Internet of Things 22 (2023) 100737

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

Secure-by-design serverless workflows on the Edge–Cloud Continuum through the Osmotic Computing paradigm

Gabriele Morabito, Christian Sicari, Armando Ruggeri, Antonio Celesti, Lorenzo Carnevale *

University of Messina, Messina, Italy

ARTICLE INFO

Keywords:
Serverless computing
Edge–Cloud Continuum
Osmotic Computing
Cloud computing
Edge computing

ABSTRACT

Nowadays, many Cloud companies adopt serverless computation based on the Function as a Service (FaaS) paradigm. Such an approach, built on the dynamic provisioning of serverless functions, has significantly altered the design of Cloud applications. Indeed, complex systems may now be viewed as a graph of serverless functions (i.e., workflow) distributed over the Cloud and Edge layers. However, building applications in such a complicated context and introducing security by design is still a non-trivial problem. Meanwhile, Osmotic Computing has advanced as a novel computational paradigm in recent years. This may serve as a suitable foundation for strengthening security in serverless applications throughout the Cloud–Edge Continuum. In this work, we rely on Osmotic Computing principles for modifying the architecture of OpenWolf, a novel serverless engine, and improving the execution time of ciphertext data by about 65%.

1. Introduction

Serverless computing emerged in 2014 from Amazon through the Lambda Project, intending to simplify software distribution over a Cloud environment. The most famous serverless implementation is called Function as a Service (FaaS), which allows developers to focus on the business logic (i.e., the *function*), while the FaaS platform containerizes, deploys, and load balances it. For this purpose, FaaS gained soon popularity, and many open-source projects have been created as alternatives to vendor lock-in solutions (i.e., Openwhisk [1], Knative [2], OpenFaaS [3–5]). Nowadays, these solutions are mature enough and the scientific community has started projects (i.e., [6,7]) that include FaaS at the edge of the network for quickly responding to triggers coming from the Internet of Things (IoT).

Unfortunately, there are only a few cases of applications that totally run at the edge of the network; indeed, they often use to interact with a Fog or Cloud tier for more intensive work. This well-known computation pipeline is called Edge–Cloud Continuum (or *Continuum*). FaaS, instead, is mainly used for spotted and poorly coupled applications where there is no need to link and orchestrate different events and relative functions. The lack of composability for the function could be considered an architectural choice, but in the end, it limits the usability of the serverless itself, especially in the Continuum. Big Players have addressed this problem by proposing their function orchestrators, like Amazon Step Function, Microsoft Azure Durable Functions, and IBM Composer. However, these systems cannot be used everywhere because they are constrained at the providers' data centers.

Recently, OpenWolf [8] has been presented as the first open-source serverless engine capable of composing functions using three main components: a workflow manifest, a Kubernetes heterogeneous cluster, and a Broker Agent. OpenWolf aims to bring the

* Corresponding author.
E-mail addresses: gamorabito@unime.it (G. Morabito), csicari@unime.it (C. Sicari), armando.ruggeri@unime.it (A. Ruggeri), acelesti@unime.it (A. Celesti), lcarnevale@unime.it (L. Carnevale).

https://doi.org/10.1016/j.iot.2023.100737
Received 30 December 2022; Received in revised form 19 February 2023; Accepted 24 February 2023
Available online 28 February 2023

serverless at the Continuum layer, but this leads to some problems related to the environment's security, such as the definition of a secure overlay network for federating both the Continuum nodes, storage, and transmission of sensitive data. These contents have been treated in the field of distributed computing, but to the best of our knowledge, no one argued it for a serverless Continuum environment.

A universal approach for dealing with these aspects was proposed in the Osmotic Computing paradigm [9]. The goal of this high-level bio-inspired paradigm is to orchestrate and secure services in a federated heterogeneous environment. Osmotic Computing, instead, was intended to enable computation offloading between Cloud and Edge Computing, which used container technologies to abstract computation from the infrastructure. Osmotic Computing has evolved over time to include and enhance new trends in the Cloud–Edge orchestration field, ultimately proposing good solutions to many continuum issues. Cloud–Edge–IoT Continuum suffers from many defects, mainly (i) the offloading management, (ii) the management of stateful applications, (iii) the security granting. To the best of our knowledge, the solutions proposed in literature are thought to solve a specific use case problem; instead, Osmotic Computing proposes a universal solution to accomplish the mentioned issues. In addition, this computing paradigm proposes (i) the MicroData and MicroService concepts to split computations from data and then transparently manage the application state, (ii) the Concentration to continuously load balance the computations over the infrastructure, and (iii) the Software Defined Membrane (SDMem) to keep secure the interconnected micro applications.

We believe that OpenWolf is a good starting point for the development of a Cloud–Edge continuum system because it easily manages distributed function-based applications spread over Cloud and Edge. Nevertheless, as well as in many other Continuum computing engines, it raises some security threats that, in this work, we aim to solve through the Osmotic Computing paradigm. Summarizing, this paper aims to address the following security aspects:

- O1 **Secure storage for sensitive data**, which assures that confidential information is not disclosed to unauthorized individuals.
- O2 **Secure and authorized secret distribution**, because serverless applications are distributed across the Cloud–Edge Continuum, and, in a so complex environment, defining a way for distributing secrets is crucial.
- O3 **Functions access control**, which guarantees that the interaction between functions is under control, disallowing malicious functions can inject forbidden invocations.
- O4 **Communication security** between nodes that prevents unauthorized parties from gaining understandable access to the communication, granting that the object of the communication is delivered to the expected receiver. This is also a complex security aspect in a serverless environment because serverless functions, which are spread across the Edge–Cloud Continuum, are deployed by a serverless function provider. Indeed, serverless workflows, composed of serverless functions, imply many internode communications, and it is impossible to a priori know on which particular node the serverless functions will be deployed or executed by the serverless functions provider.
- O5 **Message security**, that consists in the process of isolating any communication between the components of an application, mainly through the encryption of messages of the exchanged data. In the case of a serverless environment, the entities exchanging messages are serverless functions that, by definition, are unaware of being part of a particular application. For this reason, guaranteeing message encryption is challenging.

The main scientific contributions of this research are:

- identifying security vulnerabilities of the existing OpenWolf implementation;
- implementing the Osmotic SDMem concepts to improve serverless security in the Cloud–Edge continuum;
- validate the implemented features with a non-secure implementation of OpenWolf, in terms of system execution time and resource utilization.

The remainder of this paper is organized as follows. In Section 2 we discuss the related works, while in Section 3 we focus on Osmotic Computing and OpenWolf which are the baseline of this work. The implementation is reported in Section 4, providing details and insights about the choices taken to accomplish the above-mentioned objectives. Section 5 describes the overall system performance and the testbed to replicate the experiments. In the end, Section 6 concludes the work and brings light to the future.

2. Related works

When Serverless technologies are adopted, applications can be managed without the need to develop a server from scratch. By doing so, the service provider handles some security aspects. Recently, many works have been proposed addressing different aspects of the field of security in Serverless environments. In [10–12], authors underlined the fact that even if Serverless applications do not run on a managed server, they continue to execute code. If this code is not written securely, the application may be exposed to traditional application-level attacks. In particular, the security risks associated with serverless are summarized in ten points, such as (i) code injection [13], (ii) broken authentication, (iii) sensitive data exposure, (iv) XML external entities, (v) broken access control [14], (vi) security misconfigurations, (vii) cross-site scripting, (viii) insecure deserialization [15], (ix) using components with known vulnerabilities, and (x) insufficient logging and monitoring. In addition to that, it should be taken into account that applications are highly scalable in a serverless environment, and many parallel computations can be triggered. For this reason, even a single bit leak could cause problems, as it could be used to obtain access to secure data [16].

In a serverless environment, there are two main security drawbacks [17]. On one hand, the security level strictly depends on the features given by the vendor. On the other hand, if insecure code is used for serverless functions, the attack surface is extended.

