Imperial College London

LSDS
Large-Scale Data & Systems Group

Serverless Confidential Containers: Challenges and Opportunities

Carlos Segarra

(w/ Tobin Feldman-Fitzthum and Daniele Buono)

Large-Scale Data & Systems (LSDS) Group - Imperial College London

Visiting IBM TJ Watson (Sep'23 – Nov'23)

https://carlossegarra.com
<cs1620@ic.ac.uk>

CLOUD STARS

9th International Workshop on Serverless Computing (WOSC)

Agenda

1. Introduction to Confidential Serverless

- Characterising serverless functions: Cold/Warm starts and burstiness
- Problems with existing serverless offerings

2. Background:

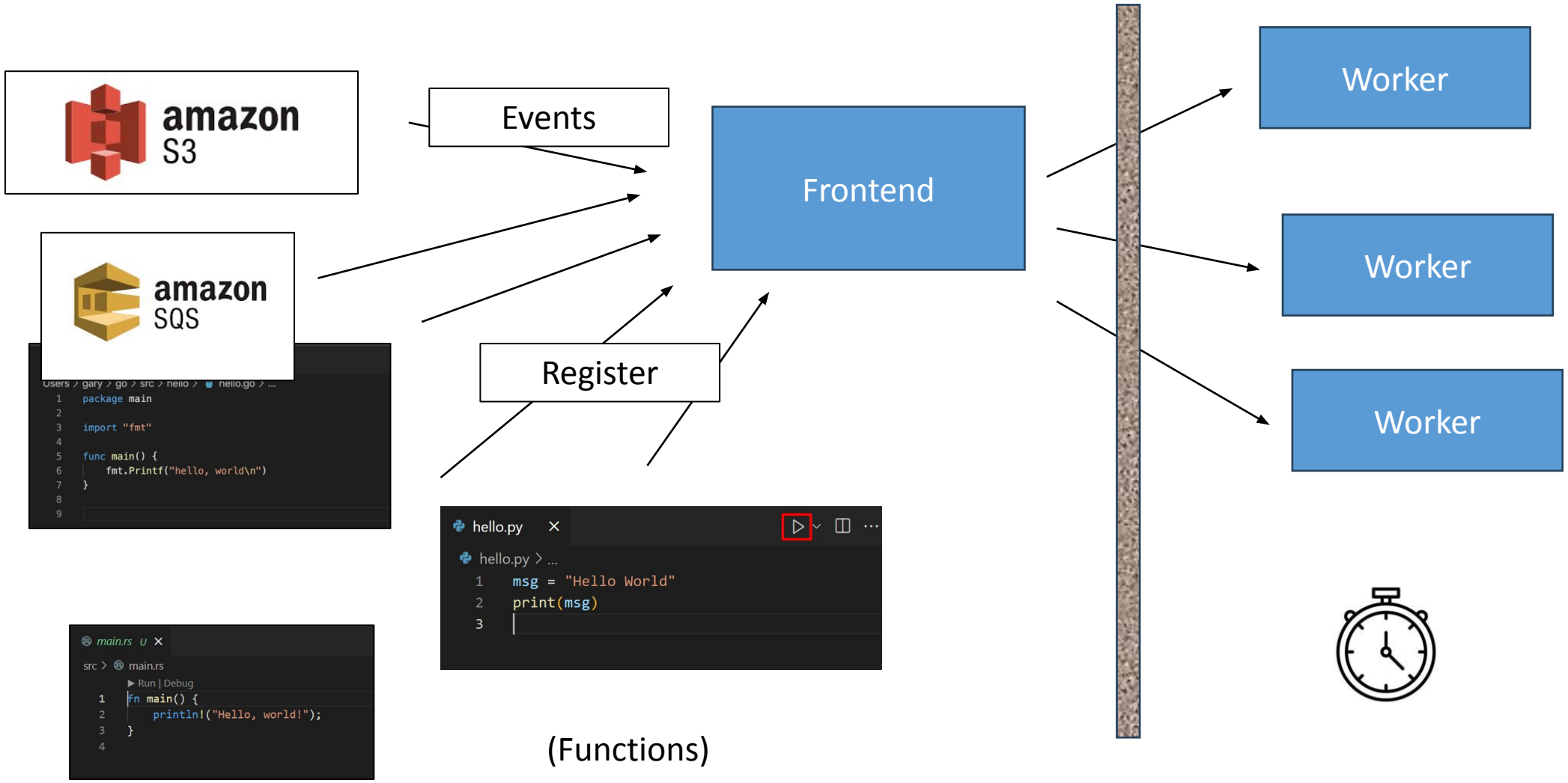
- Design space for confidential serverless
- Kata and Confidential Containers

