Serverless Confidential Containers: Challenges and Opportunities

Carlos Segarra*
cs1620@ic.ac.uk
Imperial College London
London, United Kingdom

Tobin Feldman-Fitzthum
tobin@ibm.com
IBM Research
Yorktown Heights, US

Daniele Buono
dbuono@us.ibm.com
IBM Research
Yorktown Heights, US

Peter Pietzuch
prp@imperial.ac.uk
Imperial College London
London, United Kingdom

ABSTRACT

Serverless computing allows users to execute pieces of code (so called functions) on-demand in the cloud without having to provision any hardware resources. However, by executing in the cloud and delegating control over hardware resources, the integrity of the execution and the confidentiality of function code and data are at the mercy of the cloud provider and serverless runtime. Confidential computing aims to remove trust from the cloud provider by executing applications inside hardware enclaves. In spite of the increasing adoption of confidential computing, designing a confidential serverless runtime with moderate performance overhead remains an open challenge.

In this short article we present our experience porting the Knative serverless runtime to a confidential setting using Confidential Containers (CoCo), a technology that allows the execution of unmodified (encrypted) container images inside confidential VMs (cVMs). Our results show that cVMs are not ready to execute container-based serverless functions. Starting a serverless function in a CoCo from an encrypted container image with attestation takes up to 17 seconds. Starting 16 serverless functions concurrently takes more than three minutes, 20× slower than its non-confidential counterpart. We analyze the main sources of overhead, and outline the research challenges to bridge the gap between confidential and serverless computing.

KEYWORDS

confidential computing, serverless, confidential serverless, confidential virtual machines, knative

ACM Reference Format:
Carlos Segarra, Tobin Feldman-Fitzthum, Daniele Buono, and Peter Pietzuch. 2024. Serverless Confidential Containers: Challenges and Opportunities. In *The 2nd Workshop on SErverless Systems, Applications and MEthodologies (SESAME '24), April 22, 2024, Athens, Greece.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3642977.3652097

*Work partially done whilst visiting IBM Research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SESAME '24, April 22, 2024, Athens,Greece
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0545-8/24/04
https://doi.org/10.1145/3642977.3652097

Baseline (sandbox)	Cold Start	Warm Start	Scale Out (0→16)
runc (container) [39]	6 s	1 s	16 s
Kata (VM) [34]	7 s	2 s	17 s
CC-Knative (cVM)	17.5 s	17.5 s	190 s
CC-Knative/ Kata	**2.5 ×**	**8.5 ×**	**11 ×**

Table 1: Cold start, warm start, and scale-out times of a simple Knative service using different sandboxes on K8s (Fig. 3).

1 INTRODUCTION

Serverless computing allows the execution of user-defined pieces of code, so-called serverless functions, in cloud-owned hardware resources. Serverless users register functions with a serverless runtime [63, 74, 87], configure what events may trigger function execution, and are only billed for the time their functions are running. In turn, the serverless runtime provisions the execution environment for functions to execute in, scaling it up or down in response to event load, transparently to the user. This separation of concerns between function definition (serverless user), and cloud-resources' provisioning (serverless runtime), makes serverless easy to adopt for users, and gives runtimes further opportunities to optimize hardware efficiency and usage. This has made serverless computing an ever-increasing cloud solution attracting interest from research [6, 19, 58, 69, 90, 100] and industry [2, 3, 18, 60, 84, 89] alike.

Years of serverless in production have allowed research to characterize both what kind of functions users define, and how they want to define and execute them. First, traces from different production clusters [60, 89] indicate that serverless functions are short-lived and bursty. Second, insights from leading serverless offerings [18] indicate that users want to define functions as OCI images [40] and isolate them using virtual machines (VMs) [22]. The former has motivated the key performance metrics in serverless: cold-starts, warm-starts, and scale-up latency. The latter has motivated the development of lighter-weight VMs [3] and fast container image provisioning and start-up [18, 75]. In this work we assume a serverless environment where functions are defined using OCI images, and isolated using VMs, as increasingly adopted by the leading cloud providers [18, 74].

The increasing popularity of the cloud to store and process data in favour of on-premise clusters has raised concerns in terms of data confidentiality and execution integrity [1, 20, 23, 72]. These concerns are exacerbated in a serverless context, where users do not have any control over the exeuction of their functions, as it is delegated to the serverless runtime. Confidential computing [23] aims to make cloud computing more trustworthy by allowing the creation of hardware-backed trusted execution environments (so-called enclaves) [7, 12, 54, 56] in otherwise untrusted cloud-owned servers. The security and trustworthiness of enclaves relies on three pillars [62]: (i) the confidentiality of a component within the cloud server (called the root-of-trust, RoT), (ii) the integrity and well-functioning of the CPU chip, and (iii) the ability to remotely attest the identity and integrity of the RoT and the CPU chip. If all these conditions meet, confidential computing can be used to protect the confidentiality and integrity of a variety of cloud workloads [14, 21, 33] and is being used by leading corporations and governments worldwide [20, 72].

Unfortunately, the requirements for confidential computing are at odds with serverless. First, the first-iteration of hardware enclaves, process-based enclaves, as provided by Intel SGX [54] and Arm TrustZone [12], required application re-write or offered very limited compatibility with *lift-and-shift* approaches [43, 46]. None of which capable of running arbitrary OCI images. Second, the security of enclaves comes at the cost of (considerably) longer start-up times and (usually) interactive remote attestation protocols, both negatively impacting cold-start times. Lastly, integrity protection against rollback [86] and TOCTTOU [99] attacks prevent the adoption of sandbox pre-warming [69] or sandbox re-use [6], both common techniques to speed up serverless execution. These limitations mean that, other than some academic efforts [5, 17, 45, 53, 96], confidential serverless remains to see widespread adoption.

In this short paper we present our experience building CC-Knative, a port of the Knative [63] serverless runtime to Confidential Containers (CoCo) [33] (§3). CoCo allows the execution of unmodified (encrypted) OCI images inside VM-based enclaves (so-called confidential VMs, cVMs) as provided by AMD SEV(-SNP) [7] and Intel TDX [56]. Our results are summarized in Tab. 1 and expanded in §4. Compared to Kata Containers [34], a container runtime that transparently runs K8s pods inside VMs, starting an encrypted and attested container in a cVM adds 10 and 14 seconds to the cold and warm start times respectively. Even more, scaling from 0 to 16 containers is three minutes (20×) slower. These results show that cVMs are not ready to execute container-based serverless functions. We identify two key research challenges to bridge the gap between serverless and confidential computing (§5): (i) how can we securely pre-warm and re-use cVMs? and (ii) how can we efficiently use and share encrypted container images?