Thus, the authors accurately suggested how to choose the vendor service, paying particular attention to the quality of the code and introducing continuous monitoring of the production environment. Researchers are suggesting approaches to improve the security mindset of the consumer, applying specific actions, such as (a) introducing a culture of collaboration, (b) creating security-by-design architectures [18], (c) introducing threat modeling and limitation of authorization [19], (d) implement secure coding standards [20], and (e) automate secure deployment systems, exploiting continuous monitoring [21]. Such approaches are required in a Serverless environment to ensure comprehensive security controls based on multilevel protection [22].

In the fields of authentication and authorization, external request permissions should be verified for all the functions at the workflow entry point. This allows malicious requests to be blocked as early as possible, thanks to an authentication and authorization system based on the decoupling of authentication from execution with the use of a message-oriented middleware [23,24], and using JSON Web Token (JWT) and Oauth2 protocol for authentication, authorization and access control in a serverless environment [25]. In [26], the authors proposed a dynamic and self-adapting approach for data access and protection during its whole lifecycle in the Cloud-to-Edge continuum. This solution was proposed to protect data in applications deployed in the continuum but not for sequential and parallel serverless workflow executions. However, a similar approach could also be adopted for this kind of application. In the field of resource isolation, the greatest challenge is represented by the weak isolation of lightweight virtualization systems, aided by virtual machines and containers to isolate functions and contexts [12,22]. Containers' security strictly depends on the underlying operating system (i.e., a breached container instance can easily trigger an operating system vulnerability); whereas virtual machines (VMs) ensure better isolation, but they are more resource-consuming.

Nevertheless, different solutions for a lightweight and secure execution have been proposed in recent years: Amazon introduced FireCracker [27] in 2018, a new hypervisor that uses microVMs for deploying serverless functions; Google developed gVisor in 2019 [28] to completely isolate serverless function deployed in containers from the host operating system. However, even if functions are isolated by being deployed using containers, malicious code can still trigger the execution of prohibited functions. Thus, it is not sufficient to isolate the software execution, and some network isolation should be introduced. In the field of data protection, encryption is necessary to protect data both in idle and transit. Moreover, some key management services are needed to handle application secrets [22], such as cryptographic keys.

Serverless is mainly born as a natural evolution of the microservice architecture that typically runs in Cloud, but it is common to find Edge deployment [29–31]. Unfortunately, security risks of serverless on distributed environments are poorly treated. In [32], the authors proposed a Multi-Cloud Performance and Security (MCPS) Brokering framework for resource allocation of a set of workflows across federated Multi-Cloud infrastructures. However, even if it is possible to introduce security constraints in selecting the nodes for the computation, it is limited to Cloud infrastructure and only uses VMs. Therefore, it does not use lightweight virtualization solutions, such as containers, and does not support Edge nodes. Another proposal for a secure scheduling system of workflows was made in [33], where some security constraints were introduced for the scheduling of jobs in a four-tier environment composed of IoT sensors and devices, mist resources, fog resources, and Cloud resources. Such a solution does not have an implementation for serverless computing, but its applicability could be really important for that field.

3. Background

The scenario in which we work is composed of two main elements: (i) Openwolf, the engine used for deploying FaaS workflows over the Continuum; and (ii) Osmotic Computing, the computing paradigm we plan to use for addressing the security objectives mentioned above.

3.1. OpenWolf

OpenWolf is an open source[1] workflow engine for parallel asynchronous computation based on FaaS. The OpenWolf architecture is shown in Fig. 1 and the infrastructure is based on Kubernetes; in particular, K3S is used to orchestrate services over a Continuum environment that federates Cloud, Fog, and Edge nodes. On top of K3S, OpenFaas is run, which is nowadays one of the most used open-source Serverless engines. It allows the building of multi-architecture native containerized functions that it orchestrates and balances over Kubernetes, taking even into account user-defined constraints when they are defined. OpenWolf is a single replicated agent installed alongside OpenFaas, it tells OpenFaaS when and where running functions, forwarding the function's output according to a Workflow Manifest. The Workflow Manifest is a YAML file stored inside a Redis instance that describes how a workflow is structured and how workflow instances evolve inside OpenWolf. This manifest is based on the open-source project Serverless Workflow DSL, and as shown in listing 1 is mainly composed of three parts: (i) the *States* describes all the workflow' steps; (ii) the *Functions* configure the function used inside the States; (iii) the *Workflow* that through a boolean equation describes when the states are triggered.

```
name: <workflow-name>
callbackUrl: <uri-where-to-send-result>
states:
  <state-id>:
    function:
      ref: <ref-to-function-id>
```

[1] https://github.com/christiansicari/Open-Wolf-Serverless-Workflow

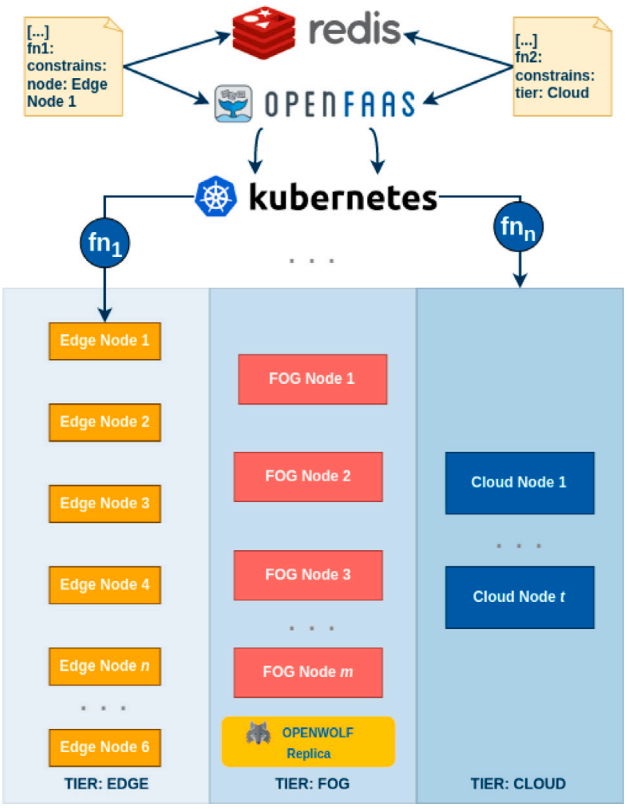


Fig. 1. OpenWolf architecture.

```
      config: [...]
      constraints: [...]
functions:
  <function-id>:
    platform: openFaaS
    endpoint: <endpoint-to-function>
    config: [...]

workflow:
  <state-id>:
    activation: <Boolean Equation>
    dataFilter: [...]
```

Listing 1: Workflow Manifest

OpenWolf works like a broker for OpenFaas. It receives external and internal events to the web interface; these events contain a reference to the workflow they want to trigger; OpenWolf fetches the referenced workflow from Redis, then solves the State Activation equations in the workflow section to decide which states must be activated. OpenWolf triggers the functions referenced in the States definition to be activated, instructing OpenFaas to send back to it the function results as events that, in turn, will restart this loop in OpenWolf, making the Workflow going ahead.

As a serverless continuum solution for scientific workflow, OpenWolf is affected by security issues related to the trustness of Cloud and Edge nodes, communication channels, and function reliability. In addition, different services (i.e., workflow) and applications may require different security constraints. Therefore, OpenWolf needs a component responsible for guaranteeing the migration of functions and end-to-end data, by dynamically creating isolation of workflows among tiers, according to defined security level agreement and quality of service. In this regard, the requirement of elastic configuration is borrowed from the Osmotic Computing paradigm, specifically, the SDMem.

3.2. Osmotic computing

Osmotic Computing is a high-level paradigm, introduced in 2016, that expands the concepts of Cloud Computing and Edge Computing, connecting them together. It aims at decomposing applications into microservices and performing dynamic tailoring of

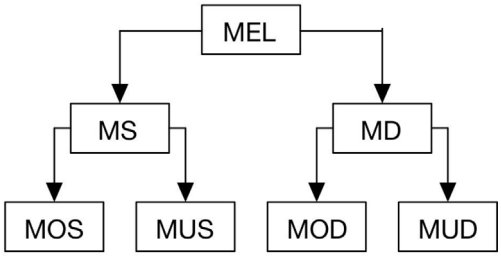


Fig. 2. MELs classification.