3. PoC: Knative on Confidential Containers

4. Evaluation

- Cold-Starts
- Warm-Starts
- Instantiation Throughput

Introduction: Serverless Functions



Introduction: Characterizing Serverless Functions

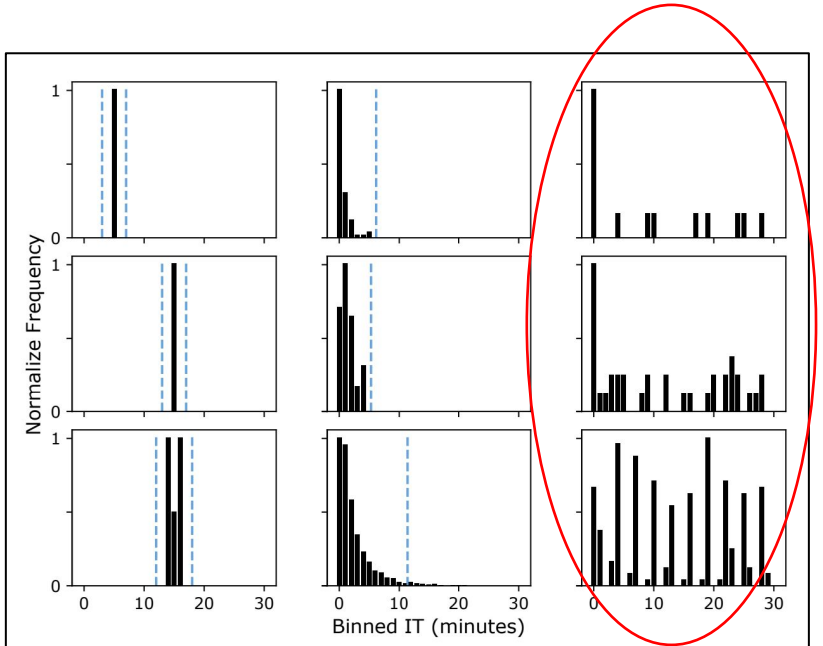


Figure 12: Nine normalized IT distributions from real FaaS workloads over a week.

Functions are bursty!

Functions are short-lived!
(90% shorter than 10s)

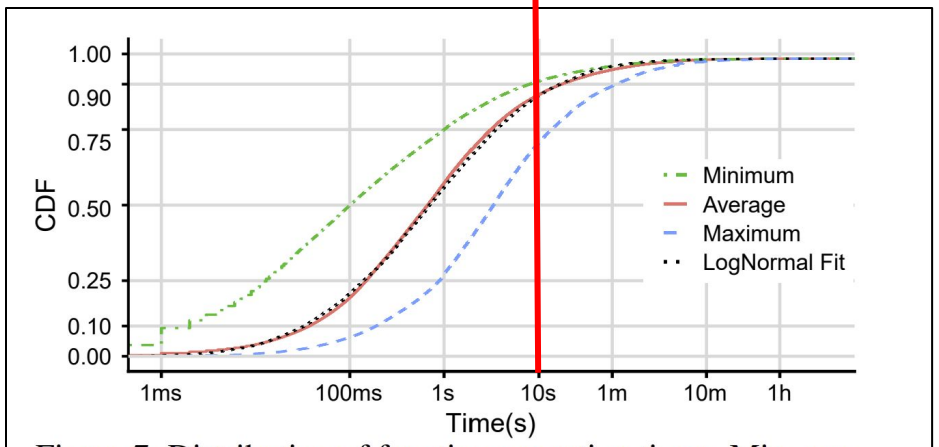
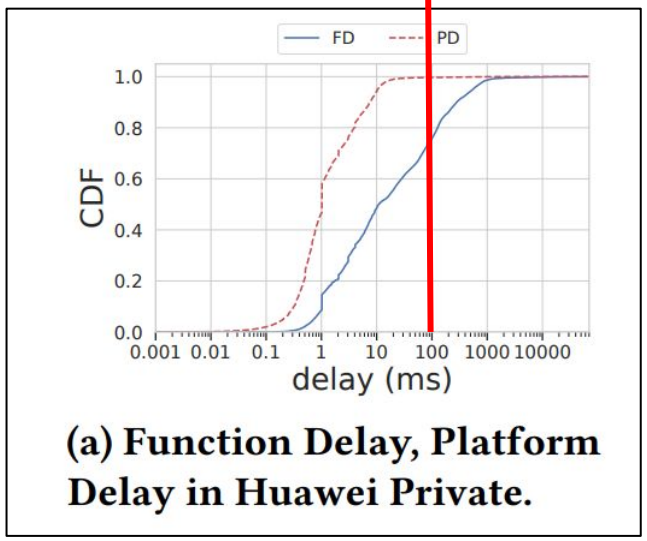


Figure 7: Distribution of function execution times. Min, avg, and max are separate CDFs, and use independent sorting.

