needs to be coped with, as for example in the architecture we devised for our use case (Figure 5). Currently, the OSB standard does not support a parent-child relationship model between services, whose handling is left inconveniently to the discretion of the broker author. The problems that arise from service de-

pendency include whether to publish multiple services as standalone packages and how to share credentials between services, provision and remove them in the proper order, and solve all these issues uniformly across all platforms.

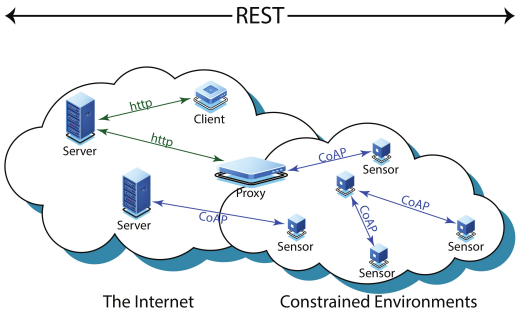

Figure 7: The REST architecture enhanced with CoAP. Source [44].

CoAP. To include IoT nodes in our REST architecture, we adopted CoAP [44], a web communication protocol for use with constrained nodes and constrained (e.g. low-power, lossy) networks. A central element of CoAP's reduced complexity compared to HTTP is that it uses the UDP transport protocol instead of TCP and defines a very simple message layer for retransmitting lost packets.

The protocol is designed for M2M applications and provides a RESTful architecture between IoT nodes, supporting built-in discovery of resources. As a result, CoAP easily interfaces with HTTP for integration with web services while meeting specialised IoT requirements such as multicast support, very low overhead and simplicity for constrained environments.

We made CoAP nodes interoperable with the rest of the Continuum by following the REST architecture's proxy pattern, as depicted in Figure 7. We built intermediaries (discussed in §5.3) that speak CoAP on one side and HTTP on the other without encoding specific application knowledge. Because equivalent methods, response codes, and options are present in HTTP and CoAP protocols, the mapping between them is straightforward. Consequently, the intermediary can discover CoAP resources and make them available at regular HTTP URIs, enabling web services in the Continuum to access CoAP servers transparently in the OSB service platform.

5.3. Orchestration

Kubernetes [11] is an open-source orchestration framework designed to manage containerised workloads on clusters, originated from Google's experience with Cloud services. Two notable features make Kubernetes especially attractive for our PoC. First, thanks to the Container Runtime Interface (CRI) API standardisation, Kubernetes allows for various container runtimes from a technical perspective, with Docker natively supported by the platform. This extensibility allowed us to leverage a uniform virtualisation platform based on

WebAssembly, while leaving the single Compute and IoT node to decide the most appropriate runtime (e.g. an interpreter compared to a Just-in-Time or Ahead-of-Time compiler).

Second, Kubernetes provides users with a wide range of options for managing their Pods (the most basic unit of deployment in Kubernetes) and how they are scheduled, even allowing for pluggable customised schedulers to be easily integrated into the system. Notably, it also supports label-based constraints for the Pods' deployment. Developers can define their labels to specify identifying attributes of objects that are meaningful and relevant to them but that do not reflect the characteristics or semantics of the system directly. More importantly, labels can also be used to force the scheduler to colocate services that communicate predominantly within the same availability zone, which improves latency very much and paves the way for context-aware services.

One more reason for our picking Kubernetes over Docker Swarm [45] was lack of multitenancy in the latter. Docker Swarm is a popular open-source orchestrator often cited for Edge orchestration (e.g. [34], and [46]) due to its simplicity. However, support for multitenancy is a must for our service platform.

Akri. To register the IoT devices on the Kubernetes cluster, we adopted Akri [47], a preliminary Microsoft open-source project which allows visibility to IoT devices from applications running within the Kubernetes cluster. Akri stretches Kubernetes' already experimental APIs to implement the discovery of IoT devices, with support for the diversity of communication protocols and ephemeral availability.

Using Akri, the Kubernetes cluster can carry out dynamic discovery to use new resources as they become available and move away from decommissioned/failed resources. Discovering IoT devices is usually accomplished by scanning all connected communication interfaces and enlisting all locally available resources.

Akri is also responsible for enabling applications to communicate with the device and deploying a broker Pod as intermediary. We devised the broker as a web server that abstracts the actual communication between devices and applications behind the RESTful API previously described.