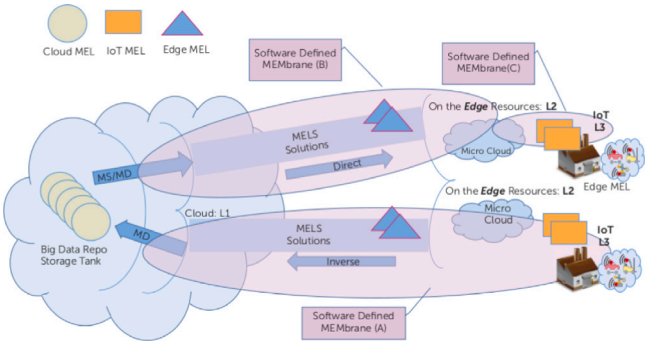


Fig. 3. SDMem in osmotic computing [35].

microservices in smart environments exploiting resources in both Cloud and Edge infrastructures [9]. As the name itself suggests, this paradigm is inspired by the chemical phenomenon of osmosis, which consists of the diffusion of the particles of a solvent from the region with its highest chemical potential to that with the lowest chemical potential when the diffusion of the solute is prevented by means of a semi-permeable membrane. Similarly, in Osmotic Computing, microservices should move within an Osmotic Membrane to find a configuration that satisfies given requirements (i.e., quality of service). Osmotic Computing defines a lot of terms and concepts, but we mainly focus on (i) MicroElements (MELs) and (ii) Membrane. The first ones are software or data abstractions and are classified according to their nature [34] of MicroService (MS) or Microdata (MD), as shown in Fig. 2. They are furthermore divided into MicroOperationalServices (MOS) and MicroUserServices (MUS) and between MicroOperationalData (MOD) and MicroUserData (MUD). More specifically: (i) MOS can be associated with operative systems; (ii) MUS can be associated with user applications; (iii) MOD stores configurations for MSs; and (iv) MUD stores user data.

While MODs are usually sharable between different applications, MUDs are designed to be accessed by the owner (i.e., a user, a MS, or an application) to ensure privacy. Osmotic Computing defines a concept of intra-membrane that isolates the MS/User space and MUD from the outside, satisfying the objectives O1 and O2.

The Osmotic SDMem is, instead, a virtual environment based on the underlying infrastructure (Cloud or Edge resources). Inside this environment, MELs are isolated from the rest of the world. Indeed, they can migrate through the osmotic nodes without any kind of interaction with external infrastructure. The Osmotic Membrane acts then as a filter to limit how MELs can be migrated and under which constraints [35]. (See Fig. 3).

SDMem allows each organization to enable the grouping and filtering of MELs [36] based on their properties and purposes when organized like a federated ecosystem. SDMem allows MELs to migrate according to constraints identified in the membrane, guaranteeing isolation of one system from another [37]. A SDMem is instead a security component that Osmotic Computing introduces in order to guarantee privacy and confidentiality for authorized interactions. Firstly, a MEL can interact only with the MELs in the same SDMem, satisfying the Objective O3. Moreover, the SDMem guarantees a network privatization that isolates the connections from the rest of the cluster, satisfying the Objective O4. In the end, any point-to-point message is confidential by design, which means that another MEL in the same SDMem cannot access it, satisfying the Objective O5. Thus, SDmem allows isolating nodes in a federated environment, in that way MELs can only move through other nodes that are part of the same SDMem. At the same time, it allows the isolation of different membranes (i.e., creating computation context) and therefore the MELs of which it is composed in order to prevent unexpected interaction.

4. Threats analysis and mitigation

According to the above-mentioned background in Osmotic Computing, we borrow the concepts of MELs and SDMem to implement a secure execution of serverless workflows on a Continuum environment, merging into OpenWolf components (i.e., OpenFaaS and Kubernetes). Below, we identify the main threats and discuss of to mitigate them.

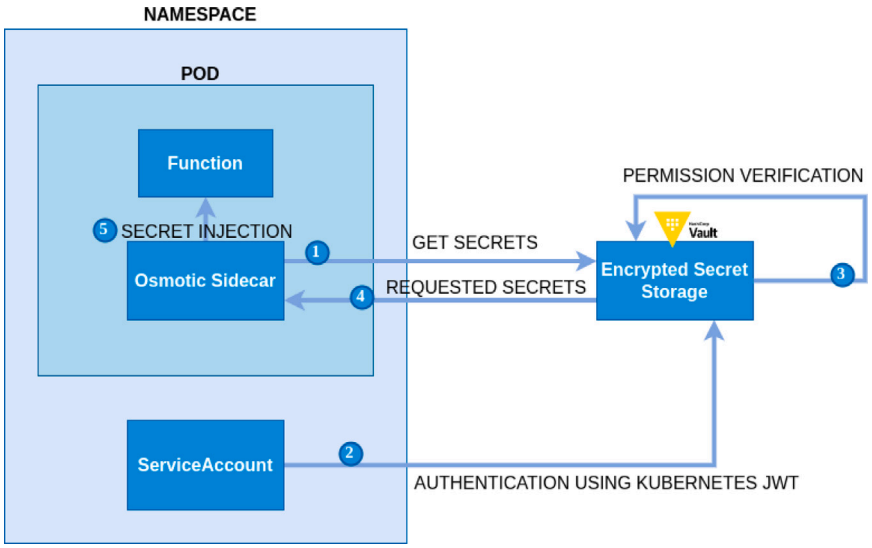


Fig. 4. MEL on OpenWolf.

4.1. Data and communication privacy

When OpenFaas builds and deploys a function, it is encapsulated within a container that Kubernetes includes in a pod, acting as MS. Even if Kubernetes includes the concept of secret, it is not good enough to guarantee data privacy, since this is stored in plain-text, and its access is not regulated by any policy. In this regard, while implementing MOD and MUD, we need to use Hashicorp Vault, which is used to store encrypted secrets (aes-gcm 256 bit). Therefore, a client needs to get authentication and authorization according to the secret's policies before accessing the secret. This feature satisfies the Objective O1. Even if Hashicorp Vault encrypts data, this could be stolen during the transmission from the Vault to the Pod. Therefore, we use the Vault Injection to deploy a vault container in the same pod where the function container lies. In this way, data are encrypted locally. This feature satisfies the objective O2.

When OpenWolf deploys the function on Kubernetes, the Vault injection happens transparently. Therefore, the function container does not need to directly access secret data.

In Kubernetes terms, running a container that supports another one in the same pod is called the side-car pattern. In our case, we are adding a container that lets the function pod act like an Osmotic MEL, guaranteeing privacy and security to confidential data; for this reason, we call this container Osmotic Sidecar. Fig. 4 shows how the function, the sidecar, and Vault work together.

Basically, to enable the sidecar in any Function we have to follow four steps: (i) enabling Vault; (ii) defining a role in Vault; (iii) associating a role to a function/pod; (iv) granting permission to a role for accessing a secret.

4.2. Malicious faas invocation

SDMem acts as Kubernetes tuning. Specifically, it allows constraining, isolating, and protecting MELs in a Continuum environment. In the OpenWolf architecture, this can be ensured by exploiting the network features of Kubernetes, which are (i) the interaction with the pods' deployment phase and (ii) the pod's network ingress/egress network rules.

Function interactions are ruled by the Manifest, which describes how each function sends data to others in the Workflow. Unfortunately, this approach does not prevent a malicious function from invoking other functions by hard coding its address and bypassing OpenWolf, which instead could notice an unattended connection attempt. SDMem is then designed to allow only legal interactions and reject any others. The implementation of the SDMem on OpenWolf is based on two features of Kubernetes: (i) Namespaces and (ii) Network Policies. Usually, OpenFaaS deploys any function in the same namespace, but we changed this behavior to add an ad-hoc namespace that contains all the pods that belong to the same membrane, and these pods can interact using their relative proxy interface placed in ad-hoc separated namespaces. To do that, we denied by default any ingress policy to the OpenFaaS gateway, with the exception of the OpenWolf agent and the proxy. The agent can access the allowed functions running in another namespace that accepts ingress traffic from those functions. An example of a policy applied to the OpenFaaS gateway is shown in the listing 2.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: faas-gateway-policy
  namespace: openfaas
```

```
spec:
  podSelector:
    matchLabels:
      app: gateway
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: openfaas
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: openwolf
        - namespaceSelector:
            matchLabels:
              app: nginx
```

Listing 2: Policy applied to the OpenFaaS gateway

An example of a policy applied to the proxy is, instead, shown in the listing 3.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: proxy-group1-network-policy
  namespace: proxy-group1
spec:
  podSelector: {}
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: openfaas
```

Listing 3: Policy applied to the proxy

The behavior we shaped is given in Fig. 5, where we graphically show how Network Policies works, basically rejecting any interaction that does not belong to the same membrane and that does not pass through the proxy.