[ATC'20] Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider

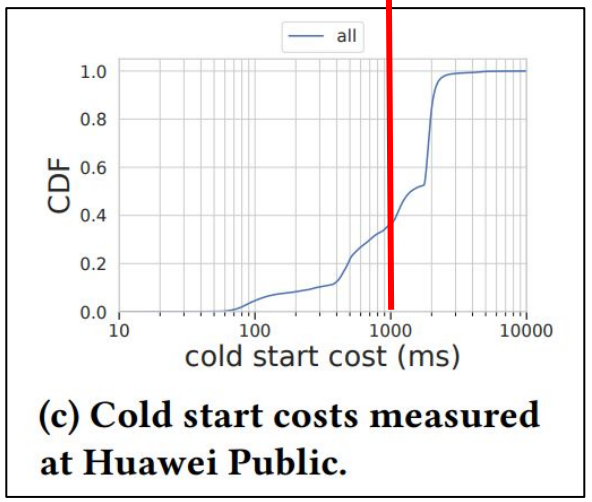
Introduction: Characterizing Serverless Functions

Yet 80% of functions execute in < 100 ms !



(a) Function Delay, Platform Delay in Huawei Private.

Only 40% of functions take less than 1s to start



(c) Cold start costs measured at Huawei Public.

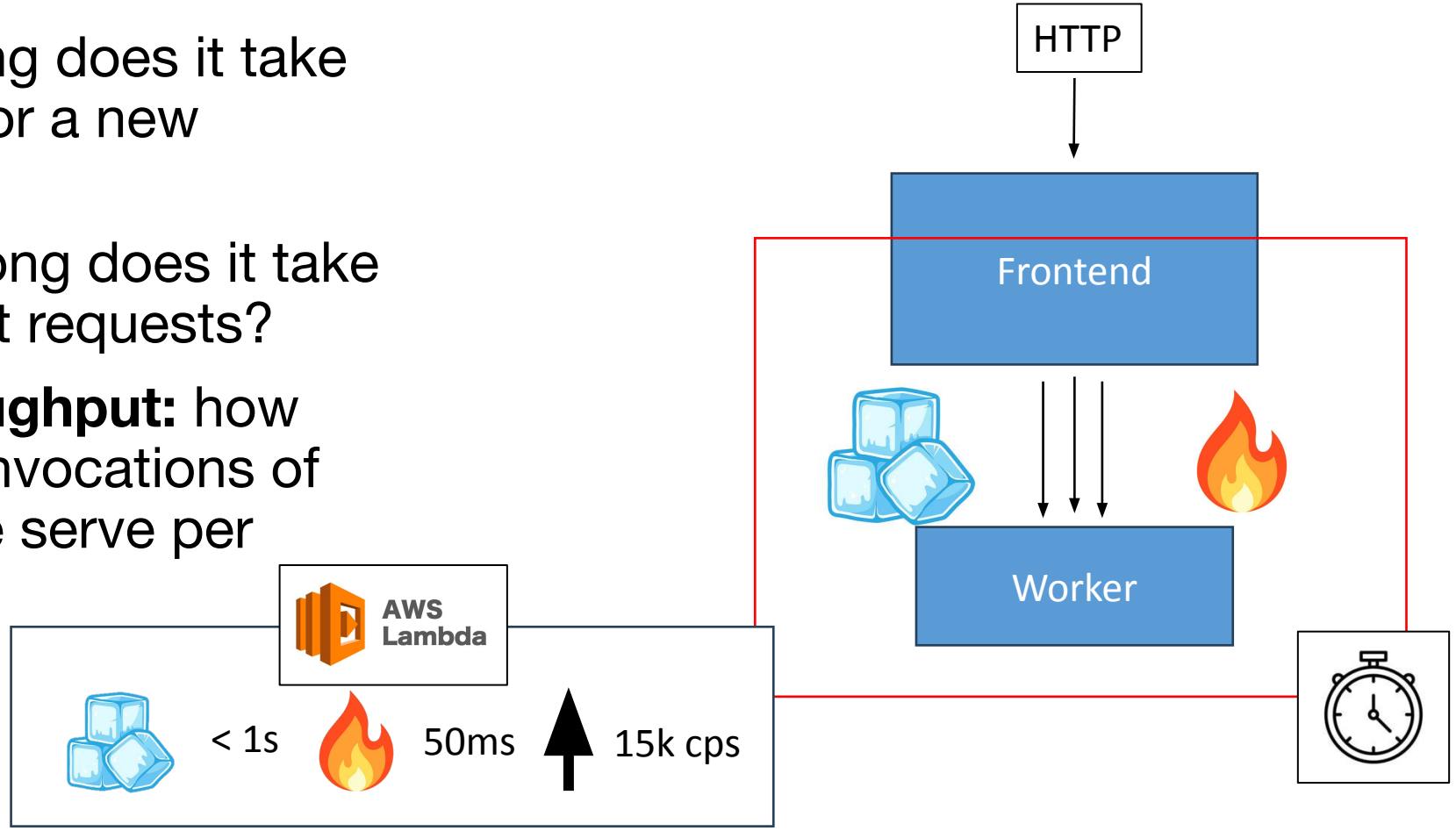
[SoCC'23] How Does It Function? Characterizing Long-term Trends in Production Serverless Workloads

