Continuum: Automate Infrastructure Deployment and Benchmarking in the Compute Continuum

Matthijs Jansen
VU Amsterdam
The Netherlands
m.s.jansen@vu.nl

Linus Wagner
VU Amsterdam
The Netherlands
l3.wagner@student.vu.nl

Animesh Trivedi
VU Amsterdam
The Netherlands
a.trivedi@vu.nl

Alexandru Iosup
VU Amsterdam
The Netherlands
a.iosup@vu.nl

ABSTRACT

As the next generation of diverse workloads like autonomous driving and augmented/virtual reality evolves, computation is shifting from cloud-based services to the edge, leading to the emergence of a cloud-edge *compute continuum*. This continuum promises a wide spectrum of deployment opportunities for workloads that can leverage the strengths of cloud (scalable infrastructure, high reliability), edge (energy efficient, low latencies), and endpoints (sensing, user-owned). Designing and deploying software in the continuum is complex because of the variety of available hardware, each with unique properties and trade-offs. In practice, developers have limited access to these resources, limiting their ability to create software deployments. To simplify research and development in the compute continuum, in this paper, we propose *Continuum*, a framework for automated infrastructure deployment and benchmarking that helps researchers and engineers to deploy and test their use cases in a few lines of code. Continuum can automatically deploy a wide variety of emulated infrastructures and networks locally and in the cloud, install software for operating services and resource managers, and deploy and benchmark applications for users with diverse configuration options. In our evaluation, we show how our design covers these requirements, allowing Continuum to be (i) highly flexible, supporting any computing model, (ii) highly configurable, allowing users to alter framework components using an intuitive API, and (iii) highly extendable, allowing users to add support for more infrastructure, applications, and more. Continuum is available at https://github.com/atlarge-research/continuum.

CCS CONCEPTS

• **Computer systems organization → n-tier architectures**; • **Computing methodologies → Distributed computing methodologies**.

KEYWORDS

Compute continuum, infrastructure deployment, resource management, benchmark, task offloading, software development

ACM Reference Format:
Matthijs Jansen, Linus Wagner, Animesh Trivedi, and Alexandru Iosup. 2023. Continuum: Automate Infrastructure Deployment and Benchmarking in the Compute Continuum. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3578245.3584936

ICPE '23 Companion, April 15–19, 2023, Coimbra, Portugal
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0072-9/23/04.
https://doi.org/10.1145/3578245.3584936

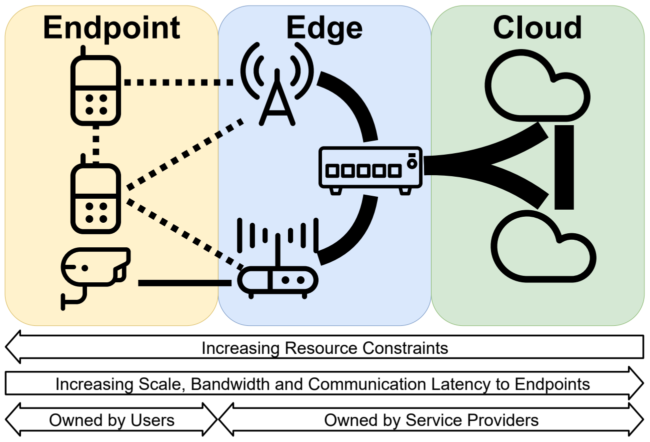
Endpoint
Edge
Cloud
Increasing Resource Constraints
Increasing Scale, Bandwidth and Communication Latency to Endpoints
Owned by Users
Owned by Service Providers

Figure 1: The compute continuum with different key properties for cloud, edge, and endpoint devices and networks.

1 INTRODUCTION

The compute continuum contains an unprecedented variety of heterogeneous cloud, edge, and endpoint devices, as well as networks connecting them. These three tiers of devices represent a trade-off in resource capacity, scale, network latency, privacy, energy, and cost (Figure 1). The cloud offers large-scale infrastructure and a wide array of computing, storage, and resource management services through providers such as AWS and GCP. Meanwhile, the edge brings compute resources closer to users on a smaller scale with less powerful devices but with lower access latencies and increased privacy. Finally, endpoint devices such as sensors and smart devices are heavily resource constrained and located at the far end of the network. They are operated by users and can offload workload to the cloud or edge to leverage their resources and services. By including cloud, edge, and endpoint, the continuum spawns a large array of use cases, including, but not limited to, content delivery networks, self-driving vehicles, and IoT [6].