Our RESTful broker also helps to scale the number of concurrent HTTP requests by implementing highly performant cache mechanisms. The IoT resource periodically sends its sensor readings to the broker, where the values are cached locally. Each application request is then served directly from this cache without accessing the actual device, with benefits on the average roundtrip time.

As many distributed monitoring applications are usually read-only during their operation (e.g. sensors collecting data in our case), this architecture exhibits great scalability. A potential goal is to enable new types of services where physical sensors can be shared with thousands of users with little impact on latency and data staleness. However, at the time of writing, this is still a very distant achievement.

The Kubernetes Device Plugin API heavily influences the current Akri architecture. Such interface, already considered experimental by the Kubernetes

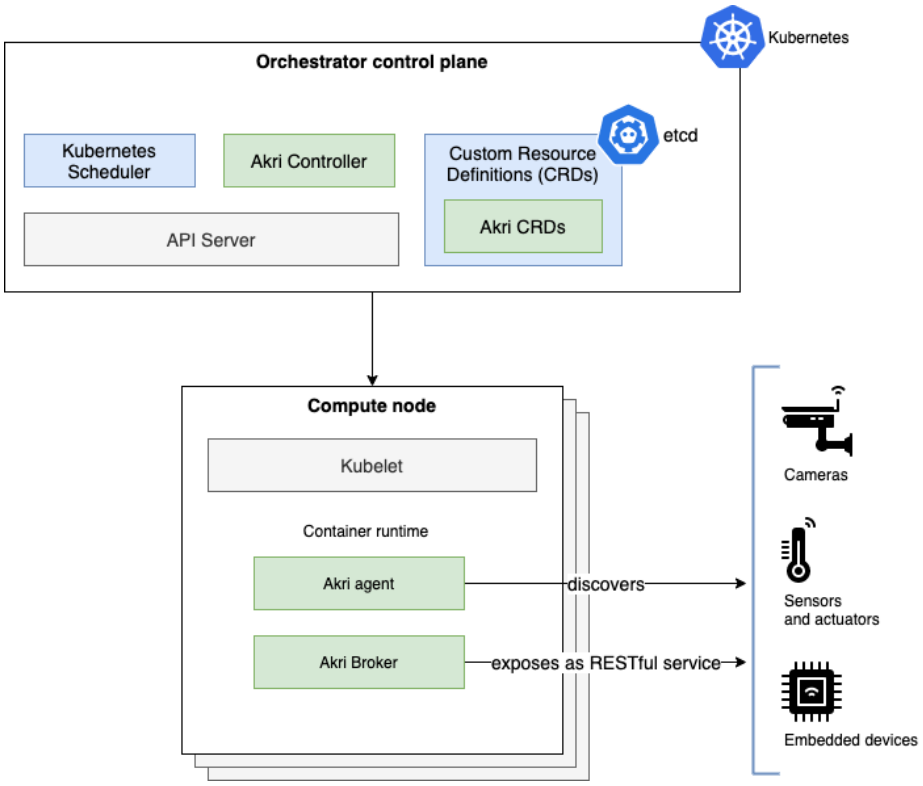

Figure 8: The Akri architecture can be divided into four main components: the agents, the controller, the brokers and the configuration. A configuration extends the Kubernetes API with new communication protocols and the related metadata, such as the protocol discovery parameters or the Docker image for the agent container. The Akri agent is a Pod responsible for discovering devices according to a communication protocol. It keeps track the device's state and communicates status updates with the Akri controller.

community, was designed for hardware attached to compute nodes, e.g. GPUs. However, IoT devices can live independently from the nodes, and most of them do. Akri expects a 1:1 relationship between compute node and device, whereas most IoT devices do not have any kind of relationship to any node per se. This mismatch has several undesired consequences, including, principally, scalability and resiliency.

Another pain point in Akri's current state is that the project lacks more advanced yet very needed features for implementing software caching or assuring high availability or autoscaling in IoT scenarios. Such features are admittedly harder to provide but highly needed to bring the Cloud to the Edge and vice versa, an essential preliminary step to the Continuum.

5.4. Virtualisation, Interoperability and Portability

WebAssembly (Wasm) [12], first announced in 2015 and released as a Minimum Viable Product in 2017, is a nascent technology that provides strong memory isolation (through sandboxing) at near-native performance with a much smaller memory footprint. WebAssembly is a language designed to address safe, fast, portable low-level code on the web. Developers who wish to leverage WebAssembly may write their code in a higher-level (compared to bytecode) language such as C++ or Rust [48] and compile it into a portable binary that runs on a stack-based virtual machine.