Introduction: Problems in Serverless

Cold-Start: how long does it take to serve a request for a new function?

Warm-Start: how long does it take to serve subsequent requests?

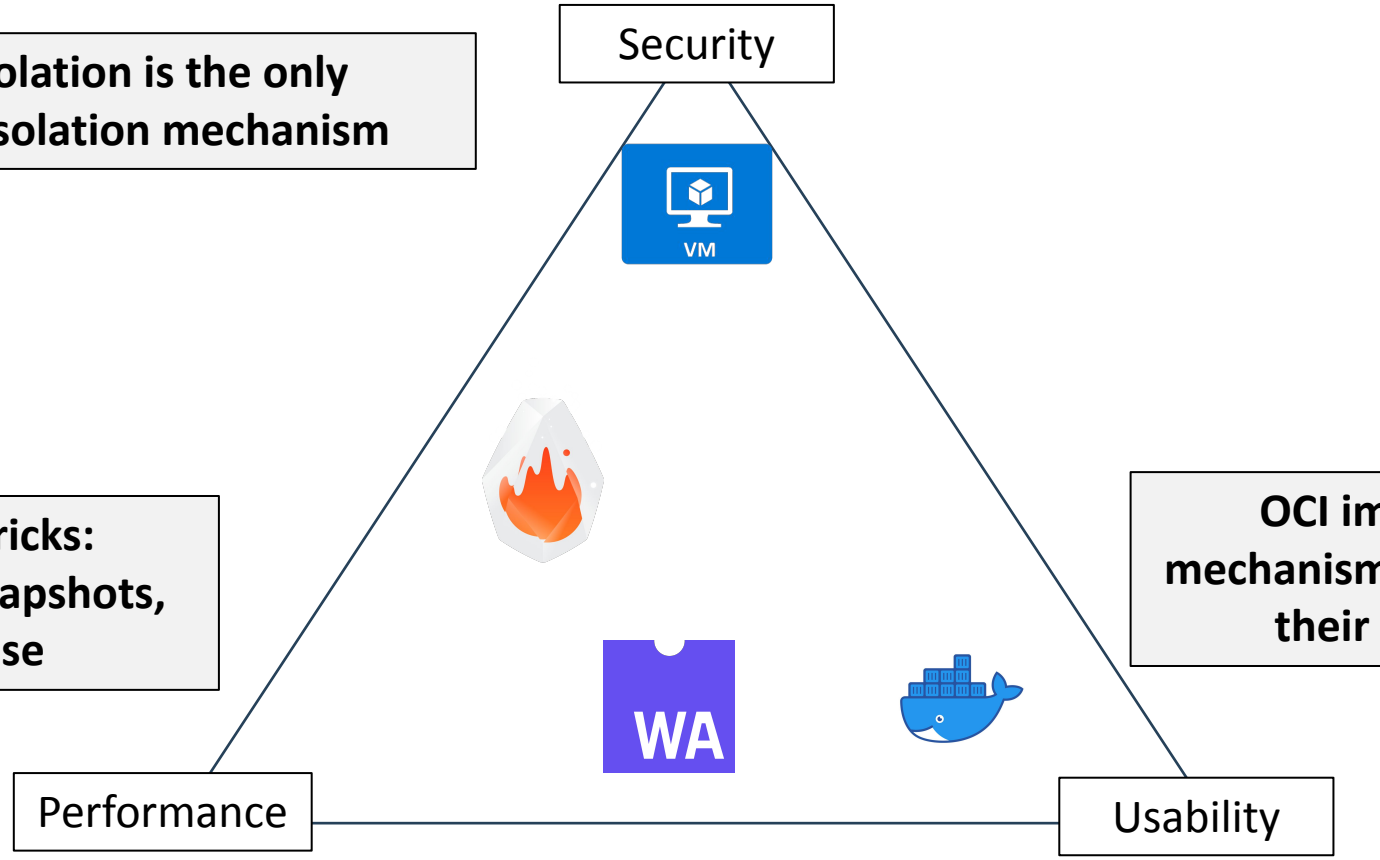
Instantiation Throughput: how many (concurrent) invocations of this function can we serve per second?