Designing and deploying software for a specific compute continuum use case is very complex because of the variety of available networks and resources, each with unique properties. Additionally, service providers need to adapt their services to support many use cases across the continuum, requiring new solutions. Developers need access to all these hardware resources to consider all available deployment models, such as cloud computing, edge computing, or fog computing [17]. Each deployment (or computing) model presents its guidelines on how to deploy software across cloud, edge, and endpoints, and has specific assumptions and requirements on device and network properties, data flow, multi-tenancy, privacy requirements, and more. In practice, developers have limited access to hardware and are therefore restricted in the deployment models they can consider, resulting in possible suboptimal deployments. Furthermore, deployment models also restrict what software can be used, such as operating systems and services, resource managers, and application back-ends, influencing and further complicating software deployments (Figure 2). This raises the question of *how developers should design, test, and deploy software in the complex environment of the compute continuum.*

In this paper, we propose *Continuum*, a framework that combines automated infrastructure deployment with application benchmarking in the compute continuum to help researchers and engineers deploy and test their use cases with a few lines of code. Continuum can automatically deploy a wide variety of cloud, edge, and endpoint infrastructures and networks, install software for operating services and resource managers, and benchmark applications for users with a collection of configuration options. To guarantee that Continuum covers all concerns of developers wanting to create software for cloud, edge, or endpoint, we analyze the SPEC-RG reference architecture for the compute continuum [4] in Figure 2 and synthesize a list of key requirements. We show how our design covers these requirements, allowing Continuum to be (i) highly flexible, supporting any computing model, (ii) highly configurable, allowing users to alter each framework component using an intuitive API, and (iii) highly extendable, allowing users to add support for more infrastructure, resource managers, applications, and more.

Continuum uses virtual machines (VM) to emulate infrastructure and currently supports infrastructure deployment on local hardware using QEMU and in the cloud using Google Cloud Platform (GCP). We show how (i) Continuum's virtualized infrastructure performs similarly to bare-metal deployments while (ii) providing increased flexibility by allowing users to change per VM CPU, memory, storage, and network resources, and (iii) allows users to emulate any device on general-purpose hardware. Continuum can automatically set up advanced software stacks on the provided virtualized infrastructure, such as distributed resource management deployments with Kubernetes and KubeEdge [16] or serverless deployments with OpenFaaS [2], and allows users to switch between them on the fly. Finally, we demonstrate how Continuum combines its unique hardware and software provisioning tools with application benchmarking functionalities using a machine learning use case and show how the framework provides key metrics to help users make more informed decisions.

Our key contributions in this paper include:

(1) We synthesize a list of key requirements and concerns for software development in the continuum by analyzing the SPEC-RG compute continuum reference architecture (§2).
(2) We present the design of Continuum, a framework for automated infrastructure, software, and application deployment in the compute continuum (§3). We show how this design allows users to freely explore the compute continuum design space using a simple but expressive API, and switch between deployments in a few lines of code.
(3) We demonstrate how Continuum virtualizes infrastructure with QEMU and Google Cloud for near-bare-metal performance while allowing users to benchmark cloud, edge, and endpoint deployments using advanced containerized and serverless resource management software such as Kubernetes and OpenFaaS, helping users make informed decisions on how, where, and when to offload workload (§4).

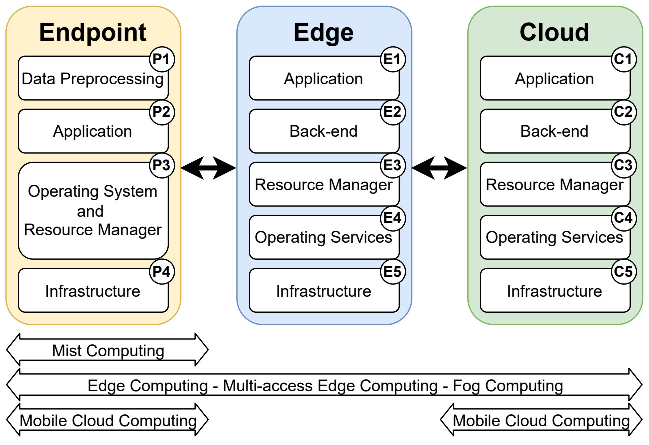

Figure 2: The SPEC-RG reference architecture for the compute continuum [4].

2 MOTIVATION FOR A CONTINUUM DEVELOPMENT FRAMEWORK

The compute continuum spawns a large design space with various complex trade-offs that are difficult to navigate, even for those with expert knowledge. Therefore, new workloads need to be tested on various hardware (e.g., ARM or x86 devices) and software configurations (e.g., different operating services and resource managers) to find an optimal deployment, which costs a significant amount of time and money. Consequently, there is an important opportunity to create a tool that helps developers quickly iterate through deployment configurations without advanced hardware requirements to understand the available trade-offs better. In this section, we present the motivation for such a tool.

2.1 Requirement Selection

We analyze the SPEC-RG reference architecture for the compute continuum, as shown in Figure 2 [4]. This architecture is unique because it lists common components in continuum deployment models, such as cloud or edge computing, each with a distinct set of requirements. These requirements are explicit when developers create software for these components or implicit when developers rely on other components (e.g., deploy an application via a resource manager). Both requirement sets should be considered during development as all architecture components rely on each other to function. We analyze the components and their requirements as described in [4] and split the components up into three categories, being hardware (P4, E5, C5), applications (P1-2, E1, C1), and the multiple software layers in between that support executing applications on hardware in a distributed environment (P3, E2-4, C2-4). We explain these categories in detail below, list their requirements in Table 1, and argue that these requirements are mandatory when designing a framework that allows users to navigate all infrastructure deployment, software installation, and application benchmarking steps needed to develop software for the continuum.