We picked WebAssembly as the technology enabling virtualisation, interoperability and portability in the Continuum for two fundamental reasons. First, WebAssembly provides language, hardware, and platform independency by offering a *consistent* execution platform independent of any underlying infrastructure to allow applications to run across all software and hardware types with the same behaviour. The importance of such a feature for the Continuum cannot be emphasised enough. Second, WebAssembly is advertised as safe *and* fast to execute. A program code cannot corrupt its execution environment, jump to arbitrary locations, or perform other undefined behaviour (which memory-safe languages, such as Rust, contribute to preventing). Thanks to that execution guarantee, a WebAssembly may suffer only data exploits mitigated by applying memory and state encapsulation at the module level rather than the application level. Granular memory encapsulation means that even untrusted modules can be safely executed in the same address space as other code, a critical point for dynamic configuration in constrained devices and multitenancy in the Compute nodes of our architecture. Performance wise, benchmarks of Wasm runtimes on modern browsers have shown a slowdown of approximately 10% compared to native execution, typically within 2x [12, 10].

WebAssembly is currently looked at as a candidate method for running portable applications without containers. Ideally, WebAssembly can provide significantly more lightweight isolation than VMs and containers for multi-tenant service execution. This idea is still in its infancy, but there has been consistent interest around it in recent years ([22], [21] and [23]), especially for serverless computing.

5.4.1. Dynamic configuration

Another strong point of WebAssembly is enabling arbitrary code execution on highly constrained devices across the Continuum. The authors of [49] and [50] have also explored various WebAssembly-based mechanisms for safe arbitrary execution on constrained devices and have evaluated the trade-offs between efficient Wasm processing and memory consumption. Generally speaking, Just-In-Time compilers for WebAssembly exist (e.g. Wasmtime [51]) and receive more attention from the community, but their size and complexity make them unsuitable as yet for microcontrollers.

Although WebAssembly interpreters can often run more than 10x slower than native C [52], they help dynamically update and debug system code, but are not yet mature in terms of performance and energy efficiency.

Interpreting WebAssembly on microcontrollers offers an appealing alternative to other language runtimes. The WebAssembly standard has many features that make it attractive for embedded devices [50]. First, as mentioned before, WebAssembly can be generated from different source languages and run on many CPU architectures. Furthermore, many broadly used language runtimes such as JavaScript, Lua, or Python cannot provide predictable execution. They may require excessive memory for a microcontroller, whereas Wasm requires no mandatory garbage collection and only a few runtime features around maintaining memory sandboxing. This lightweight-ness is a most valuable asset in an embedded adaptation.

Figure 9 summarises the key technologies we employed in the reference architecture of the Continuum infrastructure.

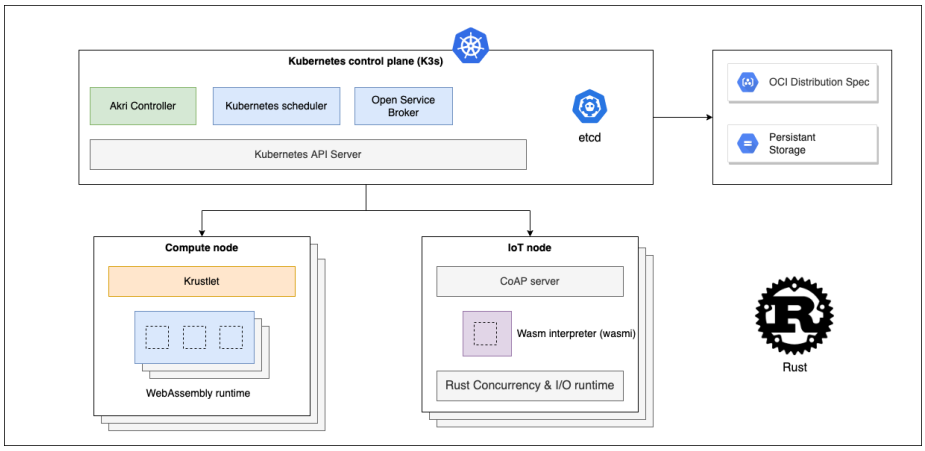

Figure 9: Technology baseline for the reference infrastructure architecture.

6. Discussing the fitness of WebAssembly for the Continuum

6.1. Rationale and devices

The above discussion has shown that WebAssembly is the central common denominator to several challenges in our quest for the Continuum, notably portability and virtualization, but also an indirect enabler of other essential characteristics of the Continuum, like computational mobility. As such, to conclude this paper, we focus our attention to an in-depth analysis of how WebAssembly fares technically in our prospective implementation.