2 CONFIDENTIAL SERVERLESS COMPUTING

2.1 The Need for Confidential Serverless

Serverless computing has seen widespread adoption in a variety of domains like image processing [44], machine learning inference [57], linear algebra [59], life sciences research [10] and transaction fraud detection [13], among many others. Growing concerns about data privacy in the cloud [20, 72], government-sponsored mass surveilance programs [78], and increasing data protection regulation [1], have motivated the adoption of confidential computing in many cloud services like analytics [102], databases [15], and general container services [14].

However, confidential computing has yet to see widespread adoption in the serverless community. This is because the performance overheads (particularly bootstrapping overheads) are at direct odds with serverless cold starts. In addition, serverless zero-ops deployment model seems to come at the expense of code and data confidentiality and execution integrity: if you want the serverles runtime to control the whole hardware provisioning and execution lifecycle of your function, you must accept that the serverless runtime will know, at least, the contents of the function that it is executing. We refute this statement and claim that it is possible to build a serverless runtime that is completely oblivious of the functions that it is executing and can not tamper with neither the hardware that it is provisioning, nor the data that is being processed (§3). This design, comes at a performance cost (Tab. 1). We breakdown the main sources of overhead and hint at the solutions that we plan on exploring in future work (§4).

Our threat model is shared with that of confidential computing [55] and confidential containers in particular [50]. We do not trust the host operating system and virtual machine monitor (VMM) neither any other co-located processes or virtual machines. We assume that any distrusted software may want to break the confidentiality of the serverless function code, the confidentiality of the data being processed, or the integrity of the execution results. Most importantly, in a serverless context, this means that we do not trust neither the serverless runtime (e.g. Knative [63]) nor the orchestration framework (e.g. K8s) nor the cloud provider's software stack (e.g. IBM Cloud [51]). Any physical, side-channel, or availability attacks outside of the scope of the confidential containers' threat model is also out of the scope in this work.

2.2 Design Space for Confidential Serverless

There are different design options for a confidential serverless runtime [49]. Most existing work [5, 17, 45, 53, 96] falls in the category of what we call *process-centric* security. These runtimes use enclaves to execute specific parts of the function's code, or *lift-and-shift* functions inside process-based enclaves. The former requires application re-write, and the latter offers very limited compatibility with arbitrary functions, let alone containers. However, these approaches come at the smallest Trusted Computing Base (TCB) size.

On the other end of the spectrum in terms of TCB size, we find runtimes that use *node-centric* security [94]. These runtimes run the whole K8s node inside a cVM, and all K8s nodes belong to the same trust domain. Inasmuch as this approach limits the data breaches outside of the K8s cluster, it is a poor fit for serverless where different serverless users distrust each other, and the serverless runtime. The TCB for these runtimes comprises the K8s control and data plane, as well as the host and guest's software stack.

In this work, we opt for a middle-ground: *pod-centric* security [29]. In this design, each K8s pod runs in a different cVM, providing strong isolation from different pods, and from the K8s control plane. This is the design chosen both by Kata Containers [34] and Confiential Containers [33]. CC-Knative builds on these two projects. The pod-centric approach is very similar to a

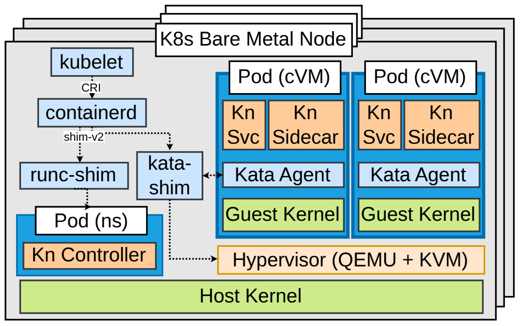


Figure 1: Block diagram of a K8s cluster using CC-Knative.

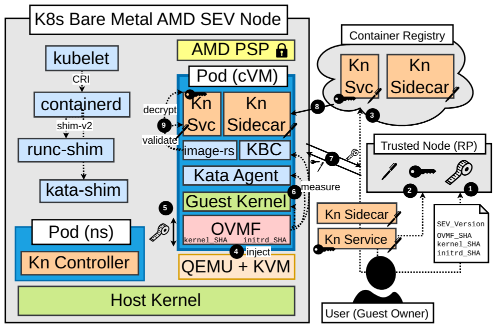


Figure 2: Steps to attest a Knative Service in CC-Knative.

container-centric approach, whereby each container runs in a different cVM. The pod-centric approach is better suited for a K8s-native environment, thus a better choice to port the Knative serverless runtime.

3 CC-KNATIVE: KNATIVE ON COCO

In this section we present CC-Knative, our prototype to run encrypted Knative services inside cVMs with CoCo. We first describe how CC-Knative seamlessly integrates in an existing K8s cluster (§3.1) and then move on to describe how we can establish trust on CC-Knative using remote attestation (§3.2). The implementation efforts required to put everything together are described in §4.

3.1 Integration with K8s

Knative (Kn) is a K8s-native serverless runtime [63]. A Knative deployment consists of a control-plane component, called the `Kn Controller`, and a set of custom resources (CRDs in K8s terminology) [64]. Serverless functions in Knative are called `(Kn) Services`, and are identified by an OCI image digest. Knative resolves image tags to image digests to guarantee execution integrity and immutability [65]. For monitoring, network programming, and observability, Knative deploys a sidecar container in the same pod where the service executes [64]. Pods in K8s share network and storage namespaces, and are also the trust boundary in CC-Knative as we opted for a pod-centric approach §2.2. Therefore, the sidecar component will be inside CC-Knative's TCB.

Fig. 1 depicts the high-level architecture of a CC-Knative deployment. Even though CC-Knative integrates transparently with upstream K8s, it can not be deployed in any K8s cluster. In particular, CC-Knative can not be deployed in a virtual K8s cluster where K8s nodes are running inside VMs. This is because CC-Knative heavily relies on the host's virtualization stack, and nested confidential virtualization is only available in a very limited form, in closed-source corporate environments [73]. CC-Knative will benefit from any improvements on nested (confidential) virtualization.

In a non-confidential Knative deployment, the local `kubelet` [67] process talks with a working container runtime over the Container Runtime Interface (CRI) [66]. In CC-Knative, our container runtime of choice is `containerd` [24]. This is because `containerd` is very flexible and modular in itself, and supports concurrently executing pods with different runtimes as long as they implement the `shim-v2` protocol [25]. When deploying a non-confidential pod (e.g. the Knative controller), `containerd` will execute containers directly in the host environment using `runc` [39] and isolate different pods using namespaces [70]. When deploying a confidential pod (by just adding an annotation to the pod manifest) `containerd` will use the `kata-shim` [37] to execute containers inside a cVM, and execute different pods in different cVMs.