Table 1: Required components for a continuum development framework, derived from the reference architecture (RA).

ID	Requirement	RA Component
R1	Cloud, edge, and endpoint resources	P4, E5, C5
R2	Network resources	
R3	Configurable resources	
R4	Flexible deployment options	
R5	Automated software stack installation	P3, E2-4, C2-4
R6	Configurable software stack	
R7	Automated application deployment	P1-2, E1, C1
R8	Advanced observability	
R9	Accurate deployments	-
R10	Performant framework	
R11	Extendable components	

2.2 Infrastructure Requirements

The first set of requirements relates to the presence and configuration of continuum hardware and networks. Developers may need to explore what infrastructure works best for their use case or need to emulate a particular environment in the absence of specific physical hardware.

R1: Cloud, edge, and endpoint resources Users should be able to use cloud (C5 in Figure 2), edge (E5), and endpoint devices (P4), either through physical or virtual hardware. Both hardware types have important advantages: Physical hardware guarantees correct performance and behavior similar to bare-metal production environments, while virtual resources such as VMs offer access to resources without needing corresponding physical resources (e.g., emulate ARM devices on x86 hardware).

R2: Network resources Devices in the continuum are connected through various types of networks, leveraging Ethernet, Bluetooth, WiFi, and cellular technologies. These networks vary in communication latency, throughput, and packet drop rate, and are critical to consider when designing software [1]. The envisioned tool should recreate these networks with various properties, even in the absence of physical networking infrastructure.

R3: Configurable resources Devices in the continuum vary significantly in compute, memory, storage, and network capacity. Therefore, the tool should allow users to configure these resources on a per-device basis to create deployments based on computing models such as fog or mist computing [11].

R4: Flexible deployment options Today, developers can choose between a vast amount of infrastructure to buy or use via a cloud service provider. For developers to reuse existing infrastructure to save time and money, or test deployment in a production-like environment, the framework must support different hardware and cloud providers such as AWS or Google Cloud Platform.

2.3 Software Requirements

Software and middleware such as resource managers like Kubernetes or operating services like Apache Kafka (P3, E2-4, C2-4) are vital for software deployment in the continuum as they help users manage highly distributed infrastructure and tackle concerns like scalability, availability, and consistency.

R5: Automated software stack installation Not only are distributed environments challenging to operate, but distributed software is also complex to set up. Consequently, our framework should include software management tools to automate this process and promote reproducibility for rapid deployment exploration.

R6: Configurable software stack Users should be able to switch between software components and configurations to suit the underlying hardware, e.g., use a lightweight edge resource manager like KubeEdge [16] when deploying on edge infrastructure instead of cloud-native alternatives.

2.4 Application and Benchmark Requirements

The third set of requirements relates to application deployment and benchmarking (P1-2, E1, C1) on the provided hardware and software stack. This step is essential as it determines the ability of the framework to help users improve their deployment.

R7: Automated application deployment Applications can be deployed as is on bare-metal or encapsulated in VMs, containers, serverless functions, or language-specific VMs like WebAssembly [10]. Furthermore, application deployment requires a new configuration for each resource manager. The framework should automatically detect and support these deployment options to help users quickly iterate on deployments with minimal manual involvement.

R8: Advanced observability With infrastructure set up, software configured, and applications deployed, monitoring, tracing, and logging systems such as Prometheus need to be in place to provide feedback to the user on the performance and behavior of all components and help them make informed decisions.

2.5 Framework Requirements

The final set of requirements relates to the functionality and implementation of the framework, and determines its ease of use and accuracy of recommendations.

R9: Accurate deployments For the framework's recommendations to apply one-on-one to real-world scenarios, its feedback to users should accurately represent the state of the user's targeted production environment, or else recommendations result in suboptimal real-world deployments instead.

R10: Performant framework The discussed phases of infrastructure, software, and application deployment should have low overhead compared to the desired production deployment to allow users to explore as many configurations in as little time as possible. Moreover, high overhead in one of the framework's components could skew recommendations and hurt accuracy.

R11: Extendable components Finally, it is critical for all hardware, software, and benchmarking components to be easily extendable by users, as without, design space exploration is limited to the out-of-the-box functions of the framework.

3 CONTINUUM DESIGN

Based on the requirements described in the previous section, we present the design of Continuum, a framework for automated infrastructure deployment and benchmarking in the compute continuum (Figure 3). Continuum's design is split into three parts, following

Table 2: Selection of Parameters Offered by the Framework.