Message Security, instead, consists of the encryption of the data exchanged between functions linked in a workflow. As we already mentioned, cryptography is a transparent option for the functions which are unaware of how data arrives at the destination. On the other hand, encryption and decryption must happen in a protected environment to avoid data theft. To respect those constraints, we generate a key encryption secret for each function and then deploy this secret both in Vault within the function's Osmotic sidecar and in the agent, as shown in Fig. 6. The Osmotic sidecar injects the key into the function container that uses it for encrypting the function's output before it reaches the pod. The Agent instead decrypts data with the same key stored in its own vault, and it encrypts this data with the key of the next function in the workflow manifest. This satisfies the Objective O5.

Even if the pods' interaction is protected by message encryption, the communication is not protected. SDMem isolates and protects by design any data exchange at the network level to prevent sniffing of encrypted data that could be decrypted with a brute force attack. OpenWolf manages the Continuum Network Federation using the Kubernetes Container Network Interface (CNI), allowing the deployment of a WireGuard VPN Server and a Wireguard VPN client in each Kubernetes node, isolating the network communication from the external and then preventing stole of data, as shown in Fig. 7. The default CNI of Kubernetes is flannel, which operates via an IPv4 overlay network. Each node in the cluster is associated with a dedicated subnet on which to internally allocate IP addresses. When a POD is started, the Docker bridging interface on each node allocates a dedicated address for each container. The PODs within a single host communicate through this bridge, while in the case of communication between PODs in different hosts, Flannel applies an "encapsulation" of the frames in UDP and performs routing to the correct destination. However, we have preferred to use a different CNI, such as Calico. Unlike Flannel, it does not stand out for its simplicity but for performance, reliability, and versatility. In fact, Calico's spectrum of action extends not only to the aspect of connectivity but also to security and network management. Calico does not use an overlay network but configures a layer 3 network using the BGP protocol for the correct routing of the packets (encapsulation is not required), with an evident performance gain as well as facilitation in case of debugging. Moreover, Calico implements a tunnel to secure the communication channel used by nodes. In this regard, it uses Elliptic Curve Cryptography (ECC), a type of public key cryptography based on elliptic curves defined on finite fields. This also satisfies the Objective O4.

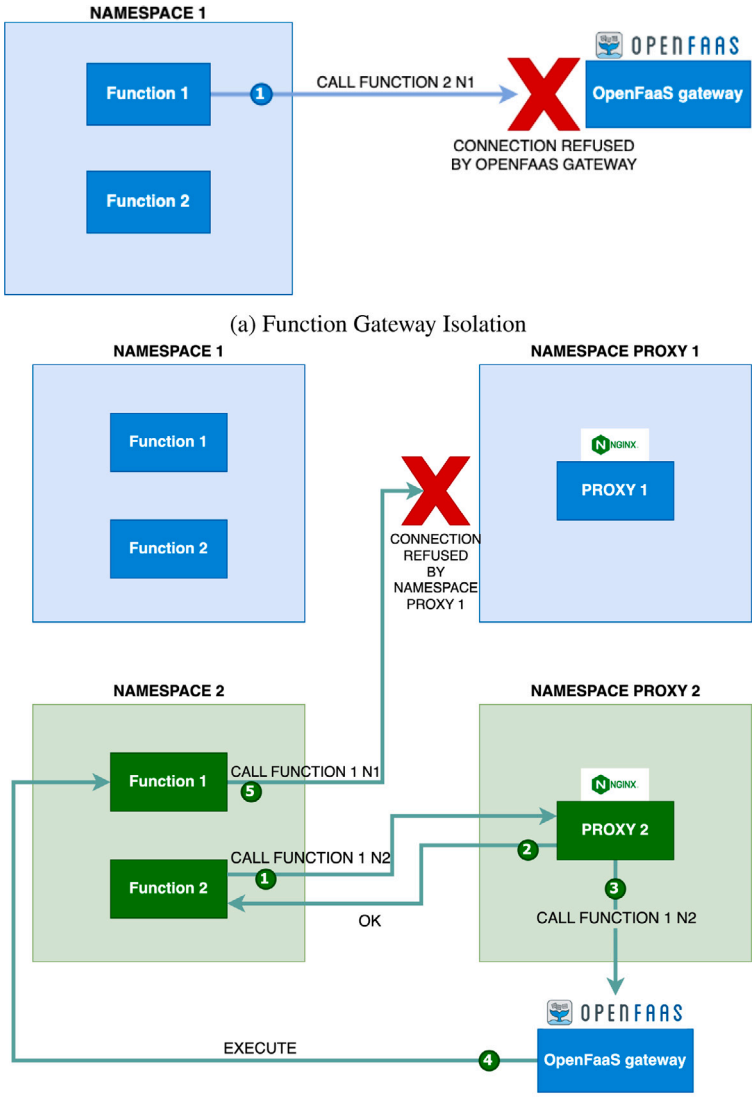


Fig. 5. Function isolation.

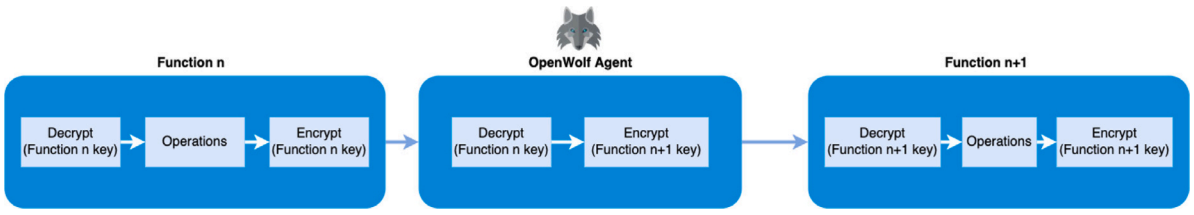


Fig. 6. Message encryption workflow.

5. Benchmarks

Experiments about the security-enhanced features implemented in our OpenWolf prototype are described in this Section. Compared to its original implementation, tested in a previous work [8], we expect to appreciate an overall performance deterioration (especially in the execution time) due to the encryption and decryption phases applied in both sequential and parallel workflows.

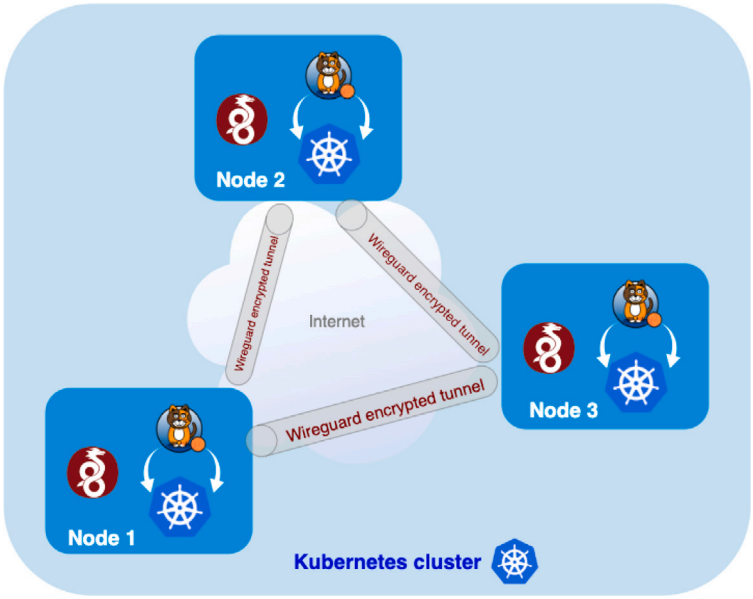


Fig. 7. SDMem VPN.

Table 1
Cluster's nodes characteristics.