[ATC'23] On-demand Container Loading in AWS Lambda

Introduction: Inter-Function Isolation in Serverless

VM-like Isolation is the only acceptable isolation mechanism

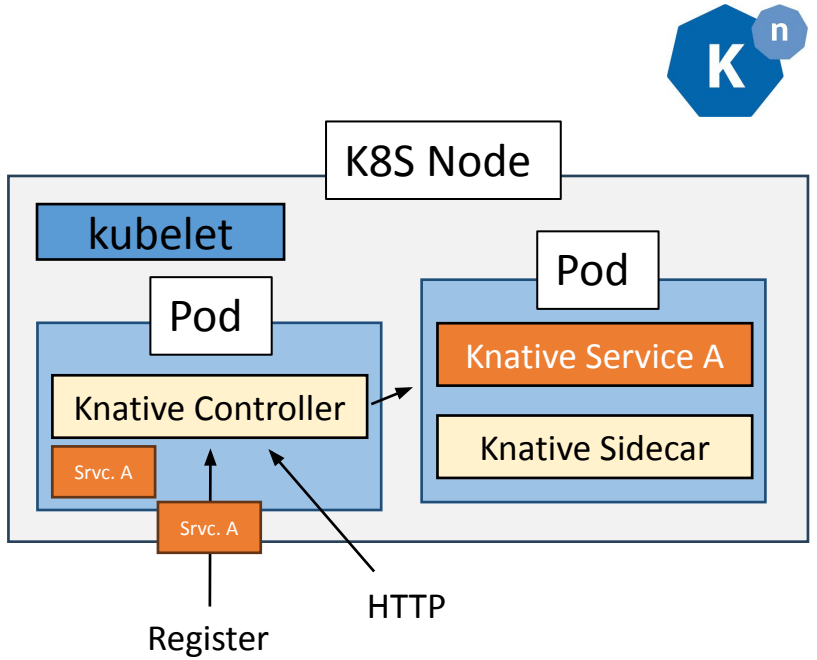


Performance Tricks: micro-VMs, VM snapshots, and VM re-use

OCI images are the de-facto mechanism to express functions (and their dependency closure)

Introduction: More Problems in Serverless!

Inter-function isolation is fine, but not enough!

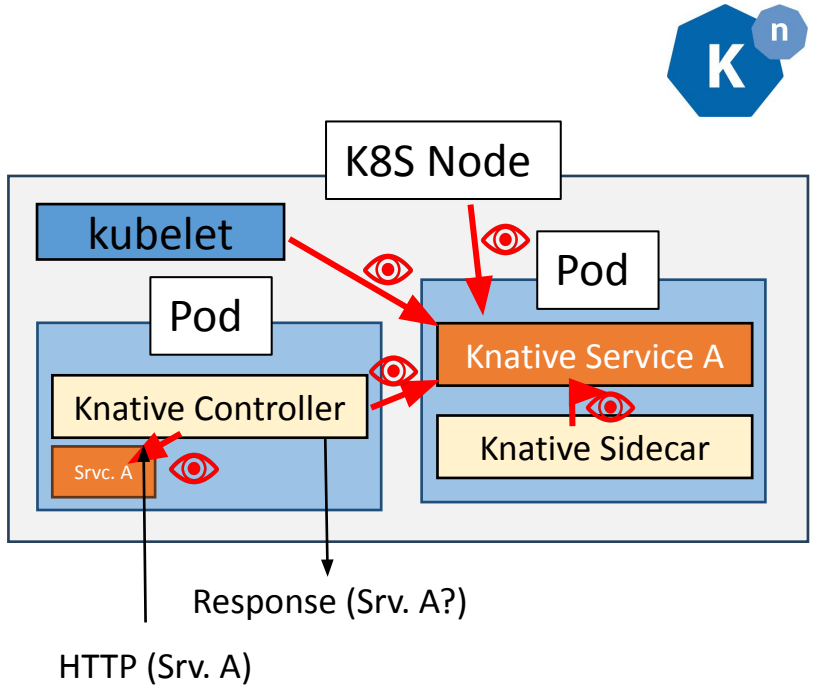


Introduction: More Problems in Serverless!