Parameter (group)	Requirement	Description
Provider	R4,R9-11	Infrastructure provider for VMs, e.g., QEMU.
Devices per tier	R1,R3	Number of cloud, edge, and endpoint devices to emulate.
Resources per device	R1,R3	Per VM CPU cores, CPU quota, and memory and storage capacity.
Network per tier	R2	Throughput and latency between VMs.
Resource manager	R5-6,R9-11	Resource manager to deploy in the continuum.
Execution model	R5-6,R9-11	Optional enhancement of resource manager functionality
Application	R7,R11	Application to deploy and benchmark.
Application resources	R7	Maximum allowed CPU and memory usage of the application
Applications per device	R7	Number of applications to deploy per device
Application parameters	R7	Application-specific parameters
Network benchmark	R8-9	Perform a network benchmark between devices using Netperf
Observability	R8-9	Deploy software for extra observability
Machine addresses	R9-10	IP addresses of local, physical hardware for large-scale experiments

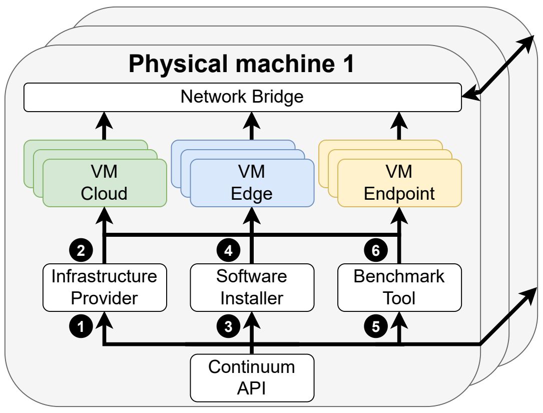

Figure 3: Design of the Continuum framework.

the sets of requirements related to hardware, software, and application deployment. We explain our design and implementation and demonstrate how users can configure the framework in Table 2.

3.1 Infrastructure Deployment

Continuum deploys infrastructure as virtual machines on one or multiple physical devices via an infrastructure provider (requirement R4 in Table 1). Currently, Continuum supports QEMU for deployment on local hardware and Google Cloud for deployment in the cloud; this can be easily extended (R11). This VM approach allows users to emulate any target architecture on any physical hardware for a performance penalty (e.g., ARM on x86), no matter if the user does or does not own the physical hardware. To achieve accurate and performant deployments (R9-10), users can emulate on the same physical hardware architecture (e.g., x86 on x86) for close-to-bare-metal performance [10]. We use VMs over other virtual resources like containers as they allow emulation of the entire operating system, contrary to containers, which is critical for users seeking to develop software there. Users can configure the provided infrastructure in terms of the number of cloud, edge, and endpoint VMs deployed (R1), resources per VM (R3), including CPU, memory, and storage characteristics, and the networks connecting VMs, including latency and throughput (R2). Continuum passes the configuration to the infrastructure provider of choice (component ❶ in Figure 3), which starts the desired virtual infrastructure (❷).

3.2 Software Deployment

Continuum uses the software automation as a service tool Ansible to install and configure software for resource managers, operating services, and application back-ends in the provided VMs (R5, R9-10). Ansible uses a declarative interface that perfectly suits Continuum's requirements: Users define the desired state of a particular software package through a single Ansible file (❸), including all configuration options; these files can then be reused (R5) and mixed (R6) to create a complex and reproducible software stack (❹). The framework does support any other software automation tool or language however, such as Bash scripts, with minimal changes required. Continuum currently supports the resource managers Kubernetes and KubeEdge for containerized applications and OpenFaaS for serverless functions, and the operating services Eclipse Mosquitto for MQTT communication, Prometheus for monitoring, and Grafana for data visualization. We demonstrate in Section 4 how these packages allow users to deploy applications in containers and serverless functions across the compute continuum.

3.3 Application Deployment

Finally, Continuum deploys applications via a resource manager such as Kubernetes, via a virtualization platform such as Docker, or directly as is using provided Ansible deployment files (❻, R7). Users can specify the deployment via the API, which includes the number of application instances per device, resources per application, and application-specific parameters (❺). Continuum then benchmarks the application and underlying systems and presents key metrics, such as resource usage, to the user (R9-10). Continuum supports Prometheus for general-purpose data gathering (R8) and helps users integrate application-specific metric gathering with Ansible (R11). This approach does not bind users to specific Continuum implementations for benchmarking and metric gathering, which related benchmarking tools suffer from (Section 5), and helps them to integrate custom solutions.

Table 3: Experimental setup for the computing model comparison in Figure 4.

Parameter	Cloud	Edge	Mist	Endpoint
(#cloud, #edge, #endpoint)	(3, 0, 4)	(1, 2, 2)	(0, 2, 2)	(0, 0, 1)
Worker location	Cloud	Edge	Endpoint	Endpoint
Workers	2	2	2	1
Endpoints per worker	2	1	1	-
Data generation rate	5	3	3	3
Resource manager	Kubernetes	KubeEdge	-	-

Table 4: VM and network configuration for the infrastructure provider comparison in Figure 4 (top) and the software and application deployment evaluation in Figures 5-7 (bottom).

Parameter	Cloud	Edge	Endpoint
CPU Cores	4	2	2
Memory (GB)	16	8	4
Quota	1.0	1.0	0.5
Latency per tier (ms)	0/7.5/45	7.5/7.5/7.5	45/7.5/7.5
CPU Cores	4	2	1
Memory (GB)	4	2	1
Quota	1.0	0.75	0.5
Latency per tier (ms)	0/7.5/45	7.5/7.5/7.5	45/7.5/7.5

4 EVALUATION

In this section, we design and perform various experiments with Continuum to show how the framework helps developers create and test software in the compute continuum, and more specifically, how Continuum satisfies the requirements listed in Table 1. Through 4 experiments, we wish to answer the following fundamental questions:

(1) *First, how does Continuum help users deploy cloud, edge, and endpoint infrastructure on general-purpose hardware?* We deploy virtualized infrastructure on local hardware with QEMU and in the cloud with Google Cloud and show through an in-depth performance analysis how differences in physical infrastructure affect Continuum's performance (Figure 4).