Instances	Tier	Model	CPU	Memory	Operating system
1	Cloud	Openstack VM	Intel(R) Xeon(R) Gold 5218 CPU @ 2.30 GHz, 2-core	4 GB	Ubuntu 20
2	Edge	Raspberry Pi 4	ARM64 SoC 1.5 GHz, 4-core	4 GB	Raspberry OS ARM64

However, the security benefits are worth this drawback. In the following paragraphs, different performance metrics are analyzed and compared, such as the (i) Time to Respond (TTR) also called Execution Time, and the (ii) CPU and RAM usage, considering both a variable number of states and different key encryption lengths, in a full-Cloud, full-Edge, and Continuum environment in both sequential and parallel configurations.

5.1. System testbed

The Osmotic version of OpenWolf has been tested using a three-node Kubernetes Cluster, composed of one node in the Cloud tier, and two nodes in the Edge tier. The OpenFaaS's Gateway, Prometheus, Authentication Server, Kubernetes Master, OpenWolf Agent, OpenFaas' Nats, Queue Manager, and a Redis instance have been deployed in the Cloud environment, while the Edge is left empty, but available at hosting functions. Table 1 contains the information about our Continuum environment. Both Cloud and Edge are suitable to host all the OpenWolf and OpenFaas's components, as well as the functions that compose the workflows we used to make the test assessments. In the next sections, we compared the behavior of OpenWolf in terms of response time and resource utilization when deployed in three environments: full-Cloud, full-Edge, and continuum. In the full-Cloud test assessments, all the OpenWolf components as well as the functions that compose the workflows are deployed in the Cloud nodes, other tiers are still federated in the Kubernetes cluster, but they are idle. In the full-Edge environment, the opposite happens, with all the architecture components and functions deployed just in the Edge nodes. Finally, in the Continuum environment, the OpenWolf components are deployed in the Cloud, while the functions are fairly distributed among all the nodes.

The systems' characteristics are summarized in Table 1, while the OpenWolf parameters are summarized in Table 2.

5.2. Encryption overhead

The first test compares the execution time of a dummy function both in Cloud and Edge with and without encryption for input and output data. The encryption algorithm adopted for evaluation of the metric is the Advanced Encryption Standard (AES), a symmetric key block cipher algorithm. The block has a fixed size (i.e., 128 bits) and the key can be 128, 192, or 256 bits. The use of the AES algorithm is justified by the fact that it is used as a standard by the government of the United States of America and can

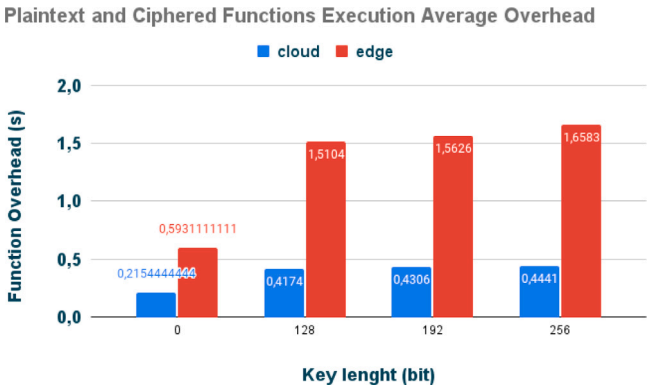


Fig. 8. Average serverless function execution overhead.

Table 2
OpenFaaS and OpenWolf parameters.

Parameter	Value	Condition
Queue Workers	1	Ever
Function replicas	State references	States Number $<$ 60
Function replicas	(State References)/2	States Number $\geq$ 60
Max_inflight	Equal to functions replicas	Ever

Table 3
Sequential in-clear execution time summary.

Tier/states	20	40	60	80	100
Cloud	5,095	9,323	14,427	19,743	25,719
Edge	12,658	25,554	38,694	52,256	65,196
Continuum	7,919	14,784	20,323	27,212	33,950

Table 4
Sequential ciphered execution time summary.

Tier/states	20	40	60	80	100
Cloud	9,317	19,038	28,352	38,6984	48,789
Edge	32,983	66,805	100,835	133,193	167,219
Continuum	20,807	41,974	63,647	85,143	107,262

Table 5
Parallel clear execution time summary.

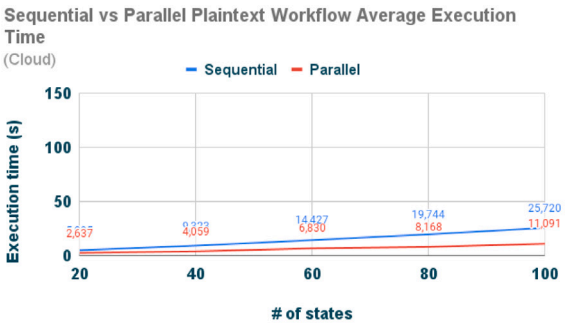
Tier/states	20	40	60	80	100
Cloud	2,637	4,059	6,830	8,168	11,091
Edge	5,307	9,505	14,820	18,493	23,451
Continuum	3,650	7,149	8,214	10,826	13,411

Table 6
Parallel ciphered execution time summary.

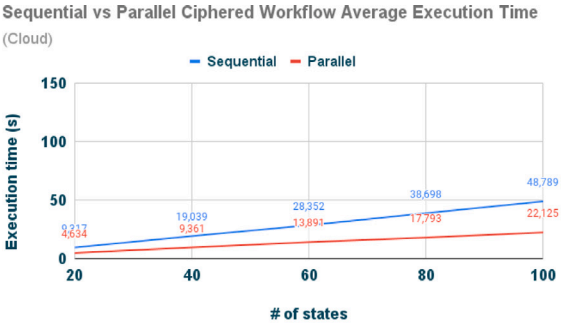
Tier/states	20	40	60	80	100
Cloud	4,634	9,361	13,891	17,793	22,125
Edge	13,902	25,446	40,067	49,961	66,601
Continuum	7,919	14,784	20,323	27,212	33,950

in fact be used to protect classified information. Specifically, a 128-bit key is sufficient for the SECRET level, while 192- or 256-bit keys are recommended for the Top secret level. AES makes 10 rounds for a 128-bit key, 12 rounds for a 192-bit key, and 14 rounds for a 256-bit key. Therefore, the tests were carried out considering an increasing key length (i.e., 128, 192, and 256-bit keys).

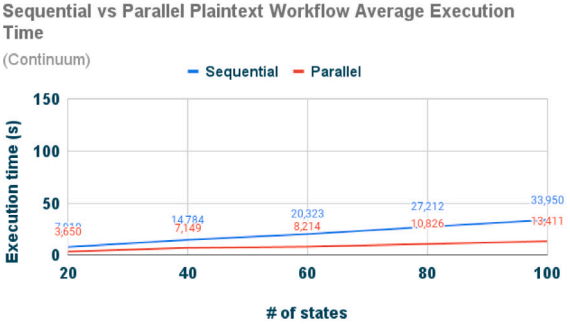
Analyzing the performance of the system when processing plaintext, as shown in Fig. 8, we evaluate that an Edge node requires 175% more to execute the same function with respect to the Cloud node. When cryptography is applied, the Edge execution time is around 260% times the Cloud one. This constant behavior is motivated by the fact that cryptography is a near-unit cost, independent