We evaluate WebAssembly's fitness for Continuum purposes from two perspectives. First, we discuss its suitability as a portable binary format for pure computational services, showing its significantly smaller size than other mainstream alternatives unfit for highly constrained IoT devices. Similarly, we evaluate WebAssembly's fitness as an interpreter for the same embedded devices

to understand if it can guarantee safe virtualised execution while keeping the promise of reasonable performance and predictability. Lastly, we assess WebAssembly's maturity for Cloud-like capabilities. We present a cluster of Kubernetes nodes whose virtualisation runtime is based on a Just-In-Time Wasm compiler. The nodes accept applications distributed as Wasm container images under the Open Containers Initiative Artifacts specification [53]. The artifact represents the weather-based flood warning system we presented in Figure 5. We used the application to examine the maturity of the Wasm System Interface and ecosystem for a traditional web server which accepts incoming network data, saves it in a local disk-based or remote cloud-based storage and outputs the result of an inference model.

For the evaluation presented in this section, we have used the following devices:

- Edge cluster nodes: 4 Raspberry Pi 4 Model 3B+ with Quad-core Cortex-A53 (ARMv8) 64-bit SoC at 1.4GHz and 1 GB physical memory. The Raspberry 3B+ model has been chosen to showcase the feasibility of the presented technologies on limited low-powered machines, relatively cheap and with only 1GB of memory. Our results are comparable with other research regarding containerised virtualization over Raspberry Pi [34];
- Sensor nodes: STM32F407 microcontrollers with ARM Cortex-M4 core, 512KiB flash storage, and 128KiB of memory. The device is also capable of many 32-bit floating-point operations.

Raspberry Pi and STM32F407 microcontrollers are designed for moderately high computational performance, low unit cost, and power efficiency in Edge and IoT computing environments. We trust these empirical results generalise to other ARM machines and microcontrollers in the Cortex-M family.

6.2. Wasm for IoT devices

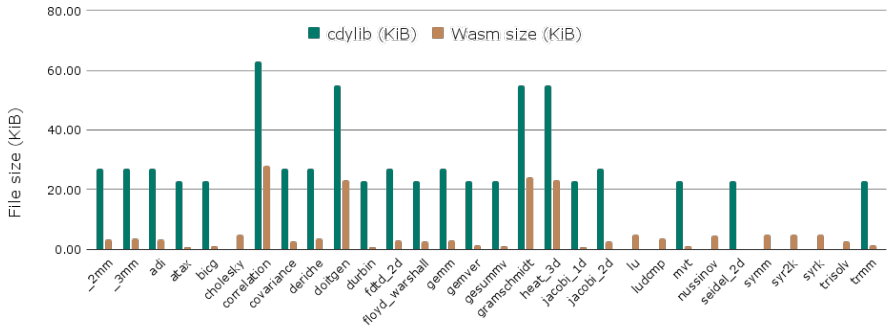

Figure 10: Comparison of Wasm size (KiB) and C dynamic library size (KiB).

Figure 10 compares the sizes of different Wasm binaries compiled from the Polybench [54] modules. The Polybench benchmark suite offers relevant functions to embedded systems as it includes standard matrix and statistical operations. We have chosen the C dynamic library size as a meaningful comparison since it is a close alternative to Wasm binary files. Both outputs have been compiled from the same Rust source code and using the same LLVM toolchain and optimisation flags.

The results undeniably favour the Wasm binary format as the C dynamic lib is often many times larger. Comparing Wasm files to containers would be even less relevant and greatly favour the former, as containers package a whole operative system filesystem, which is unnecessary for pure computational IoT services. Even the tiniest image base (Alpine Linux Mini Root Filesystem) has an additional size of about 5.5MB uncompressed.

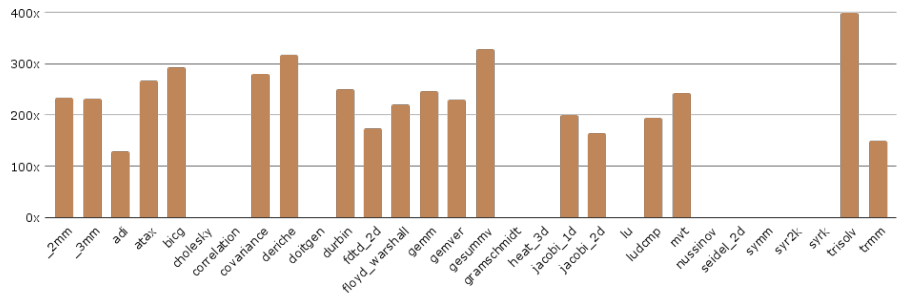

Figure 11: Comparison of Wasm interpreter performance and Rust native performance.