Inter-function isolation is fine, but not enough!

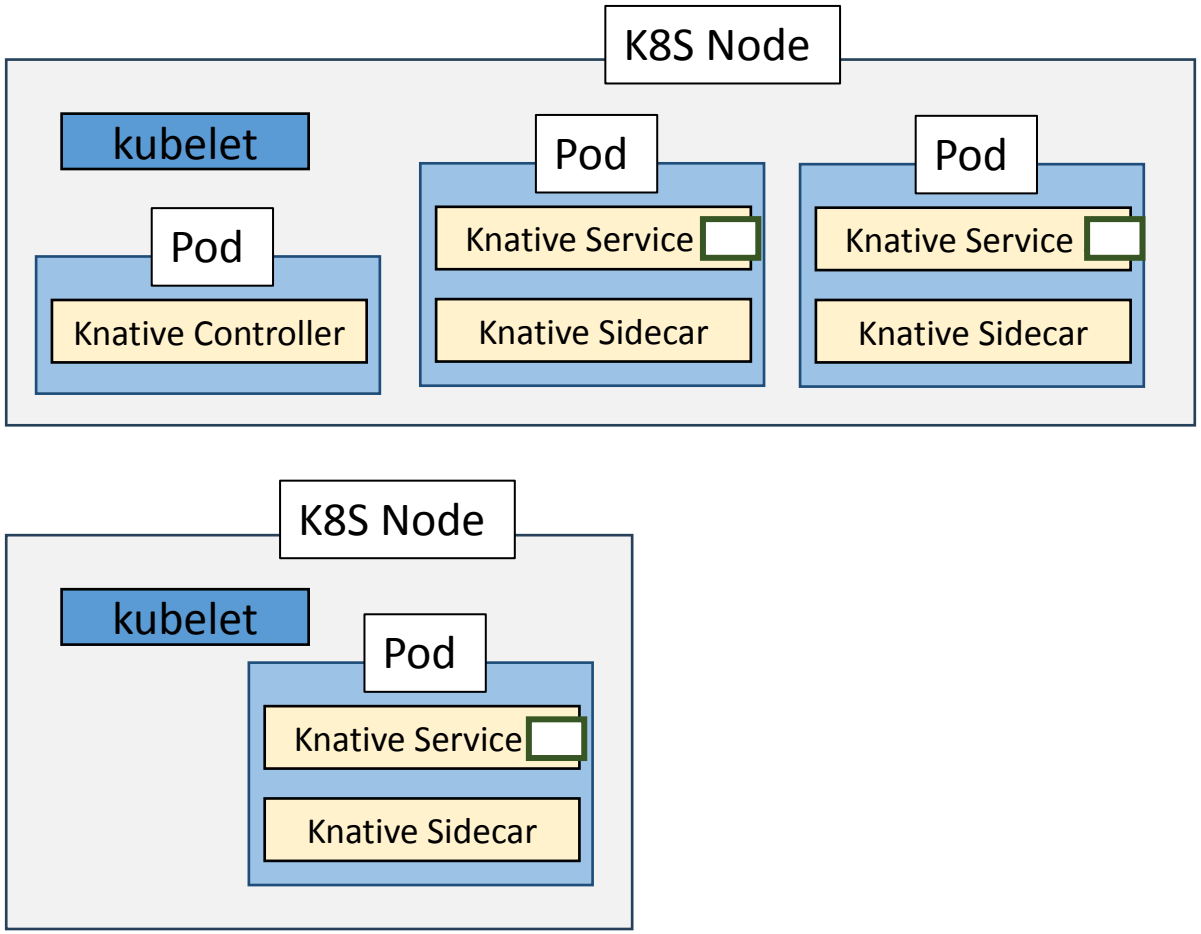
We need isolation from the host environment to guarantee...