(2) *Second, does Continuum allow for the exploration of different computing models?* We explore various cloud and edge computing deployments (Figure 5) and show how users can switch between them in a few lines of code (Listing 1).

(3) *Third, how does our framework help users make task-offloading decisions in a heterogeneous compute continuum?* We demonstrate how Continuum offers fine-grained hardware and network configuration options to help users compare deployment options and offers detailed system and application-level metrics to guide users to an optimal deployment (Figure 6).

(4) *Fourth, does Continuum's design allow for efficient navigation of complex software stacks?* We deploy containerized applications with Kubernetes and serverless functions with OpenFaaS, and show how Continuum's modular design allows users to explore and answer the same fundamental questions for different software deployments (Figure 7).

```
[infrastructure]
provider = qemu

# VM settings for cloud, edge, endpoint
devices_per_tier = 3,0,4
cores_per_device = 4,0,2
memory_per_device = 16,0,4
quota_per_cpu = 1.0,0,0.5

# Latency (ms): average,variability
cloud_to_cloud = 0,0
cloud_to_endpoint = 45,5

# Throughput (Mbit): average
cloud_to_cloud = 1000
cloud_to_endpoint = 7.5

[benchmark]
resource_manager = kubernetes
application = image_classification
data_generation_frequency = 5
```

Listing 1: Framework configuration for the cloud deployment with QEMU presented in Tables 3 and 4 and Figure 4.

4.1 Experimental Setup

Tables 3 and 4 list our experimental setup. We deploy Continuum locally on a cluster of Xeon Silver 4210R machines connected with a 1 Gbps link and create virtual infrastructure using the QEMU/KVM hypervisor (v6.0). We use multiple physical machines in setups that require many virtual resources to prevent performance degradation due to oversubscription. To emulate network and storage resources, we utilize the software tools *tc* and *blkiotune*. We also use Google Cloud Platform (GCP) for virtual machines in the cloud, and compare GCP to QEMU in Figure 4 using the same resource configuration (top half of Table 4). These resources differ from the remaining experiments (bottom half of Table 4) due to restrictions in deployment options in Google Cloud. In general, cloud VMs have the most processing power and memory, followed by edge and endpoint, but also have the highest communication latency and lowest throughput to the endpoint data source.

For our experiments, we use a machine learning use case that emulates a security camera endpoint generating a constant number of images per second, which then need to be processed with image classification algorithms. The processing can happen locally on an endpoint or can be offloaded to the cloud or edge for increased performance. Communication between cloud, edge, and endpoint applications is handled by Eclipse Mosquitto, a communication service for the lightweight pub/sub MQTT network protocol that can function with minimal network resources, such as those available at the edge or endpoint. In general, by splitting our application into a data generation and processing component, our findings apply to any application that follows the same split of responsibilities.

We deploy Kubernetes as the resource manager of choice in the cloud and KubeEdge on the edge. Endpoint devices are typically single-tenant [11] and therefore do not need advanced resource management software. For Figure 7, we use serverless computing with OpenFaaS deployed on top of Kubernetes. We provide more detailed information on our setup in the following section.

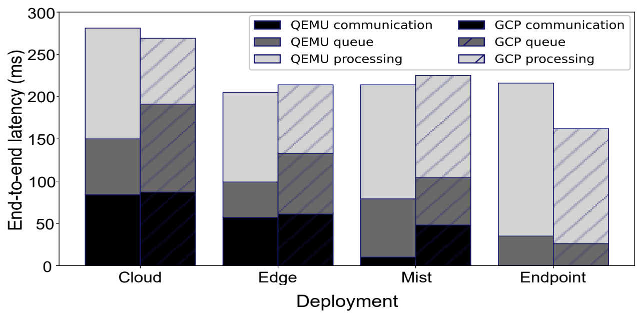

Figure 4: End-to-end latency breakdown for diverse deployments with QEMU and GCP as infrastructure providers.

4.2 Infrastructure Provisioning

For our first experiment, we show Continuum's infrastructure provisioning on local hardware and remote clouds. We deploy the aforementioned machine learning application on four different hardware deployments: (i) cloud computing, offloading workload from endpoints to cloud, (ii) edge computing, offloading from endpoints to edge, (iii) mist computing, offloading from endpoints to more powerful endpoints, and (iv) a baseline of processing on endpoints without offloading. The exact configurations are shown in Tables 3 and 4. We provide Continuum's configuration file for the cloud computing deployment in Listing 1. This configuration describes how image processing tasks are offloaded from four endpoint VMs to two cloud VMs, with the third cloud VM exclusively running Kubernetes. Each cloud VM runs one instance of our processing application and is connected to two data-offloading endpoints. Users can easily switch to an edge computing deployment by changing the *devices_per_tier* to 1, 2, 2 as shown in Table 3, and the resources per device and network between devices accordingly. We provide more examples in our open-source code.