To start a cVM, the `kata-shim` can use a variety of hypervisors. The choice of hypervisor and cVM software stack is tightly coupled to the attestation protocol (§3.2). In this section it suffices to say that CC-Knative uses QEMU [79] (with KVM [68]) as hypervisor, and uses a virtual firmware image to boot a Linux guest inside the cVM. Inside the guest, the `Kata Agent` [35] runs as the `/init` process [11], and communicates with the `kata-shim` over a vSocket [98] to make the pod inside the cVM look like a regular pod. This vSocket interface is the main attack vector into the cVM and it is, as a consequence, highly constrained [38].

3.2 Starting a Knative Service in CC-Knative

To guarantee the confidentiality and integrity of Knative services and the data therein processed we need verifiable evidence that the system in Fig. 1 has been correctly bootstrapped and has not been tampered with. Remote attestation [48] is a cryptographic protocol through which a relying party (RP) can gain confidence of the identity and integrity of a piece of software running in an untrusted environment. For liveness purposes, remote attestation is generally an interactive process between the RP and the software to be attested. Given that serverless function's execution is triggered by different events (non-interactively w.r.t the user), in CC-Knative we delegate the role of RP to an external, always available, trusted node. The implementation and availability of this trusted RP node is out of the scope of this work, and we refer to the relevant related work [71]. In CC-Knative we heavily leverage the attestation infrastructure already in place for CoCo [28], and the trusted RP node runs CoCo's Key Broker Service (KBS) [30].

The protocol to securely start Knative services in CC-Knative is depicted in Fig. 2. The protocol is specific to the current CC-Knative prototype that uses AMD SEV as cVM technology of choice. A relatively similar protocol would apply for AMD SEV-SNP or Intel TDX (with some caveats §5, §4.3), both supported in CoCo [29]. Ahead-of-time, the user registering the serverless function generates a *launch measurement* [7] of the cVM (❶). This

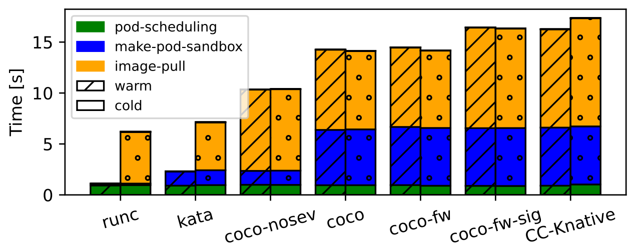


Figure 3: Start-up latency for a Knative service.

launch measurement includes the SEV version, together with the hash digests of: the virtual firmware (OVMF [95]) image, the kernel, the kernel command line, and the `initrd`. The launch measurement can be generated in a non-SEV machine, with open source tools [97]. Then, the user can encrypt the layers of the service image (using for example `skopeo` [27] and `ocicrypt` [26]), and sign both the service and sidecar image for integrity (using for example `cosign` [91]). The launch measurement and encryption and signing keys are then stored in the trusted RP (❷) and the encrypted and signed container images in any compatible container registry (for example GHCR [82], or Quay [81], ❸).

When the Knative controller triggers the execution of a Knative Service, QEMU starts the cVM. In the QEMU command, QEMU presents the OVMF image as a flash drive in `flash0`. When placing the OVMF image in memory, QEMU injects the hash digest of the kernel, kernel command line, and `initrd` in the OVMF image itself (❹). This mechanism [83] is crucial to the attestation protocol, and was upstreamed to QEMU [16] and OVMF [77] by the CoCo developers. Once the digests are injected, AMD's PSP measures and encrypts the memory pages for the cVM [7], which at this point only include the patched OVMF image (❺). Then, during OVMF boot, QEMU sends OVMF the blobs for the kernel, kernel command line, and `initrd` using the `fw_cfg` interface [80]. Before transferring control to the kernel, OVMF measures the given blobs against the expected digests ❻. If QEMU provides a malicious blob, the measurement fails and cVM boot aborts. Similarly, if QEMU injects the digest of the malicious blob during ❹, the measurement in ❺ fails, aborting cVM boot too.

The software to interact with the `kata-shim` (the Kata Agent), to interact with the trusted RP node (the Key Broker Client, KBC [32]), and to fetch and decrypt the container images (`image-rs` [31]), is distributed inside the `initrd`. As a consequence, its' integrity is also validated in ❻. The guest runs an SEV-enlightened Linux kernel, uses the Kata Agent as `/init` process, and boots directly from the `initrd` (without a bootloader). To fetch the encryption and signature keys from ❷, the KBC presents the trusted RP node the launch measurement from ❺ signed by the PSP (❼). Finally, `image-rs` fetches the encrypted and signed container images (❽) and validates the signatures and decrypts the image layers (❾).

4 EVALUATION

Motivated by Tab. 1, in this section we aim to answer the following questions: (i) what are the different contributors to CC-Knative's overhead? (§4.1)? Particularly in terms of (ii) cold starts (§4.2), (iii) warm starts (§4.3), and (iv) scale-out latency? (§4.4)

Evaluation Set-Up. We run all experiments on a single-node Kubernetes (`v1.28.2`) cluster. Our node is a bare metal server on IBM Cloud [52] with an AMD EPYC 7763 (Milan architecture) CPU, with 128 cores and SEV enabled. We use CoCo version `v0.7.0` and Knative `v1.11.0`. The CoCo version pins the `containerd`, Kata Containers, and guest kernel version. For our OVMF patches we use OVMF version `edk2-stable202302`.

Implementation. To implement CC-Knative we had to configure the Knative controller to spawn service pods in confidential containers. We also had to configure the KBS to verify the integrity of the sidecar, and the integrity and confidentiality of the service pod. Lastly, we had to patch the Kata Agent to correctly resolve Knative services from image digests. All the required patches and evaluation scripts are open-source and available at: *removed for anonymity.*

Baselines. We compare CC-Knative with a variety of baselines and with different security features disabled. As non-confidential baselines we use `runc`, the default sandbox in Knative, that executes different services as different containers in the host, and `Kata` that executes different pods in different VMs. We also compare against CC-Knative with no attestation at all (`coco`), with only launch measurement attestation (`coco-fw`), also adding image signature (`coco-fw-sig`), and lastly also adding image encryption (`coco-fw-sig-enc`). This last baseline is equivalent to CC-Knative. In addition, we also configure a baseline to use CC-Knative with non-SEV VMs (`coco-nosev`), and the same baseline with OVMF (non-SEV guests in Kata use SeaBIOS [85] by default, `coco-nosev-ovmf`).