- Data Confidentiality
- Code Confidentiality
- Execution Integrity



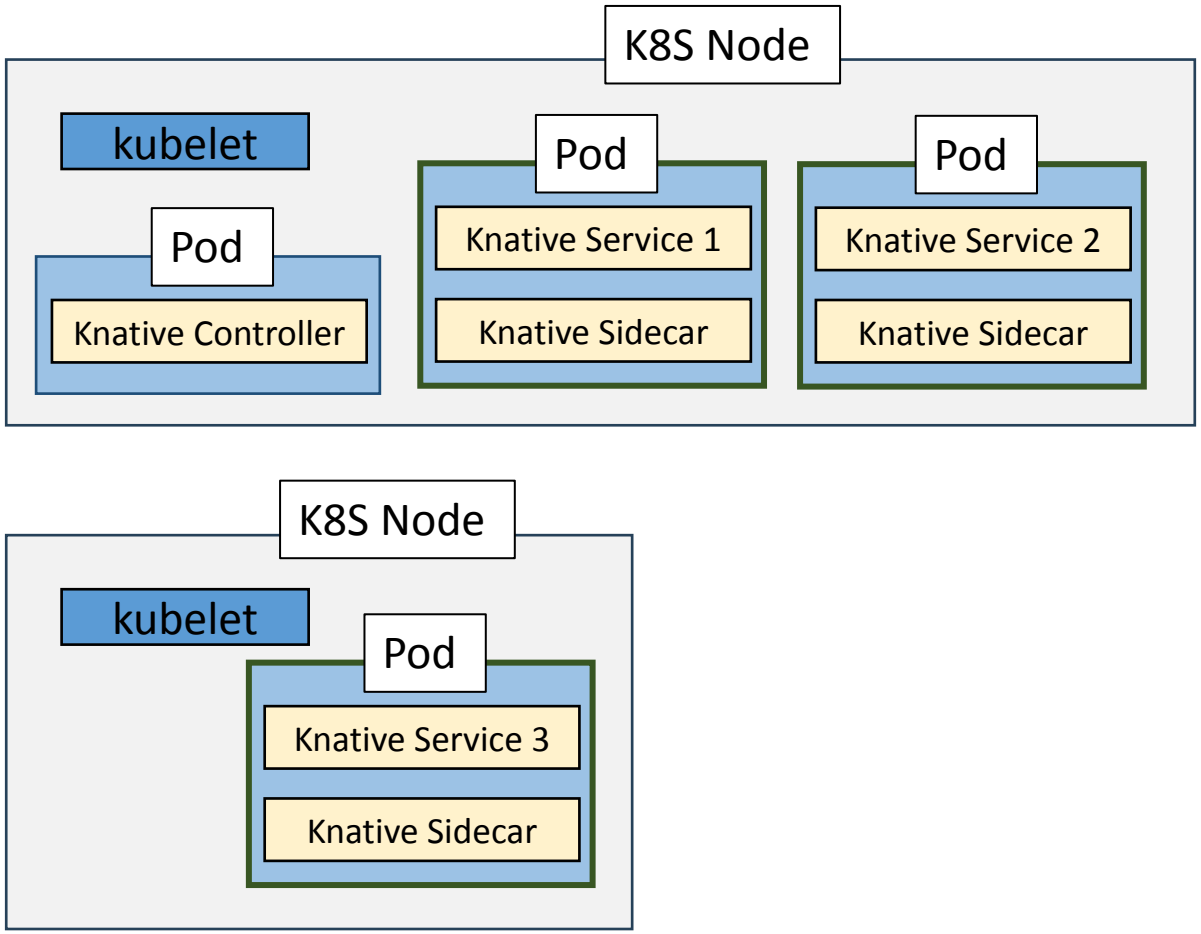
Confidential Computing

Background: Design Space for Confidential Serverless



Process-Centric Security