We deploy and emulate deployments for the four computing models on local hardware with QEMU and with GCP in the cloud, and provide a breakdown of end-to-end latency in Figure 4, the time between the generation of a data element and its processed output being available to the original data source device. The breakdown consists of 3 parts: First, network communication between endpoint and offload target and vice versa. Second, queuing delays, the time between data arriving on the offload target and the processing application starting to process the data element. Third, processing time per data element. The endpoint baseline processes data locally and therefore has no network communication.

GCP is faster in all processing tasks in Figure 4 due to the use of more performant physical hardware compared to our local setup. Communication delays are similar for cloud and edge but significantly differ for GCP's mist deployment: We argue that this and other performance variations for GCP that occurred over multiple repetitions of this experiment (these are omitted from Figure 4 for clarity) are caused by the multi-tenant environment of Google Cloud as performance variations on our local setup are negligible. Finally, the queuing overhead differs significantly between deployments and does not follow a clear pattern, e.g., QEMU has less overhead for cloud and more for mist. The implementation of the ML application most likely causes this: Endpoints generate a

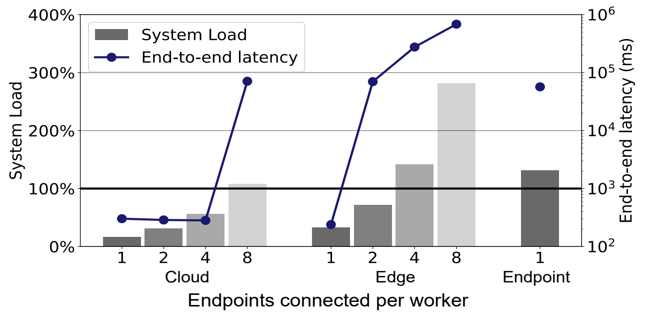

Figure 5: System load and end-to-end latency when offloading from an increasing number of endpoints to cloud or edge workers, compared to local processing on an endpoint.

constant number of images per second and offload these images without waiting on the offloading result. The data processing application maintains a queue of incoming images to process, and if it can process each data element before the next one arrives, queuing delays will be minimal, and real-time processing is guaranteed. The deployments in Figure 4 have been configured to achieve real-time processing and so minimize queuing delays. However, a queue can still form for a limited time due to performance fluctuations, which then significantly increases the reported average queuing delay.

In conclusion, we show that Continuum helps users to emulate infrastructure and networks on general-purpose hardware using QEMU and GCP, and achieves close-to-bare-metal performance by leveraging hardware acceleration technology [10].

4.3 Exploration of Computing Models

For our second experiment, we show Continuum's ability to deploy applications with various computing models and offloading variations. Our results in Figure 4 show an expected increase in communication overhead when offloading further away from the endpoint, with cloud offloading having the highest overhead, as defined in Listing 1. On the other hand, the difference in processing time is only apparent when comparing cloud, edge, and mist to the endpoint baseline, which is set up to be resource constrained. To make the difference in processing power more apparent, we increase the number of endpoints connected per offload target in Figure 5 and report both end-to-end latency and system load. Here, a system load of less than 100% signifies that the processing application has enough resources to process an offloading request before the next one comes in; see our previous discussion on queuing overheads. We see that the cloud deployment can process offloaded workload from up to four endpoints in real-time, however, when increasing the number of endpoints beyond four, the required processing capacity for real-time processing exceeds the available cloud resources, indicated by the system load of 105%. This results in offload tasks queuing up at the cloud and end-to-end latency exponentially increasing. We can see similar results for edge computing, where workload queues start forming with two simultaneously connected endpoints, albeit temporarily due to performance fluctuations as the average system load is still below 100%. To prevent this, one should either reduce the processing requirement (by reducing the number of endpoints or the data generation rate) or increase the processing capacity (by using more cloud or edge resources). Users

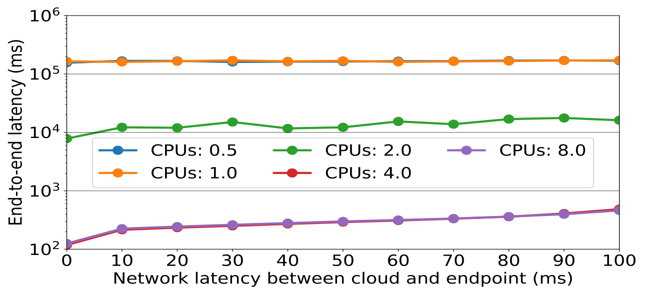

Figure 6: End-to-end latency when offloading from 4 endpoints to a single cloud worker with a variable network latency as well as CPU cores and memory on the cloud worker.

can explore these parameters in Continuum to iteratively search for suitable deployments, as displayed in Listing 1.

In conclusion, we show that Continuum simplifies the exploration and deployment of computing models based on cloud, edge, and endpoint resources, and helps users determine if task offloading is required for their use case, and if so, where to offload to.