Metrics. We are interested in measuring the key metrics that we identified for serverless runtimes: cold starts, warm starts, and scale up latency. As a consequence, we do not study the performance overheads of using cVMs to execute container images. We refer to the relevant related work [4, 8, 9]. For all experiments we use a very simple Knative Service, a Python "Hello World", distributed as part of Knative's demos [42].

4.1 End-to-end Start-Up Latency

Fig. 3 summarises the end-to-end latency to start a Knative service for the first time (cold-starts, bars with dots), and subsequent times (warm-starts, bars with diagonal lines). Compared to `Kata`, using CC-Knative adds 10 and 14 extra seconds to cold and warm starts respectively.

To sanity check the results, the time to create a VM (`make-pod-sandbox`) for non-SEV guests (`kata`, `coco-nosev`) is roughly the same, around 1 second. Similarly, the time to pull the container images (`image-pull`) for baselines that do do not pull images inside the guest (i.e. do not use `image-rs`, `runc` and `kata`) is roughly the same, around 5 seconds. Lastly, the time to schedule the pod (`pod-scheduling`) is roughly the same for all baselines and we can ignore it from now onwards.

The results in Fig. 3 rise three different questions. In terms of cold starts (§4.2), (i) why is cVM start-up 4 seconds slower than VM start-up? (§4.2.1) (ii) why is image pulling inside the guest 3-5 seconds

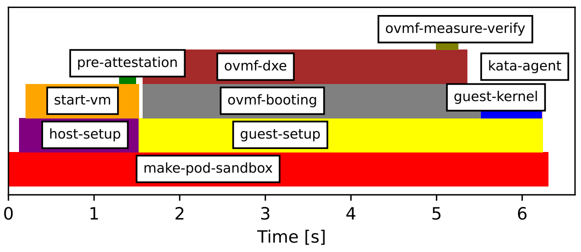


Figure 4: Flame graph of the time spent booting a cVM.

slower than host-side pulling? (§4.2.2). In terms of warm starts, (iii) why are there no warm starts for cVM-based baselines? (§4.3)

4.2 Cold Starts

4.2.1 cVM Start-Up Overhead. To break down the overheads of starting a cVM, we patch OVMF to log performance counter values and use these to create a flame graph [47] of the `make-pod-sandbox` bar in Fig. 3 for the CC-Knative baseline. We present the flame graph in Fig. 4.

We differentiate between `host-setup`, the time spent preparing the cVM on the host, and `guest-setup`, the time spent from OVMF start-up to the Kata Agent being responsive. Within the host set up, most time is spent in the `start-vm` task, which corresponds to the time spent executing the QEMU command. We can also see that the time spent waiting for the PSP in `pre-attestation` is comparatively low. Within the guest set up, we spend 4 seconds booting OVMF, and 0.7 seconds booting Linux (the time to start the Kata Agent is negligible). Inside OVMF, we spend most time initializing the driver's execution environment (DXE phase in the PEI specification [93]). In particular, the time spent measuring the blobs provided by QEMU through the `fw_cfg` interface (`ovmf-measure-verify`) is also comparatively low.

Intuitively, it seems that we spend a lot of time executing virtual firmware in OVMF. However, it is hard to confirm this hypothesis without comparing the flame graphs of different baselines. To this extent, in Fig. 5 we include three flame graph comparisons for different baselines. The top flame graph always corresponds to CC-Knative, and the bottom one to the compared-to baseline. If we compare the flame graph of CC-Knative to `Kata` (Fig. 5a), we observe that: (i) the `start-vm` task is slowed down by 10× and (ii) the `ovmf-booting` task is slowed down by 50-100×.

The `start-vm` task corresponds to the time spent executing the QEMU command. When provisioning an SEV guest (non-SNP), QEMU must provision (and pin) *all* memory pages assigned to the guest upfront [4, 61]. Thus, our hypothesis is that the slowdown in (i) is due to the memory page provisioning for SEV guests. To confirm this hypothesis, we plot the same start-up latency as we increase the guest's memory size (i.e. QEMU's `-m` flag [76]) in Fig. 6a. As we expected, the latency increases for SEV guests as we increase the memory size. We take the top-most, right-most, point in Fig. 6a (CC-Knative-128 GB memory size), and compare the flame graph with the original CC-Knative flame graph (with the default 2 GB guest memory size) in Fig. 5b. As we expected again, all the difference in the flame graphs correspond to the `start-vm` task. This confirms our hypothesis that the slowdown in the `start-vm` task corresponds to the SEV guest memory provisioning mechanism.

The `ovmf-booting` task corresponds to the time spent executing the virtual firmware binary. The slowdown reported in Fig. 5a is not a fair one, the `Kata` baseline uses SeaBIOS [85] as virtual firmware, whereas CC-Knative uses OVMF. For an apples-to-apples comparison, we patch the `Kata` baseline to use OVMF instead, and report the flame graph comparison in Fig. 5c. Still, virtual firmware execution is 4-5× slower. The only noticeable difference between CC-Knative's OVMF package and `Kata`'s is the measurement and verification of the kernel, kernel command line, and `initrd` memory blobs passed by QEMU via the `fw_cfg` interface. To check whether this measurement is the source of overhead, we measure the start-up latency as we increase the size of the `initrd` in Fig. 6b. Unlike before, the start-up latency does not seem to be greatly influenced by the `initrd` size. We then add additional monitoring to OVMF, and present a zoomed in flame graph comparison of CC-Knative's OVMF execution and `Kata`'s with OVMF in Fig. 6c (left). In addition, and for further clarification, we plot a per-event slowdown in Fig. 6c (right). The blob measurement and verification is now part of the `dxe-dispatch` task that is only slowed down by a factor of 2×. On the other hand, the `dxe-ctors` task is slowed down by a factor of 20×. This is unexpected, likely a bug in SEV's OVMF package, and something that we are looking into fixing.

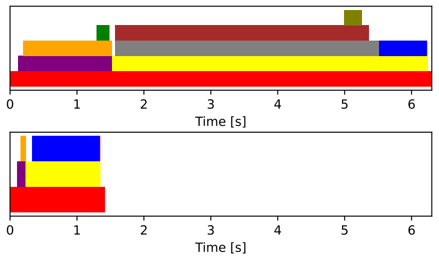
In summary, starting a cVM is 4 seconds slower than starting a VM. Provisioning all guest memory pages upfront introduces 1 second for a 2 GB guest and the remaining 3 seconds come from an unaccounted slowdown in the OVMF DXE constructor phase.