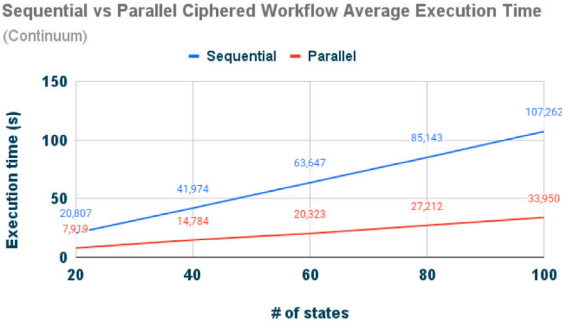
(a) Plaintext TTR Comparison in Cloud



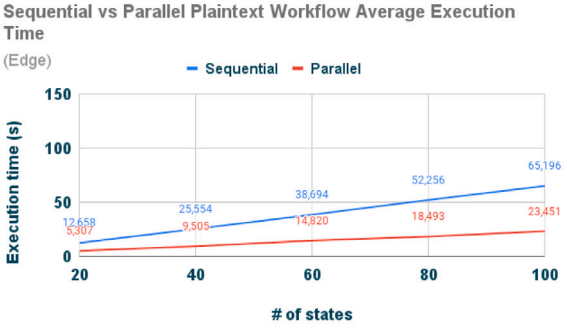
(b) Ciphertext TTR Comparison in Cloud



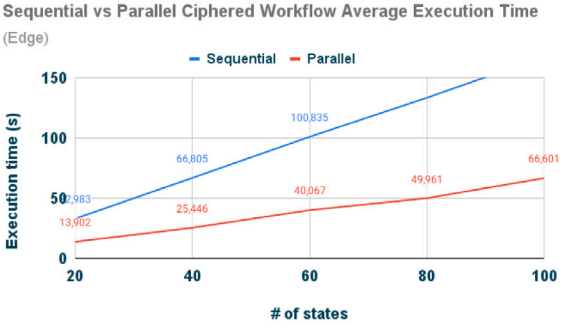
(c) Plaintext TTR Comparison in Continuum



(d) Ciphertext TTR Comparison in Continuum



(e) Plaintext TTR Comparison in Edge



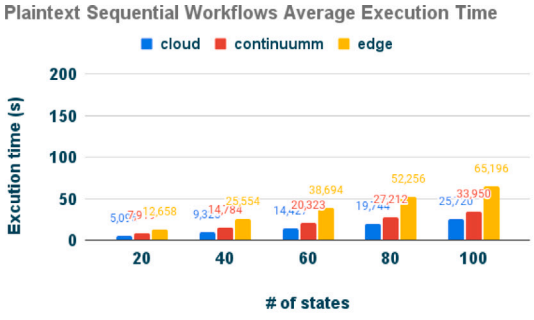
(f) Ciphertext TTR Comparison in Edge

Fig. 9. Sequential vs parallel execution comparison.

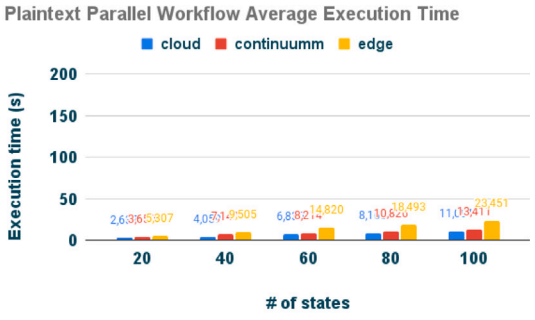
of the key dimension for a small range of keys. This is confirmed by the fact that Cloud and Edge encryption execution time takes respectively 100% and 260% more respect the plaintext execution, regardless of the key size.

5.3. Parallel and sequential workflow execution time

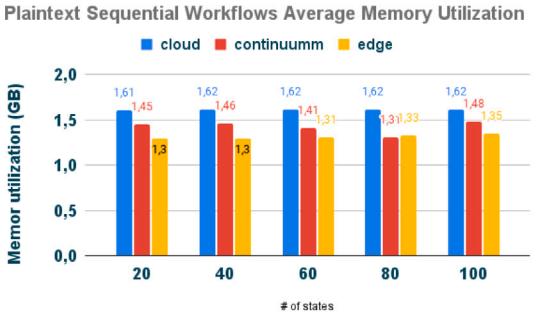
OpenWolf is able to build complex functions relationships thanks to the use of the Serverless Workflow DSL. The simplest serverless combination is chaining, where each workflow function follows the previous one. With respect to OpenFaaS and OpenWhisk, OpenWolf is also able to run parallel functions at once and then join them later. This possibility can be time-saving, as already demonstrated in the previous works, but the cipher capabilities could badly affect the scaling factor from the sequential to the parallel execution due to heavier resource work. In Fig. 9, we compare the execution time of the same workflow deployed both in sequential and parallel mode. In sub Figs. 9(a), 9(e), 9(e), all messages are exchanged in plaintext between functions. Instead, in sub Figs. 9(d), 9(f), 9(d), messages are encrypted using a 256-length key. Same results are even shown in the Tables 3–6. The



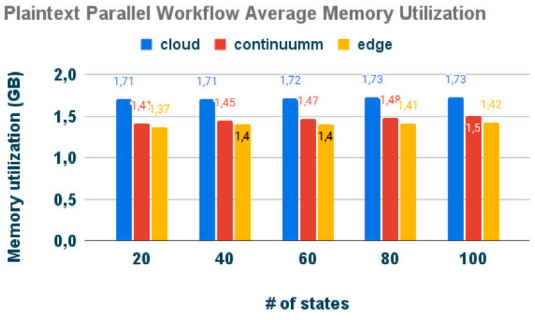
(a) TTR



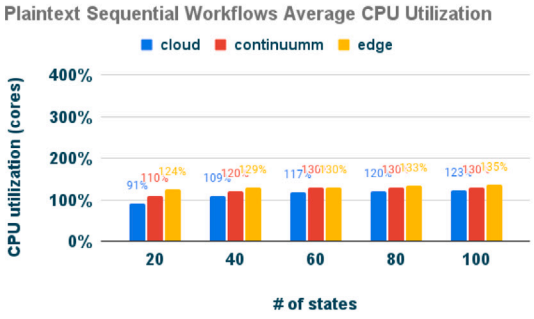
(b) TTR



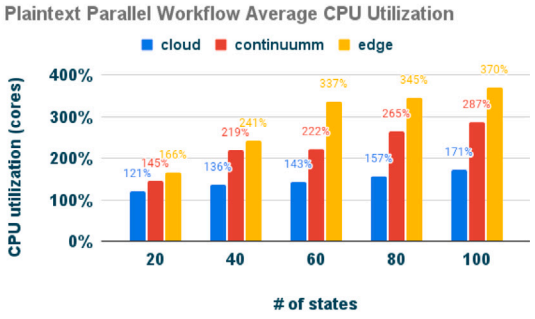
(c) Memory Usage



(d) Memory Usage



(e) CPU Usage



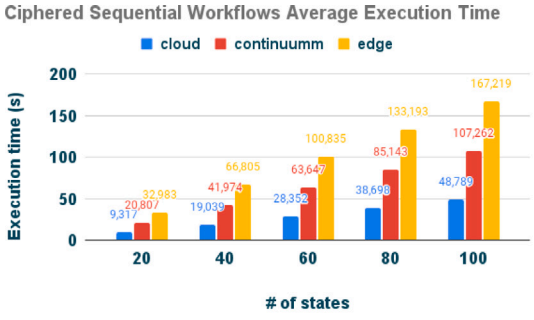
(f) CPU Usage

Fig. 10. Plaintext execution, sequential vs parallel workflow.

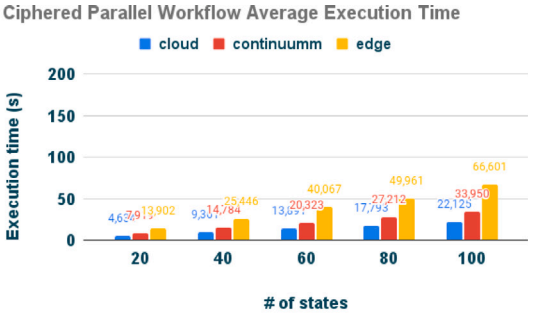
results obtained with this test show a good scaling factor on all three configurations. In the Cloud tier, a parallelized job can save from 48% of the time to 56% for a plaintext execution and from 50% to 54% for a ciphertext one. This result is obtained using two computing nodes. On the Edge, the improvements are similar to the Cloud. The plaintext execution saves from 54% to 64% of the time, while the ciphertext execution from 58% to 60%. Finally, thanks to the use of more computation nodes and a greater parallelization factor in the Continuum environment, a plaintext execution saves from 53% to 61%, while a ciphertext one from 62% to 68% of the time.