4.4 Exploration of Deployment Requirements

For our third experiment, we show how users can iteratively update their infrastructure configuration to find a deployment that satisfies their desired target metrics. We provide an example in Figure 6: We offload workload from four endpoints to one cloud machine and vary the communication latency between the two types of machines and the processing power of the cloud machine in terms of CPU cores and memory. We use 1 GB of memory per CPU core. For the variation in processing power on the cloud, configurations with less than four CPU cores and four GB of memory have very high end-to-end latency, indicating large queuing delays. Therefore, in this case, users are recommended to use cloud VMs with at least four CPU cores and four GB of memory; more processing power does not significantly improve performance but can reduce performance variability. For this reason, we only see the effect of varying communication latency on the configurations with four or more CPUs. The end-to-end latency steadily increases when increasing network delay as expected, allowing users to accurately determine what network latency is tolerable for their use case. For example, VR and AR applications require much lower end-to-end latency for a satisfactory user experience than smart farming and weather monitoring applications [9].

In conclusion, we demonstrate Continuum's ability to quickly emulate and benchmark various hardware and software deployments, allowing users to iterate through various configurations in search of one that matches their desired requirements.

4.5 Deployment Flexibility

For our final experiment, we deploy and benchmark a serverless software stack with OpenFaaS as an alternative to the previously used container deployments with Kubernetes and KubeEdge to show how Continuum's modular design allows users to switch between software components on the fly while providing the same system and application metrics to the user. Serverless computing requires different deployment logic and application architecture for

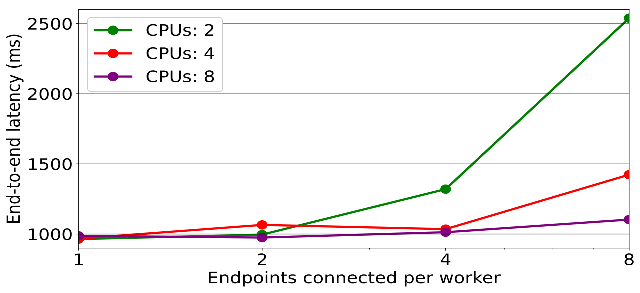

Figure 7: Serverless application deployment with varying compute resources and endpoints connected per worker.

optimal performance: For our previous container deployment, we used a client-server model for the machine learning use case, with data-generating applications at the endpoint offloading images to a permanently running data processing server for analysis. With the serverless paradigm however, a single instance of the data processing application is started on the cloud or edge for every incoming image offloaded from the endpoint, and every instance only processes a single image. This also means that offloaded tasks can no longer queue up in the data processing application; endpoints now have to wait for their offloaded data to be processed before offloading a new item, moving the queuing delay to the endpoint. This approach simplifies application deployment as many distributed system concerns like scalability and availability are moved from the user to the serverless resource manager.

We implement our machine learning use case as a serverless application, deploy it on the cloud with OpenFaaS, and show our results in Figure 7. We vary the number of endpoints connected to a single cloud worker and cloud CPU and memory resources with 1 GB of memory per CPU core. A key difference between serverless and container deployments shows: Even with sufficient resource capacity, the end-to-end latency is never lower than 1000 ms compared to latencies of 200 to 300 ms for containerized deployments previously shown in Figure 4. This is because OpenFaaS has to start the data processing application for each incoming task offloading request, which was previously not needed due to our client-server operation model. Unsurprisingly, we also see that as soon as more endpoints connect to a single cloud worker than resources available to that worker, end-to-end latency increases due to endpoints having to wait longer for their offloaded processing tasks to complete, delaying the offloading of newly generated images. This shows the complexity of comparing various distributed application deployments and again confirms that Continuum is vital for accelerating software development in the compute continuum.

In conclusion, we prove that Continuum helps users switch between and compare complex distributed software stacks, such as containerized or serverless application benchmarks, without altering the framework's core functionality that assists users in improving and iterating on their deployments.

5 RELATED WORK

The main related work of Continuum comprises tools and frameworks that offer (i) infrastructure emulation for the compute continuum, (ii) automatic software installation for application and

Table 5: Requirement analysis for selected infrastructure emulators and simulators as well as application benchmarking frameworks for the compute continuum. Symbols: ●: Requirement covered; ○: Not covered; ◐: Partially covered.

ID	Requirement	[14]	[3]	[13]	[7]	[8]	[5]	[12]	This work
R1	Cloud, edge, and endpoint resources	●	●	●	●	○	○	○	●
R2	Network resources	●	◐	●	●	○	○	○	●
R3	Configurable resources	●	●	●	●	○	○	○	●
R4	Flexible deployment options	●	○	●	●	●	◐	●	●
R5	Automated software stack installation	◐	◐	○	○	○	○	○	●
R6	Configurable software stack	◐	●	○	○	○	○	○	●
R7	Automated application deployment	◐	●	○	○	●	◐	●	●
R8	Advanced observability	●	●	○	○	●	●	●	●
R9	Accurate deployments	◐	●	○	○	○	○	○	●
R10	Performant framework	●	●	●	●	○	○	○	●
R11	Extendable components	●	●	◐	◐	●	◐	●	●

system-level software, and (iii) application benchmarking and observability. We select prominent related work in these fields and analyze if they satisfy our requirements for such tools in Table 5.