4.2.2 Image Pulling Overhead. Fig. 3 reports that pulling the images for both the Knative service and Knative sidecar is between 3-5 seconds slower when pulling inside the guest (using `image-rs`) than when pulling from the host (`runc` and `kata` baselines). The variability between guest-side pulling baselines is to be expected: some baselines just pull regular images (`coco`) whereas others pull signed and encrypted images (CC-Knative). But even for the baselines that pull unencrypted and unsigned images, image pulling is still 3 seconds slower.

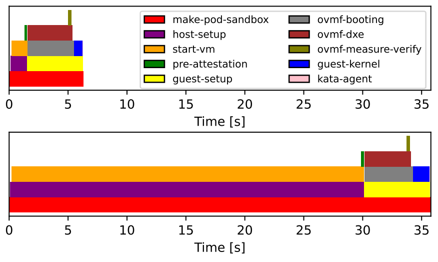
In Fig. 7a we plot the combined flame graph of pulling both container images in CC-Knative. The first thing we notice is that images are pulled in serial, whereas for the host-side pulling baselines they are pulled in parallel. This is because when forwarding `Kata`'s `PullImage` API call [36] from the host to the guest it becomes blocking (whereas normally it would not block). The second thing we notice is that fetching image layer bytes over the internet (`pull-single-layer` task) is roughly the same for both images. However, processing these bytes (i.e. decompressing for the sidecar, or decompressing and decrypting for the service, `handle-single-layer` task) takes very different amounts of time. This makes us conclude that the biggest source of overhead in pulling a single encrypted image is decrypting image layers.

4.3 Warm Starts

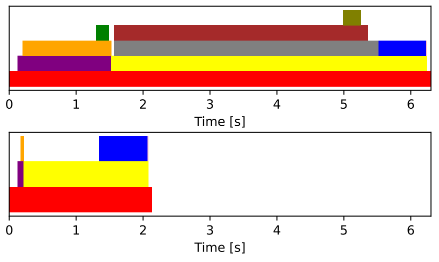
Fig. 3 reports that there is no difference bewteen cold and warm starts in CC-Knative. This is true. Right now, none of the work involved in executing a Knative service in CC-Knative can be re-used if the same function is invoked again shortly after. As a



(a) CC-Knative (top) vs Kata (bottom)

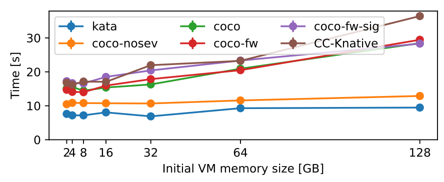


(b) CC-Knative 2 GB (top) vs 128 GB (bottom)

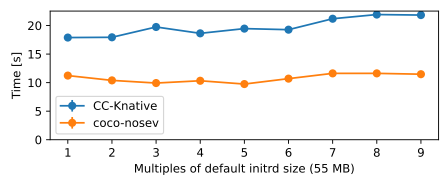


(c) CC-Knative (top) vs Kata w/ OVMF (bottom)

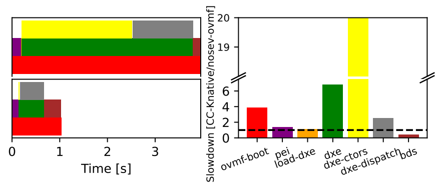
Figure 5: Comparison of the flame graph to start a cVM between CC-Knative (top row) and different baselines (bottom row).



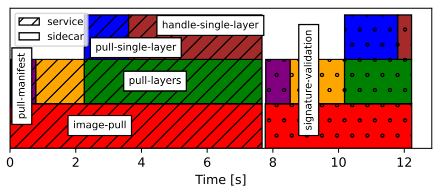
(a) Cold starts as we increase the guest memory.



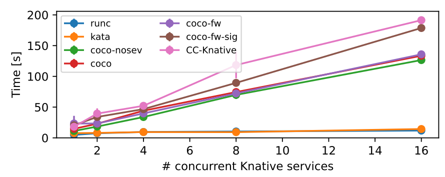
(b) Cold starts as we increase the `initrd` size.



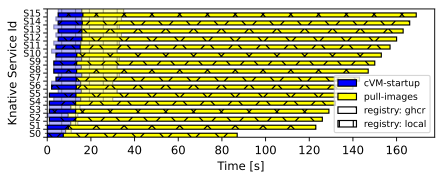
(c) CC-Knative (top) vs Kata w/ OVMF (bottom)



(a) Flame graph of the image-pulling process.



(b) Throughput-latency of service instantiation.



(c) Time-series of starting 16 Knative services.

clarification, shortly after means shortly after the service has been scaled-to-zero. Similarly to any other Knative service, CC-Knative pods are left to linger for a configurable amount of time after they are done executing, so sending two successive requests for the same service would trigger one cold-start, not two [88].

The work involved in executing a Knative service can be divided, as we studied in the previous sections, between provisioning a cVM and pulling encrypted container images. Right now, cVMs are cryptographically-bound to a guest owner, and can not be pre-warmed. This is the case for cVMs that use pre-attestation (AMD SEV, SEV-ES). Other cVMs (Intel TDX and AMD SEV-SNP) can be more easily pre-warmed as they use guest-initiated attestation protocols [62]. Even if pre-warming were to be implemented, the eager upfront guest-memory provisioning (and memory page pinning [4, 61]) for SEV(-ES) would severely limit the total amount of pre-warmed cVMs. With SEV-SNP [101] it is possible to hot-plug memory pages to cVMs, but with the added performance overheads of the `RMPUPDATE,PVALIDATE` mechanism [92], so the performance benefits of hot-plugging are still to be measured. In terms of image pulling, CC-Knative can not rely on the untrusted host to mount container images, even if they are encrypted (§5). The CoCo community is exploring secure host-side pulling mechanisms [75].

4.4 Scale-Out Latency

We now analyze the last column in Tab. 1, the scale-out latency. Fig. 7b reports the start-up latency as we increase the number of concurrent instances, simulating the situation of scaling from 0 to 16. We observe that, for all guest-side pulling baselines (even if they do not use SEV, like `coco-nosev`), scale-up latency increases linearly with the number of concurrent instances. To shed light on what is going on, we take the top-most, right-most point in Fig. 7b (CC-Knative, with 16 concurrent instances) and plot the time series of starting each different instance in Fig. 7c. We observe that the time to pull the images, even for the first service, is inflated by several orders of magnitude. This is because all 16 pods are attempting to fetch the same image from the same registry, what is causing the container registry (GHCR [82] in this case) to severely throttle our requests. This is an inherent scalability limitation of the guest-side pulling approach, and can be verified by running the same baseline with a local registry (Fig. 7c, faded bars with horizontal lines). Using the default registry image [41] in `localhost` (without disabling throttling, but having a much more permitting default than the free tier of a commercial registry), already reduces the start-up latency by at least an order of magnitude, confirming our hypothesis.