5.4. Resources utilization comparison

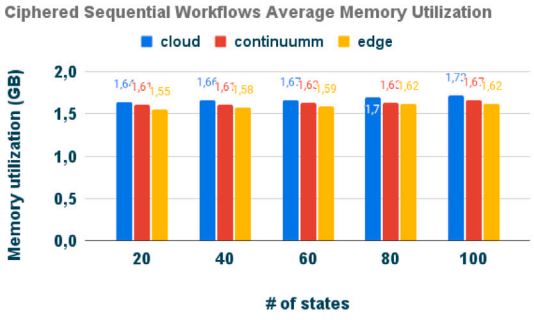
The main differences between Cloud and Edge are given by closeness to data, cost, and resource availability. In this test, we will focus on the latter factor to understand how CPU and RAM are affected in both environments when different kinds of workflows are run on. We compared OpenWolf with the cryptography feature enabled and disabled, using workflows composed of an increasing number of states. All states invoke the same function that is built to isolate OpenWolf overhead from the OpenFaas's container



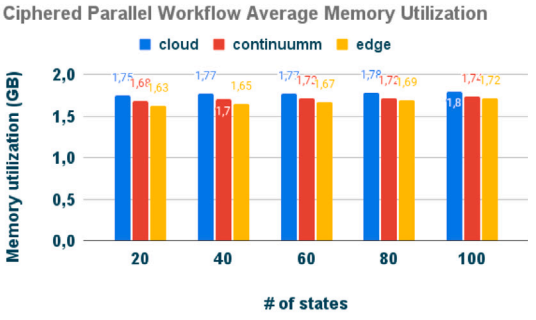
(a) TTR



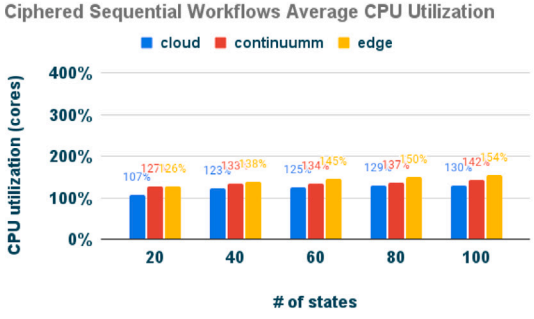
(b) TTR



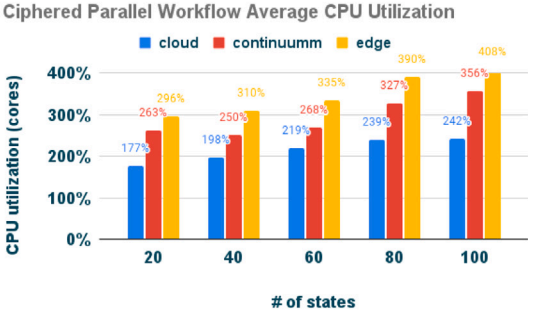
(c) Memory Usage



(d) Memory Usage



(e) CPU Usage



(f) CPU Usage

Fig. 11. Ciphered sequential and parallel workflow execution.

management overhead. In Fig. 10 we compared the metrics when a sequential and a parallel workflow are run in three different environments using plaintext data. Sub Figs. 10(a) and 10(b) confirm that the Cloud is better than the Edge in terms of execution time for a sequential and parallel workflow, as this performs 50% faster. The behavior at the Continuum is around the average point between Cloud and Edge, and this can be guaranteed by a good load balancing among the resources. As shown in sub Figs. 11(a) and 11(b), the difference is in the gap between Cloud and Edge. In fact, the Cloud is able to run up to 4x times faster than the Edge, which delay is related to the encryption.

In Figs. 10(e) and 10(f) we compare the CPU usage in the environments. Even in this case, Cloud is faster than Edge because of a higher clock in the CPU. On the other hand, the function parallelization involves all the cores of the Edge devices, increasing the overall usage. When encryption is taken into account, as shown in Sub Figs. 11(e) and 11(f), the differences between Edge and Cloud decrease. This happens because Cloud nodes can still increase their usage, while Edge nodes' utilization is already at its max. This also explains the difference in terms of execution time as commented before. In all the scenarios studied, the Continuum environment records good performances, load balancing CPU usage proportionally with the available nodes.

While we have seen better performances in terms of execution time and CPU utilization in the Cloud, the memory usage, as highlighted in Sub Figs. 10(c) and 10(d), is in the average 26% more used. Considering that just one function is run in each node and that OpenFaas does not allow a zero-scale, this value is on average equal for any execution. Such a difference is mainly led by resource-consuming containers when compiled for amd64 architecture [38] even if idle. This is especially true when compared with a container compiled for arm64 architectures.

6. Conclusions and future works

In this work, we tried to identify and address the security risks involved in a serverless environment over the Continuum. In particular, we highlighted five different threats that are: (i) the possibility to safely store secrets, (ii) the ability to access them in a safe and authorized way, (iii) the flow control which means the ability to allow some functions interactions and disallow others, (iv) the capacity to guarantee a safe communication channel over the internet, and finally (v) the capacity of guaranteeing message privacy. To address all these aspects, we followed the Osmotic Computing paradigm with the aim of providing a safe and balanced distributed environment in which running applications composed of many tasks called microservices and locating user and system data called MicroData. Osmotic Computing is safe-by-design because involves a security abstraction called Software Defined Membrane, which isolates applications and nodes. We dealt, therefore, with a real application of Osmotic Computing inside OpenWolf, a workflow engine able to spread and coordinate serverless functions deployed using OpenFaaS among the Cloud–Edge Continuum. In this regard, we deeply changed the OpenWolf architecture, modifying the underlying Kubernetes used both to build the federated network of Continuum and orchestrate the workflow functions. We performed several tests comparing the implemented features with an unsafe version of OpenWolf. In particular, we investigated metrics such as usage (i.e., CPU usage and memory) and time to satisfy different-size workflows when running both in sequence or parallel.

OpenWolf is still an in-progress work, so we aim to improve it in different aspects. For example, we plan to include a Manifest parser fully compliant with the Serverless Workflow DSL, in order to improve the expressivity of OpenWolf and its capacity to build complex function relations. Therefore, our intention is to improve the serverless integration that OpenWolf currently offers, for supporting not just OpenFaaS but even more open-source platforms.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] K. Djemame, M. Parker, D. Datsev, Open-source serverless architectures: An evaluation of Apache OpenWhisk, in: 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing, UCC, 2020, pp. 329–335, http://dx.doi.org/10.1109/UCC48980.2020.00052.

[2] N. Kaviani, D. Kalinin, M. Maximilien, Towards serverless as commodity: A case of knative, in: Proceedings of the 5th International Workshop on Serverless Computing, WOSC '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 13–18, http://dx.doi.org/10.1145/3366623.3368135.

[3] D. Balla, M. Maliosz, C. Simon, Open source FaaS performance aspects, in: 2020 43rd International Conference on Telecommunications and Signal Processing, TSP, 2020, pp. 358–364, http://dx.doi.org/10.1109/TSP49548.2020.9163456.

[4] S.K. Mohanty, G. Premsankar, M. Di Francesco, An evaluation of open source serverless computing frameworks, in: Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom, Vol. 2018-Decem, IEEE Computer Society, 2018, pp. 115–120, http://dx.doi.org/10.1109/CloudCom2018.2018.00033.

[5] P. Garcia Lopez, M. Sanchez-Artigas, G. Paris, D. Barcelona Pons, A. Ruiz Ollobarren, D. Arroyo Pinto, Comparison of FaaS orchestration systems, in: Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018, 2019, pp. 109–114, http://dx.doi.org/10.1109/UCC-Companion.2018.00049, arXiv:1807.11248.

[6] M. Ciavotta, D. Motterlini, M. Savi, A. Tundo, DFaaS: Decentralized function-as-a-service for federated edge computing, in: 2021 IEEE 10th International Conference on Cloud Networking, CloudNet, 2021, pp. 1–4, http://dx.doi.org/10.1109/CloudNet53349.2021.9657141.

[7] T. Pfandzelter, D. Bermbach, TinyFaaS: A lightweight faas platform for edge environments, in: 2020 IEEE International Conference on Fog Computing, ICFC, 2020, pp. 17–24, http://dx.doi.org/10.1109/ICFC49376.2020.00011.

[8] C. Sicari, L. Carnevale, A. Galletta, M. Villari, OpenWolf: A serverless workflow engine for native cloud-edge continuum, 2022.