Figure 11 plots the slowdown of the Wasm interpreter executing Polybench benchmarks on the STM32F407 microcontroller against native Rust. The results show a dramatic slowdown, with a reduction factor of 100-400x. Such results dispel the prospect of using Wasm interpreters on microcontrollers to support dynamic reconfiguration. However, the Wasm interpreter we used, wasmi [55], was the only available Rust WebAssembly interpreter, and we adapted it to work on embedded devices. The interpreter was designed for safe execution in the blockchain instead of efficiency in highly constrained devices. Alternative embeddable interpreters, implemented in the memory-unsafe C language, show a much inferior execution penalty, in the order of 30-60x slower than native [50]. Arguably, however, a 30x execution penalty can still seriously deter the usage of interpreters in microcontrollers.

Another crucial concern we have found in our work is that the heap overhead of using Wasm interpreters is not predictable. Such unpredictability does not come again in favour of the usage of WebAssembly on microcontrollers, as embedded devices have extremely limited resources and must have predictable behaviours to ensure proper execution. Such deficiency is an intrinsic issue with interpreters, as the code instructions and execution data structures must be stored in heap memory. This behaviour contrasts with the binary executables

that can save and access instructions or read-only data on the more capable flash storage. Writable data is saved in the stack instead, and it can be estimated with accuracy in many production-grade toolchains like C and Ada.

Finally, running Wasm on resource-constrained microcontrollers also presents a memory-design issue. Wasm's pages are 64KiB by standard, too large for microcontrollers that often have between 16-256 KiB RAM. Dynamic allocation is a common requirement even for embedded systems. However, Wasm specifies that the sandbox should expand memory by 64KiB chunks, insufficiently granular for constrained embedded systems. Consequently, we had to adapt the interpreter to allocate non-standard pages of 16KiB. Otherwise, it would have been impossible to execute any benchmark on the STM32F407 microcontroller, as additional heap space is required for the interpreter's internal structures and the Wasm instructions themselves.

6.3. Wasm on the Cloud

We successfully integrated WebAssembly into Kubernetes with Wasm Pods running on Krustlet [56], while the container Pods are scheduled on K3s [57] Kubelet. Krustlet (a Kubernetes-Rust-Kubelet stack) is an experimental implementation of the Kubernetes compute node (Kubelet) API that supports Wasm as virtualisation technology. Therefore, it listens to the Kubernetes API event stream for new units of execution (Pods) and runs them under a WebAssembly System Interface (WASI) runtime (notably, Mozilla's Wasmtime [51]).

K3s is a fully certified Kubernetes distribution geared towards Edge environments backed by a commercial company. K3s is implemented in Go and packaged as a single binary of about 50MB in size.

At the time of writing, it has yet to be possible to implement a portable web server (e.g. to act as an electrical load forecasting service) and compile the application to Wasm. There is an underlying issue with implementing network servers as there is neither sufficient network API nor multi-threading support in the standard yet.

On the one hand, the current WebAssembly System Interface (WASI) standard only contains a few methods for working with sockets that are insufficient for complete networking support. Adding support for connecting to sockets is fundamental to allow Wasm modules to connect to web servers, databases, or any service. On the other hand, the lack of concurrency primitives means that a server running in WebAssembly is single-threaded, or its implementation has to be significantly more complex (e.g., like Node.js's event loop [58]). This limitation severely restricts the workload capabilities of the server. Lately, the WebAssembly specifications have outlined a thread and atomics proposal intending to speed up multi-threaded applications. At the time of this writing, however, that proposal is still in the early stage and implemented only in web browsers, behind an experimental flag.

Finally, it is also worth mentioning that the weather-based flood prediction model we compiled to Wasm is a conventional Deep Neural Network. The modal is constituted by dense layers trained on the Cloud using the traditional machine learning framework Keras [59]. As of the time of writing, TensorFlow [60]

provides a WebAssembly backend only for the browser and Node.js [61]. As a result, the model is saved to the alternative ONNX [62] standard format and executed by a WebAssembly neural network inference library (*tract* [63]) that can read and run Tensorflow or ONNX models.