5 OPEN CHALLENGES

CC-Knative allows the execution of attested and encrypted container images in cVMs transparently to users and Knative. It is, as a consequence, a perfect candidate to provide multi-tenant confidential serverless in the cloud. However, the performance overheads described in §4 are a major hindrance to widespread adoption. In this section we outline the key research challenges to overcome.

First, to reduce cold-start times (§4.2) we need to overcome two challenges. *C1: How can we reduce the time to provision a cVM? C2: How can we reduce the time to provision an encrypted container image?* To tackle C1, we plan on exploring existing techniques to remove sandbox creation from the hot path [19, 90]. However, given the nature of (pre-)attesation protocols (§3.2), this may be challenging with SEV(-ES) cVMs. Newer cVMs also face challenges to guarantee freshness and efficient memory hot-plugging. To tackle C2, we plan on exploring lazy image loading techniques [18, 75].

Second, to reduce warm-start times (§4.3), we want to be able to re-use the effort associated to serving a cold-start. For cVM provisioning, this involves exploring cVM re-use [6, 90]. *C3: How can we re-use a cVM without affecting its confidentiality and integrity guarantees?* For encrypted images, we face a problem of encrypted data movement and sharing. *C4: How can we allow sharing and re-use of encrypted container images without affecting confidentiality?*

Lastly, improvements in scale-up latency (§4.4) will come as a combination of engineering efforts together with improvements in cold and warm starts.

6 CONCLUSIONS

In this work we have presented our experience building CC-Knative, a port of the Knative serverless runtime that executes attested, encrypted, unmodified container images inside cVMs. CC-Knative's security guarantees add 10 and 14 seconds to cold and warm starts respectively, and three minutes to scaling up from 0 to 16 functions. We extensively analyze these performance overheads, and hint towards solutions that we plan on exploring in future work.

ACKNOWLEDGMENTS

This work has been partially funded by the European Union through the Horizon Europe CLOUDSTARS project (101086248).

REFERENCES

[1] 2021. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC.

[2] Mania Abdi, Samuel Ginzburg, Xiayue Charles Lin, Jose Faleiro, Gohar Irfan Chaudhry, Inigo Goiri, Ricardo Bianchini, Daniel S Berger, and Rodrigo Fonseca. 2023. Palette Load Balancing: Locality Hints for Serverless Functions. In *Proceedings of the Eighteenth European Conference on Computer Systems* (Rome, Italy) *(EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 365–380. https://doi.org/10.1145/3552326.3567496

[3] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 419–434. https://www.usenix.org/conference/nsdi20/presentation/agache

[4] Ayaz Akram, Anna Giannakou, Venkatesh Akella, Jason Lowe-Power, and Sean Peisert. 2021. Performance Analysis of Scientific Computing Workloads on General Purpose TEEs. 1066–1076. https://doi.org/10.1109/IPDPS49936.2021.00115

[5] Fritz Alder, N Asokan, Arseny Kurnikov, Andrew Paverd, and Michael Steiner. 2019. S-faas: Trustworthy and accountable function-as-a-service using intel SGX. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*.

[6] Mohamed Alzayat, Jonathan Mace, Peter Druschel, and Deepak Garg. 2023. Groundhog: Efficient Request Isolation in FaaS. In *Proceedings of the Eighteenth European Conference on Computer Systems* (Rome, Italy) *(EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 398–415. https://doi.org/10.1145/3552326.3567503

[7] AMD. 2022. AMD Secure Encrypted Virtualization. https://developer.amd.com/sev/.

[8] AMD. 2023. Confidential Computing Performance - Google Cloud C2D VM Instances. https://www.amd.com/system/files/documents/3rd-gen-epyc-gcp-c2d-conf-compute-perf-brief.pdf.

[9] AMD. 2023. Microsoft Azure Confidential Computing Powered by 3rd Gen EPYC CPUs. https://community.amd.com/t5/epyc-processors/microsoft-azure-confidential-computing-powered-by-3rd-gen-epyc/ba-p/497796.

[10] AntStack. 2024. Serverless For Unstructured Data Problems in Life Sciences. https://www.antstack.com/blog/how-serverless-is-solving-unstructured-data-problem-for-life-sciences/.

[11] archlinux Wiki. 2024. init. https://wiki.archlinux.org/title/init.

[12] Arm. 2022. Arm TrustZone. https://www.arm.com/technologies/trustzone-for-cortex-a.

[13] Aws. 2024. Real-time fraud detection using AWS serverless and machine learning services. https://aws.amazon.com/blogs/machine-learning/real-time-fraud-detection-using-aws-serverless-and-machine-learning-services/.

[14] Microsoft Azure. 2024. Confidential Containers on Azure Container Instances. https://learn.microsoft.com/en-us/azure/container-instances/container-instances-confidential-overview.

[15] Maurice Bailleu, Dimitra Giantsidi, Vasilis Gavrielatos, Do Le Quoc, Vijay Nagarajan, and Pramod Bhatotia. 2021. Avocado: A Secure In-Memory Distributed Storage System. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 65–79. https://www.usenix.org/conference/atc21/presentation/bailleu

[16] James Bottomley. 2024. QEMU Mailing List - sev: enable secret injection to a self described area in OVMF. https://lore.kernel.org/qemu-devel/20201214154429.11023-1-jejb@linux.ibm.com/.

[17] Stefan Brenner and Rüdiger Kapitza. 2019. Trust more, serverless. In *Proceedings of the 12th ACM International Conference on Systems and Storage*.

[18] Marc Brooker, Mike Danilov, Chris Greenwood, and Phil Piwonka. 2023. On-demand Container Loading in AWS Lambda. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 315–328. https://www.usenix.org/conference/atc23/presentation/brooker

[19] James Cadden, Thomas Unger, Yara Awad, Han Dong, Orran Krieger, and Jonathan Appavoo. 2020. SEUSS: Skip Redundant Paths to Make Serverless Fast. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 32, 15 pages. https://doi.org/10.1145/3342195.3392698

[20] Google Cloud. 2022. Confidential Computing. https://cloud.google.com/confidential-computing.