[9] M. Villari, M. Fazio, S. Dustdar, O. Rana, R. Ranjan, Osmotic computing: A new paradigm for edge/cloud integration, IEEE Cloud Comput. (2016).

[10] T. Melamed, Interpretation for serverless, in: OWASP Top 10, 2017, URL https://owasp.org/www-pdf-archive/OWASP-Top-10-Serverless-Interpretation-en.pdf.

[11] N. Mateus-Coelho, M. Cruz-Cunha, Serverless service architectures and security minimals, in: 2022 10th International Symposium on Digital Forensics and Security, ISDFS, 2022, pp. 1–6, http://dx.doi.org/10.1109/ISDFS55398.2022.9800779.

[12] M. Wu, Z. Mi, Y. Xia, A survey on serverless computing and its implications for JointCloud computing, in: 2020 IEEE International Conference on Joint Cloud Computing, Vol. 18, 2020, http://dx.doi.org/10.1109/JCC49151.2020.00023.

[13] J.N. O.S., S. Mary Saira Bhanu, A survey on code injection attacks in mobile cloud computing environment, in: 2018 8th International Conference on Cloud Computing, Data Science & Engineering, Confluence, 2018, pp. 1–6, http://dx.doi.org/10.1109/CONFLUENCE.2018.8443032.

[14] M.A. Aleisa, A. Abuhussein, F.T. Sheldon, Access control in fog computing: Challenges and research agenda, IEEE Access 8 (2020) 83986–83999, http://dx.doi.org/10.1109/ACCESS.2020.2992460.

[15] V. Dehalwar, A. Kalam, M.L. Kolhe, A. Zayegh, Review of web-based information security threats in smart grid, in: 2017 7th International Conference on Power Systems, ICPS, 2017, pp. 849–853, http://dx.doi.org/10.1109/ICPES.2017.8387407.

[16] G. Mani, B.S. Rao, D.J.S. Kumar, C. Prasad, Distributed information flow control in serverless computing, in: 2022 4th International Conference on Smart Systems and Inventive Technology, ICSSIT, 2022, http://dx.doi.org/10.1109/ICSSIT53264.2022.9716444.
[17] M.-C. Nuno, M. Cruz-Cunha, Distributed information flow control in serverless computing, in: 2022 10th International Symposium on Digital Forensics and Security, ISDFS, 2022, http://dx.doi.org/10.1109/ISDFS55398.2022.9800779.
[18] V. Vallois, F. Guenane, A. Mehaoua, Reference architectures for security-by-design IoT: Comparative study, in: 2019 Fifth Conference on Mobile and Secure Services, MobiSecServ, 2019, pp. 1–6, http://dx.doi.org/10.1109/MOBISECSERV.2019.8686650.
[19] F.M. Awaysheh, M.N. Aladwan, M. Alazab, S. Alawadi, J.C. Cabaleiro, T.F. Pena, Security by design for big data frameworks over cloud computing, IEEE Trans. Eng. Manage. 69 (6) (2022) 3676–3693, http://dx.doi.org/10.1109/TEM.2020.3045661.
[20] T. Espinha Gasiba, U. Lechner, Raising secure coding awareness for software developers in the industry, in: 2019 IEEE 27th International Requirements Engineering Conference Workshops, REW, 2019, pp. 141–143, http://dx.doi.org/10.1109/REW.2019.00030.
[21] W. O'Meara, R.G. Lennon, Serverless computing security: Protecting application logic, in: 31st Irish Signals and Systems Conference, ISSC, 2020.
[22] X. Li, X. Leng, Y. Chen, Securing serverless computing: Challenges, solutions, and opportunities, in: IEEE Network, 2022, http://dx.doi.org/10.1109/MNET.005.2100335.
[23] A. Sabbioni, C. Mazzocca, M. Colajanni, R. Montanari, A. Corradi, A fully decentralized architecture for access control verification in serverless environments, in: 2022 IEEE Symposium on Computers and Communications, ISCC, 2022, http://dx.doi.org/10.1109/ISCC55528.2022.9912764.
[24] C. Sicari, A. Catalfamo, A. Galletta, M. Villari, A distributed peer to peer identity and access management for the osmotic computing, in: 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing, CCGrid, 2022, pp. 775–781, http://dx.doi.org/10.1109/CCGrid54584.2022.00091.
[25] M. Golec, R. Ozturac, Z. Pooranian, S.S. Gill, R. Buyya, IFaaSBus: A security- and privacy-based lightweight framework for serverless computing using IoT and machine learning, IEEE Trans. Ind. Inform. 18 (5) (2021) 3522–3529, http://dx.doi.org/10.1109/TII.2021.3095466.
[26] D. Ayed, P.-A. Dragan, E. Félix, Z.A. Mann, E. Salant, R. Seidl, A. Sidiropoulos, S. Taylor, R. Vitorino, Protecting sensitive data in the cloud-to-edge continuum: The FogProtect approach, in: 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing, CCGrid, 2022, http://dx.doi.org/10.1109/CCGrid54584.2022.00037.
[27] J. Barr, Firecracker lightweight virtualization for serverless computing, 2018.
[28] Google, Gvisor: A container sandbox runtime focused on security efficiency and ease of use, 2019.
[29] C. Cicconetti, M. Conti, A. Passarella, On realizing stateful FaaS in serverless edge networks: State propagation, in: 2021 IEEE International Conference on Smart Computing, SMARTCOMP, 2021, pp. 89–96, http://dx.doi.org/10.1109/SMARTCOMP52413.2021.00033.
[30] H. Javed, A.N. Toosi, M.S. Aslanpour, Serverless platforms on the edge: A performance analysis, in: R. Buyya, L. Garg, G. Fortino, S. Misra (Eds.), New Frontiers in Cloud Computing and Internet of Things, Springer International Publishing, Cham, 2022, pp. 165–184, http://dx.doi.org/10.1007/978-3-031-05528-7_6.
[31] M.S. Aslanpour, A.N. Toosi, M.A. Cheema, R. Gaire, Energy-aware resource scheduling for serverless edge computing, in: 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing, CCGrid, 2022, pp. 190–199, http://dx.doi.org/10.1109/CCGrid54584.2022.00028.
[32] M. Nguyen, S. Debroy, P. Calyam, Z. Lyu, T. Joshi, Multi-cloud performance and security-driven brokering for bioinformatics workflows, in: 2019 IEEE 27th International Conference on Network Protocols, ICNP, 2019, http://dx.doi.org/10.1109/ICNP.2019.8888071.
[33] G.L. Stavrinides, H.D. Karatza, Security and cost aware scheduling of real-time IoT workflows in a mist computing environment, in: 2021 8th International Conference on Future Internet of Things and Cloud, FiCloud, 2021, http://dx.doi.org/10.1109/FiCloud49777.2021.00013.
[34] L. Carnevale, A. Celesti, A. Galletta, S. Dustdar, M. Villari, From the cloud to edge and IoT: A smart orchestration architecture for enabling osmotic computing, 2018.
[35] M. Villari, M. Fazio, S. Dustdar, O. Rana, L. Chen, R. Ranjan, Software defined membrane: Policy-driven edge and Internet of things security, IEEE Cloud Comput. 4 (2017).
[36] C. Sicari, A. Galletta, A. Celesti, M. Fazio, M. Villari, An osmotic computing enabled domain naming system (OCE-DNS) for distributed service relocation between cloud and edge, Comput. Electr. Eng. 96 (2021) 107578, http://dx.doi.org/10.1016/j.compeleceng.2021.107578, URL https://www.sciencedirect.com/science/article/pii/S0045790621005176.
[37] A. Buzachis, M. Villari, Basic principles of osmotic computing: Secure and dependable MicroElements (MELs) orchestration leveraging blockchain facilities, in: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion, 2018.
[38] Y. Liu, D. Lan, Z. Pang, M. Karlsson, S. Gong, Performance evaluation of containerization in edge-cloud computing stacks for industrial applications: A client perspective, IEEE Open J. Ind. Electron. Soc. 2 (2021) 153–168, http://dx.doi.org/10.1109/OJIES.2021.3055901.