To summarise the current state of WebAssembly, the technology's overall design is auspicious for the fits of the Continuum. In practice, the current standard and, subsequently, the industry implementations, severely limit the feasibility of applications that can be deployed on constrained and relatively powerful machines alike. The underlying theme is that the Wasm specifications (notably memory management, networking and concurrency) are not mature or robust enough as yet for real-world application, let alone innovative Continuum services. At the time of writing, it is still the application developer's responsibility to concretely navigate the WebAssembly landscape, leading them to create ad-hoc workarounds [64] and hindering portability.

7. Conclusion

This paper presents a Continuum of Computing constituted by software and hardware resources provided as a service and delivered anywhere the user is, as long as equipped with a host device, independently of their respective location, without requiring static configurations. To evaluate the viability of this concept for real-world industrial applications, we highlighted the principal technical challenges that the research community will have to face to make the Continuum real.

The experience reported in this paper allows us to conclude that many solutions needed by the Continuum are organically sprouting in the areas of Cloud and Edge development, which is very encouraging. Problems like IoT orchestration, service composition and multi-platform virtualisation are prominent hot challenges. Naturally, each of them has attracted interest in the last few years, and nascent solutions have emerged consequently. However, we have yet to observe any underlying governance over the evolution of these areas.

WebAssembly is a prominent representative of such issue, as Wasm-based alternatives of mainstream technologies (executables, interpreters, compilers, to name a few) spontaneously emerge and advocate for its use as a glue of the Continuum. However, its development still relies heavily on unmanaged volunteer efforts, and the progress is slow and limited. Our empirical results show that the present WebAssembly is nearly exclusively suited for pure compute functions, which is a natural fit for the Serverless but too restricting for the Continuum. The lack of multi-threading and a mature network interface severely limit the space of real-world applications that WebAssembly can serve. Likewise, the results we obtained from running WebAssembly on microcontrollers discourage the idea of using its interpreters on resource-constrained nodes. On the bright side, however, we trust these problems should be overcome given sufficient time and effort by the developers' community. On this account, we anticipate WebAssembly to receive significant attention in the coming years.

[1] Ericsson, Ericsson november 2022 mobility report, https://www.ericsson.com/en/mobility-report, 2022. Accessed: 2022-12-11.

[2] B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas, Q. Zhang, Edge computing in iot-based manufacturing, IEEE Communications Magazine 56 (2018) 103–109.

[3] I. Mugarza, A. Amurrio, E. Azketa, E. Jacob, Dynamic software updates to enhance security and privacy in high availability energy management applications in smart cities, IEEE Access 7 (2019) 42269–42279.

[4] F. H. Cabrini, F. Valiante Filho, P. Rito, A. Barros Filho, S. Sargento, A. Venâncio Neto, S. T. Kofuji, Enabling the industrial internet of things to cloud continuum in a real city environment, Sensors 21 (2021) 7707.

[5] P. Beckman, J. Dongarra, N. Ferrier, G. Fox, T. Moore, D. Reed, M. Beck, Harnessing the computing continuum for programming our world, Fog Computing: Theory and Practice (2020) 215–230.

[6] D. Balouek-Thomert, E. G. Renart, A. R. Zamani, A. Simonet, M. Parashar, Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows, The International Journal of High Performance Computing Applications 33 (2019) 1159–1174.

[7] M. AbdelBaky, M. Zou, A. R. Zamani, E. Renart, J. Diaz-Montes, M. Parashar, Computing in the continuum: Combining pervasive devices and services to support data-driven applications, in: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), IEEE, pp. 1815–1824.

[8] S. Risco, G. Moltó, D. M. Naranjo, I. Blanquer, Serverless workflows for containerised applications in the cloud continuum, Journal of Grid Computing 19 (2021) 1–18.

[9] H. Shafiei, A. Khonsari, P. Mousavi, Serverless computing: a survey of opportunities, challenges, and applications, ACM Computing Surveys 54 (2022) 1–32.

[10] J. Ménétrey, M. Pasin, P. Felber, V. Schiavoni, Webassembly as a common layer for the cloud-edge continuum, in: Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge, FRAME '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 3–8.

[11] The Linux Foundation, Kubernetes, https://kubernetes.io/, 2021. Accessed: 2022-12-11.

[12] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, J. Bastien, Bringing the web up to speed with webassembly, in: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 185–200.

[13] T. Lynn, J. G. Mooney, B. Lee, P. T. Endo, The cloud-to-thing continuum: opportunities and challenges in cloud, fog and edge computing, Springer Nature, 2020.