[21] Google Cloud. 2022. Ubiquitous Data Encryption. https://cloud.google.com/compute/confidential-vm/docs/ubiquitous-data-encryption.

[22] Google Cloud. 2024. What is a Virtual Machine? https://cloud.google.com/learn/what-is-a-virtual-machine.

[23] Confidential Computing Consortium. 2022. Confidential Computing - Open Source Community. https://confidentialcomputing.io/.

[24] containerd. 2024. An industry-standard container runtime with an emphasis on simplicity, robustness and portability. https://containerd.io/.

[25] containerd. 2024. Runtime v2. https://github.com/containerd/containerd/tree/main/runtime/v2.

[26] Containers. 2024. OCIcrypt - Encryption libraries for OCI container images. https://github.com/containers/ocicrypt.

[27] Containers. 2024. Skopeo - Work with remote image registries. https://github.com/containers/skopeo.

[28] Confidential Containers. 2024. Attestation Agent. https://github.com/confidential-containers/guest-components/tree/main/attestation-agent.

[29] Confidential Containers. 2024. Confidential Containers - Overview. https://github.com/confidential-containers/confidential-containers/blob/main/overview.md.

[30] Confidential Containers. 2024. Generic Key Broker Service. https://github.com/confidential-containers/kbs.

[31] Confidential Containers. 2024. image-rs - Container Images Rust Crate. https://github.com/confidential-containers/guest-components/tree/main/image-rs.

[32] Confidential Containers. 2024. Key Broker Client. https://github.com/confidential-containers/guest-components/tree/main/attestation-agent/kbc.

[33] Confidential Containers. 2024. Welcome to Confidential Containers! https://confidentialcontainers.org/.

[34] Kata Containers. 2023. The speed of containers, the security of VMs. https://katacontainers.io/.
[35] Kata Containers. 2024. Kata Agent. https://github.com/kata-containers/kata-containers/blob/main/src/agent/README.md.
[36] Kata Containers. 2024. Kata Agent API - Github. https://github.com/kata-containers/kata-containers/blob/CCv0/src/runtime/virtcontainers/kata_agent.go_L2518-L2531.
[37] Kata Containers. 2024. Kata Containers Architecture. https://github.com/kata-containers/kata-containers/tree/main/docs/design/architecture.
[38] Kata Containers. 2024. Kata Open Policy Agent. https://github.com/kata-containers/kata-containers/tree/main/src/kata-opa.
[39] Open Containers. 2023. runc - CLI tool for spawning and running containers according to the OCI specification. https://github.com/opencontainers/runc.
[40] Open Containers. 2024. OCI Image Format Specification. https://github.com/opencontainers/image-spec.
[41] DockerHub. 2024. registry - Distribution implementation for storing and distributing container images and artifacts. https://hub.docker.com/_/registry.
[42] Knative Serving Docs. 2023. Hello World - Python. https://github.com/knative/docs/tree/main/code-samples/serving/hello-world/helloworld-python.
[43] enclave cc. 2024. Process-based Confidential Container Runtime. https://github.com/confidential-containers/enclave-cc.
[44] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 363–376. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi
[45] Anders Tungeland Gjerdrum, Håvard Dagenborg Johansen, Lars Brenna, and Dag Johansen. 2019. Diggi: A Secure Framework for Hosting Native Cloud Functions with Minimal Trust. In *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*. 18–27. https://doi.org/10.1109/TPS-ISA48467.2019.00012
[46] Gramine. 2024. Gramine Project - a library OS for Unmodified Applications. https://gramineproject.io/.
[47] Brendan Gregg. 2023. Flame Graphs. https://www.brendangregg.com/flamegraphs.html.
[48] Red Hat. 2024. Attestation in Confidential Computing. https://www.redhat.com/en/blog/attestation-confidential-computing.
[49] Red Hat. 2024. Confidential computing use cases. https://www.redhat.com/en/blog/confidential-computing-use-cases.
[50] Red Hat. 2024. Understanding the Confidential Containers Attestation Flow. https://www.redhat.com/en/blog/understanding-confidential-containers-attestation-flow.
[51] IBM. 2023. IBM Cloud. https://www.ibm.com/cloud.
[52] IBM. 2023. IBM Cloud Bare Metal Servers. https://www.ibm.com/products/bare-metal-servers.
[53] Apache Incubator. 2021. Teaclave. https://github.com/apache/incubator-teaclave.
[54] Intel. 2022. Intel Software Guard Extensions. https://www.intel.co.uk/content/www/uk/en/architecture-and-technology/software-guard-extensions.html.
[55] Intel. 2024. Intel TDX - CCC Linux Guest Hardening. https://intel.github.io/ccc-linux-guest-hardening-docs/security-spec.html.
[56] Intel. 2024. Intel Trust Domain Extensions. https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html.
[57] Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2018. Serving Deep Learning Models in a Serverless Platform. In *IEEE International Conference on Cloud Engineering, (IC2E)*.
[58] Zhipeng Jia and Emmett Witchel. 2021. Boki: Stateful Serverless Computing with Shared Logs. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 691–707. https://doi.org/10.1145/3477132.3483541
[59] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the Cloud: Distributed Computing for the 99%. In *ACM Symposium on Cloud Computing (SOCC)*.
[60] Artjom Joosen, Ahmed Hassan, Martin Asenov, Rajkarn Singh, Luke Darlow, Jianfeng Wang, and Adam Barker. 2023. How Does It Function? Characterizing Long-Term Trends in Production Serverless Workloads. In *Proceedings of the 2023 ACM Symposium on Cloud Computing* (, Santa Cruz, CA, USA,) *(SoCC '23)*. Association for Computing Machinery, New York, NY, USA, 443–458. https://doi.org/10.1145/3620678.3624783
[61] David Kaplan. 2016. AMD x86 Memory Encryption Technologies. USENIX Association, Austin, TX.
[62] David Kaplan. 2023. Hardware VM Isolation in the Cloud: Enabling confidential computing with AMD SEV-SNP technology. *Queue* 21, 4 (sep 2023), 49–67. https://doi.org/10.1145/3623392
[63] Knative. 2024. Knative is an Open-Source Enterprise-level solution to build Serverless and Event Driven Applications. https://knative.dev/docs/.
[64] Knative. 2024. Knative Serving Architecture. https://knative.dev/docs/serving/architecture/.
[65] Knative. 2024. Tag Resolution. https://knative.dev/docs/serving/tag-resolution/.
[66] Kubernetes. 2024. CRI - Container Runtime Interface. https://kubernetes.io/docs/concepts/architecture/cri/.
[67] Kubernetes. 2024. kubelet. https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/.
[68] Linux KVM. 2024. Kernel Virtual Machine. https://linux-kvm.org/page/Main_Page.
[69] Ashraf Mahgoub, Edgardo Barsallo Yi, Karthick Shankar, Sameh Elnikety, Somali Chaterji, and Saurabh Bagchi. 2022. ORION and the Three Rights: Sizing, Bundling, and Prewarming for Serverless DAGs. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 303–320. https://www.usenix.org/conference/osdi22/presentation/mahgoub
[70] Linux manual page. 2024. namespaces. https://man7.org/linux/man-pages/man7/namespaces.7.html.
[71] Microsoft. 2020. Microsoft Azure Attestation. https://docs.microsoft.com/azure/attestation/overview.
[72] Microsoft. 2022. Microsoft Azure Confidential Computing. https://azure.microsoft.com/en-gb/solutions/confidential-compute/.
[73] Microsoft. 2023. Inside Look: How Azure Linux powers Confidential Containers on AKS. https://techcommunity.microsoft.com/t5/linux-and-open-source-blog/inside-look-how-azure-linux-powers-confidential-containers-on/ba-p/3981296.
[74] Microsoft. 2024. Azure Functions - Execute event-driven serverless code with an end-to-end development experience. https://azure.microsoft.com/en-us/products/functions/.
[75] Nydus. 2024. Nydus - Acceleration Framework For Container Image. https://nydus.dev/.
[76] QEMU Options. 2023. RAM. https://wiki.gentoo.org/wiki/QEMU/Options_RAM.
[77] OVMF. 2024. AMD SEV x64 Package. https://github.com/tianocore/edk2/blob/master/OvmfPkg/AmdSev/AmdSevX64.dsc.
[78] The Washington Post. 2024. NSA infiltrates links to Yahoo, Google data centers worldwide, Snowden documents say. https://www.washingtonpost.com/world/national-security/nsa-infiltrates-links-to-yahoo-google-data-centers-worldwide-snowden-documents-say/2013/10/30/e51d661e-4166-11e3-8b74-d89d714ca4dd_story.html.
[79] Qemu. 2024. Qemu - A generic and open-source machine emulator and virtualizer. https://www.qemu.org/.
[80] Qemu. 2024. QEMU Firmware Configuration Device. https://www.qemu.org/docs/master/specs/fw_cfg.html.
[81] Quay. 2024. Quay Container Registry. https://quay.io/.
[82] Github Container Registry. 2024. Your packages, at home with their code. https://github.com/features/packages.
[83] IBM Research. 2024. LPC 2021 - Attestation and Secret Injection for Confidential VMs, Containers, and Pods. https://lpc.events/event/11/contributions/994/.
[84] Alireza Sahraei, Soteris Demetriou, Amirali Sobhgol, Haoran Zhang, Abhigna Nagaraja, Neeraj Pathak, Girish Joshi, Carla Souza, Bo Huang, Wyatt Cook, Andrii Golovei, Pradeep Venkat, Andrew Mcfague, Dimitrios Skarlatos, Vipul Patel, Ravinder Thind, Ernesto Gonzalez, Yun Jin, and Chunqiang Tang. 2023. XFaaS: Hyperscale and Low Cost Serverless Functions at Meta. 231–246. https://doi.org/10.1145/3600006.3613155
[85] SeaBIOS. 2023. SeaBIOS. https://www.seabios.org/SeaBIOS.
[86] Kaspersky Security. 2024. Downgrade Attack. https://encyclopedia.kaspersky.com/glossary/downgrade-attack/.
[87] Amazon Web Services. 2024. AWS Lambda - Run code without thinking of servers or clusters. https://aws.amazon.com/lambda/.
[88] Knative Serving. 2024. Configuring Scale to Zero. https://knative.dev/docs/serving/autoscaling/scale-to-zero/.
[89] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 205–218. https://www.usenix.org/conference/atc20/presentation/shahrad
[90] Simon Shillaker and Peter Pietzuch. 2020. Faasm: Lightweight Isolation for Efficient Stateful Serverless Computing. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 419–433. https://www.usenix.org/conference/atc20/presentation/shillaker
[91] Sigstore. 2024. Cosign - Container Signing. https://github.com/sigstore/cosign.
[92] Brijesh Singh. 2024. [PATCH v9 00/43] Add AMD Secure Nested Paging (SEV-SNP) Guest Support. https://lore.kernel.org/linux-mm/20220205162249.4dkttihw6my7iha3@amd.com/t/.