Symeneonides et al. present Fogify, a framework for emulating edge and fog deployments using containers [14]. Fogify allows for advanced configuration options, but its reliance on containers over virtual machines limits the software that can be deployed on it, and therefore may not give an accurate representation of the user's targeted deployment. Hasenburg et al. present MockFog, a tool for the automated execution of applications on emulated fog devices [3]. Mockfog deploys virtual machines similarly to Continuum but only uses cloud services. Furthermore, it heavily relies on Docker for application execution, similar to Fogify, and does not leverage resource managers or distributed services within VMs, limiting the types of deployments that can be tested.

EdgeCloudSim [13] and iFogSim [7] are examples of simulators for compute continuum deployments and provide an easy-to-use but simplified view of hardware and software. These simulators should therefore only be used in an initial exploratory phase before an emulator such as Continuum [15].

Many benchmark suites have been published that consider continuum resources, systems, and applications, such as DeFog [8], CoAP [5], and RIoTBench [12]. These tools do not offer any infrastructure or software provisioning services but focus on the deployment and benchmarking of applications instead. For example, DeFog can automatically offload applications to user-provided cloud and edge devices and advise a particular scenario using system and application-level metrics, similar to what we have shown in Figure 4. We argue that the coupling of infrastructure emulation and benchmarking that our framework offers is key in efficiently exploring the design space of workload deployments in the compute continuum and that Continuum is unique in this offering; specialized benchmarking tools offer more application and observability choices out of the box, but limit users in benchmarking on their existing physical hardware.

6 CONCLUSION

In this paper, we present Continuum, a framework for (i) cloud, edge, and endpoint infrastructure emulation, (ii) automatic software installation and configuration, and (iii) application benchmarking. We make a case that these three components are essential for exploring the compute continuum design space and that missing one of these significantly limits a developer's ability to quickly iterate between deployments with reliable performance data. We analyze the SPEC-RG compute continuum reference architecture and synthesize a list of key requirements that back this claim. Then, we present Continuum's design that implements these requirements and present an intuitive API for users to configure their deployment. We demonstrate Continuum's ability to guide users towards more informed offloading decisions by deploying container and serverless-based application benchmarks with Kubernetes and OpenFaaS on local and remote hardware using QEMU and Google Cloud. Our future work includes extending Continuum's out-of-the-box infrastructure and software support to more cloud providers and resource managers and diversifying the available metrics to better assist more users. Continuum is available at https://github.com/atlarge-research/continuum.

ACKNOWLEDGMENTS

This work is funded by NWO TOP OffSense (OCENW.KLEIN.209)

REFERENCES

[1] Corneo et al. 2021. Surrounded by the Clouds: A Comprehensive Cloud Reachability Study. In *WWW*.
[2] Alex Ellis. 2022. OpenFaaS - Serverless Functions Made Simple. https://github.com/openfaas/faas. Accessed: 2023-01-21.
[3] Hasenburg et al. 2020. MockFog 2.0: Automated Execution of Fog Application Experiments in the Cloud. *CoRR* abs/2009.10579 (2020).
[4] Matthijs Jansen et al. 2022. The SPEC-RG Reference Architecture for the Edge Continuum. https://doi.org/10.48550/ARXIV.2207.04159
[5] Kruger and Hancke. 2014. Benchmarking Internet of things devices. In *INDIN*.
[6] Linux Foundation. 2021. State of the Edge 2021. https://project.linuxfoundation.org/hubfs/LF%20Edge/StateoftheEdgeReport_2021.pdf. Accessed: 2021-06-06.
[7] Mahmud et al. 2022. iFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environments. *J. Syst. Softw.* 190 (2022).
[8] McChesney et al. 2019. DeFog: fog computing benchmarks. In *SEC*.
[9] Mohan et al. 2020. Pruning Edge Research with Latency Shears. In *HotNets*.
[10] Rijn and Rellermeyer. 2021. A fresh look at the architecture and performance of contemporary isolation platforms. In *Middleware*.
[11] Satyanarayanan et al. 2019. The Computing Landscape of the 21st Century. In *HotMobile*.
[12] Shukla et al. 2017. RIoTBench: An IoT benchmark for distributed stream processing systems. *CCPE* 29, 21 (2017).
[13] Sonmez et al. 2018. EdgeCloudSim: An environment for performance evaluation of edge computing systems. *Trans. Emerg. Telecommun. Technol.* 29, 11 (2018).
[14] Symeonides et al. 2020. Fogify: A Fog Computing Emulation Framework. In *SEC*.
[15] Varghese et al. 2022. A Survey on Edge Performance Benchmarking. *ACM Comput. Surv.* 54, 3 (2022).
[16] Xiong et al. 2018. Extend Cloud to Edge with KubeEdge. In *SEC*.
[17] Yousefpour et al. 2019. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *J. Syst. Archit.* 98 (2019).