[14] J. Santos, T. Wauters, B. Volckaert, F. De Turck, Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and research directions, IEEE Communications Surveys & Tutorials 23 (2021) 2557–2589.

[15] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, O. Rana, The internet of things, fog and cloud continuum: Integration and challenges, Internet of Things 3-4 (2018) 134–155.

[16] HiPEAC, Vision 2021, https://www.hipeac.net/vision/#/latest/articles/?q=continuum, 2021. Accessed: 2022-12-11.

[17] S. Latre, J. Famaey, F. De Turck, P. Demeester, The fluid internet: service-centric management of a virtualized future internet, IEEE Communications Magazine 52 (2014) 140–148.

[18] M. Villari, M. Fazio, S. Dustdar, O. Rana, R. Ranjan, Osmotic computing: A new paradigm for edge/cloud integration, IEEE Cloud Computing 3 (2016) 76–83.

[19] A. Camero, E. Alba, Smart city and information technology: A review, Cities 93 (2019) 84–94.

[20] B. Neha, S. K. Panda, P. K. Sahu, K. S. Sahoo, A. H. Gandomi, A systematic review on osmotic computing, ACM Transactions on Internet of Things 3 (2022) 1–30.

[21] P. K. Gadepalli, S. McBride, G. Peach, L. Cherkasova, G. Parmer, Sledge: a serverless-first, light-weight wasm runtime for the edge, in: Proceedings of the 21st International Middleware Conference, pp. 265–279.

[22] A. Hall, U. Ramachandran, An execution model for serverless functions at the edge, in: Proceedings of the International Conference on Internet of Things Design and Implementation, pp. 225–236.

[23] S. Shillaker, P. Pietzuch, Faasm: lightweight isolation for efficient stateful serverless computing, in: 2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20), pp. 419–433.

[24] M. Malawski, A. Gajek, A. Zima, B. Balis, K. Figiela, Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions, Future Generation Computer Systems 110 (2020) 502–514.

[25] dotcom monitor, Visual traceroute, https://www.dotcom-monitor.com/wiki/knowledge-base/visual-traceroute-graphical-tool/, 2022. Accessed: 2022-12-11.

[26] R. Brzoza-Woch, M. Konieczny, P. Nawrocki, T. Szydlo, K. Zielinski, Embedded systems in the application of fog computing—levee monitoring use case, in: 2016 11th IEEE Symposium on Industrial Embedded Systems (SIES), IEEE, pp. 1–6.

[27] L. Li, K. Ota, M. Dong, When weather matters: Iot-based electrical load forecasting for smart grid, IEEE Communications Magazine 55 (2017) 46–51.

[28] B. Keswani, A. G. Mohapatra, A. Mohanty, A. Khanna, J. J. Rodrigues, D. Gupta, V. H. C. De Albuquerque, Adapting weather conditions based iot enabled smart irrigation technique in precision agriculture mechanisms, Neural Computing and Applications 31 (2019) 277–292.

[29] S. Haller, S. Karnouskos, C. Schroth, The internet of things in an enterprise context, in: Future Internet Symposium, Springer, pp. 14–28.

[30] N. Grozev, R. Buyya, Inter-cloud architectures and application brokering: taxonomy and survey, Software: Practice and Experience 44 (2014) 369–390.

[31] Google, Protocol buffers, https://developers.google.com/protocol-buffers, 2021. Accessed: 2022-12-11.

[32] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, M. Parashar, Mobility-aware application scheduling in fog computing, IEEE Cloud Computing 4 (2017) 26 – 35. Cited by: 268; All Open Access, Green Open Access.

[33] C. Pahl, S. Helmer, L. Miori, J. Sanin, B. Lee, A container-based edge cloud paas architecture based on raspberry pi clusters, in: 2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW), IEEE, pp. 117–124.

[34] P. Bellavista, A. Zanni, Feasibility of fog computing deployment based on docker containerization over raspberrypi, in: Proceedings of the 18th international conference on distributed computing and networking, pp. 1–10.

[35] Docker, Docker image specification 1.0.0, https://github.com/moby/moby/blob/master/image/spec/v1.md, 2021. Accessed: 2022-12-11.

[36] S. Böhm, G. Wirtz, Profiling lightweight container platforms: Microk8s and k3s in comparison to kubernetes., in: ZEUS, pp. 65–73.