[93] UEFI Platform Initialization Specification. 2023. Driver Execution Environment (DXE) Phase. https://uefi.org/specs/PI/1.8/V2_Overview.html.

[94] Edgless Systems. 2024. The world's most secure Kubernetes. https://www.edgeless.systems/products/constellation/.

[95] Tianocore. 2024. OVMF - Open Virtual Machine Firmware. https://github.com/tianocore/tianocore.github.io/wiki/OVMF.

[96] Bohdan Trach, Oleksii Oleksenko, Franz Gregor, Pramod Bhatotia, and Christof Fetzer. 2019. Clemmys: Towards secure remote execution in FaaS. In *Proceedings of the 12th ACM International Conference on Systems and Storage*.

[97] VirTEE. 2024. Calculate AMD SEV/SEV-ES/SEV-SNP measurement for confidential computing. https://github.com/virtee/sev-snp-measure.

[98] VMWare. 2024. Introduction to vSockets. https://vdc-repo.vmware.com/vmwb-repository/dcr-public/a49be05e-fa6d-4da1-9186-922fbfef149e/a65f3c51-aaeb-476d-80c3-827b805c2f9e/doc/vsockAbout.3.2.html.

[99] Jinpeng Wei and Calton Pu. 2005. TOCTTOU Vulnerabilities in UNIX-Style File Systems: An Anatomical Study. In *4th USENIX Conference on File and Storage Technologies (FAST 05)*. USENIX Association, San Francisco, CA. https://www.usenix.org/conference/fast-05/tocttou-vulnerabilities-unix-style-file-systems-anatomical-study

[100] Xingda Wei, Fangming Lu, Tianxia Wang, Jinyu Gu, Yuhan Yang, Rong Chen, and Haibo Chen. 2023. No Provisioned Concurrency: Fast RDMA-codesigned Remote Fork for Serverless Computing. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 497–517. https://www.usenix.org/conference/osdi23/presentation/wei-rdma

[101] AMD Whitepaper. 2024. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf.

[102] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 283–298. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zheng