[37] S. Y. Jang, Y. Lee, B. Shin, D. Lee, Application-aware iot camera virtualization for video analytics edge computing, in: 2018 IEEE/ACM Symposium on Edge Computing (SEC), IEEE, pp. 132–144.

[38] M. Jacobsson, J. Willén, Virtual machine execution for wearables based on webassembly, in: EAI International Conference on Body Area Networks, Springer, pp. 381–389.

[39] E. Al-Masri, K. R. Kalyanam, J. Batts, J. Kim, S. Singh, T. Vo, C. Yan, Investigating messaging protocols for the internet of things (iot), IEEE Access 8 (2020) 94880–94911.

[40] Docker, buildx, https://github.com/docker/buildx, 2022. Accessed: 2022-12-11.

[41] R. Fielding, Representational state transfer (rest), https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000. Accessed: 2022-12-11.

[42] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al., The quic transport protocol: Design and internet-scale deployment, in: Proceedings of the conference of the ACM special interest group on data communication, pp. 183–196.

[43] Cloud Foundry, Open service broker, https://www.openservicebrokerapi.org/, 2016. Accessed: 2022-12-11.

[44] C. Bormann, A. P. Castellani, Z. Shelby, Coap: An application protocol for billions of tiny internet nodes, IEEE Internet Computing 16 (2012) 62–67.

[45] Docker, Swarm mode overview, https://docs.docker.com/engine/swarm/, 2021. Accessed: 2022-12-11.

[46] B. I. Ismail, E. M. Goortani, M. B. Ab Karim, W. M. Tat, S. Setapa, J. Y. Luke, O. H. Hoe, Evaluation of docker as edge computing platform, in: 2015 IEEE Conference on Open Systems (ICOS), IEEE, pp. 130–135.

[47] Deis Labs, Akri, https://github.com/deislabs/akri, 2021. Accessed: 2022-12-11.

[48] AWS, Why aws loves rust and and how we'd like to help, https://aws.amazon.com/blogs/opensource/why-aws-loves-rust-and-how-wed-like-to-help/, 2020. Accessed: 2022-12-11.

[49] M. Jacobsson, J. Willén, Virtual machine execution for wearables based on webassembly, in: C. Sugimoto, H. Farhadi, M. Hämäläinen (Eds.), 13th EAI International Conference on Body Area Networks, Springer International Publishing, Cham, 2020, pp. 381–389.

[50] G. Peach, R. Pan, Z. Wu, G. Parmer, C. Haster, L. Cherkasova, ewasm: Practical software fault isolation for reliable embedded devices, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39 (2020) 3492–3505.

[51] Bytecode Alliance, wasmtime, https://github.com/bytecodealliance/wasmtime, 2021. Accessed: 2022-12-11.

[52] wasm3, wasm3 performance, https://github.com/wasm3/wasm3/blob/main/docs/Performance.md, 2021. Accessed: 2022-12-11.

[53] OCI, Artifacts, https://github.com/opencontainers/artifacts/blob/main/artifact-authors.md, 2022. Accessed: 2022-12-11.

[54] T. Yuki, Understanding polybench/c 3.2 kernels, in: International workshop on Polyhedral Compilation Techniques (IMPACT), pp. 1–5.

[55] Parity, wasmi, https://github.com/paritytech/wasmi, 2021. Accessed: 2022-12-11.

[56] Deis Labs, Krustlet, https://github.com/deislabs/krustlet, 2021. Accessed: 2022-12-11.

[57] Rancher, k3s, https://k3s.io/, 2021. Accessed: 2022-12-11.

[58] Node.js, The node.js event loop, https://nodejs.dev/learn/the-nodejs-event-loop, 2021. Accessed: 2022-12-11.

[59] Keras, Keras, https://keras.io/, 2021. Accessed: 2022-12-11.

[60] Google, Tensorflow, https://www.tensorflow.org/, 2021. Accessed: 2022-12-11.

[61] Tensorflow, Introducing the webassembly backend for tensorflow.js, https://blog.tensorflow.org/2020/03/introducing-webassembly-backend-for-tensorflow-js.html, 2020. Accessed: 2022-12-11.

[62] T. L. Foundation, Onnx, https://onnx.ai/, 2021. Accessed: 2022-12-11.

[63] I. Sonos, tract, https://github.com/sonos/tract, 2021. Accessed: 2022-12-11.

[64] Deislabs, wasi-experimental-http, https://github.com/deislabs/wasi-experimental-http, 2022. Accessed: 2022-12-11.