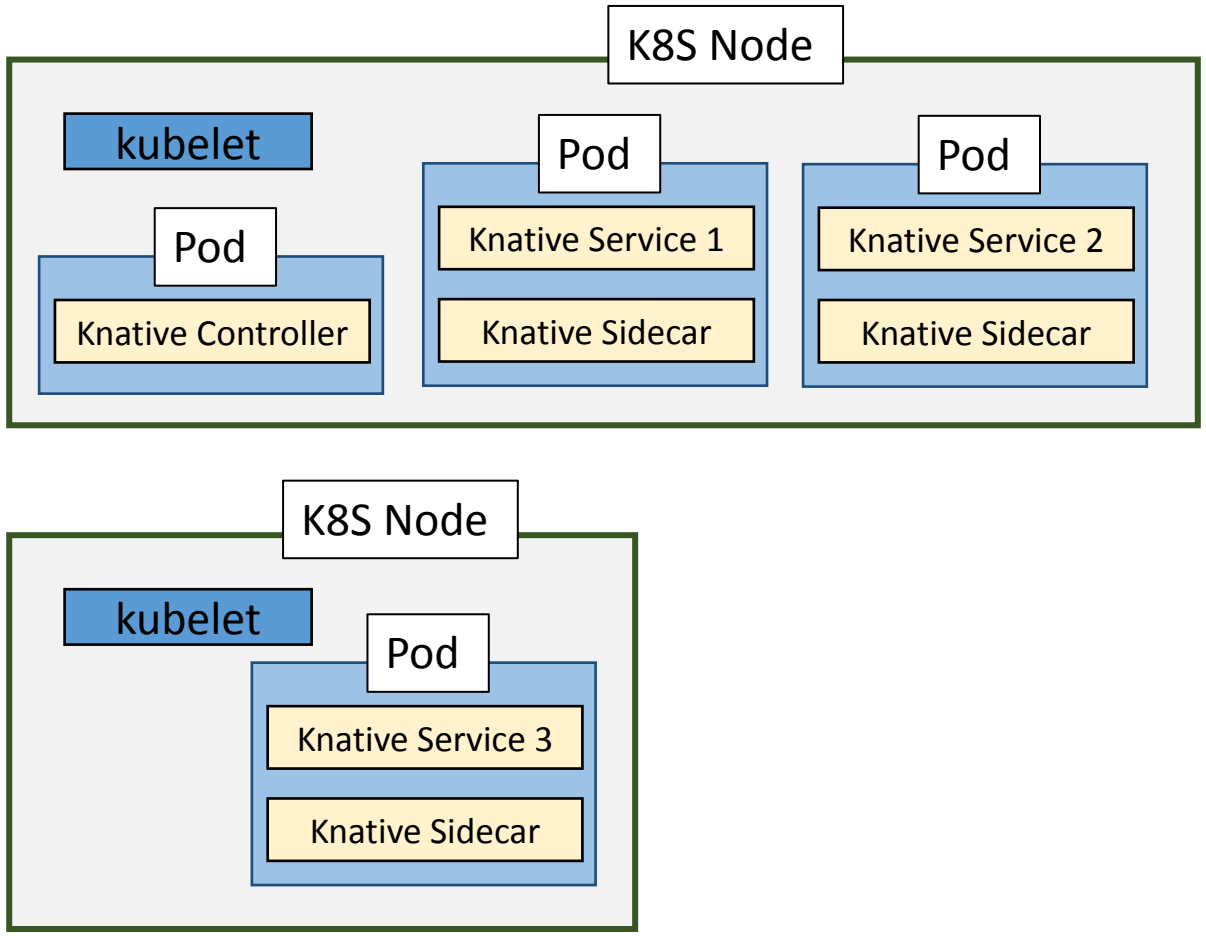
Background: Design Space for Confidential Serverless



Process-Centric Security

Pod-Centric Security

Background: Design Space for Confidential Serverless

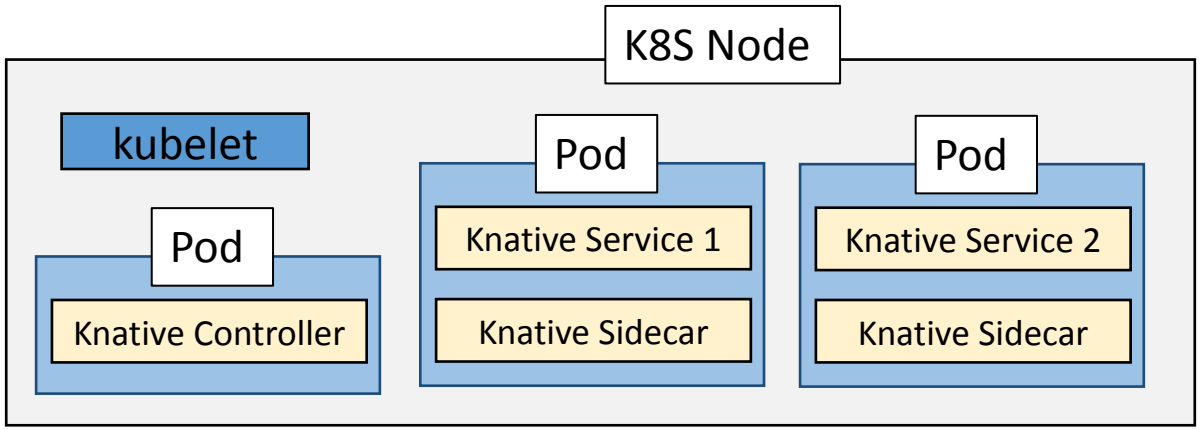


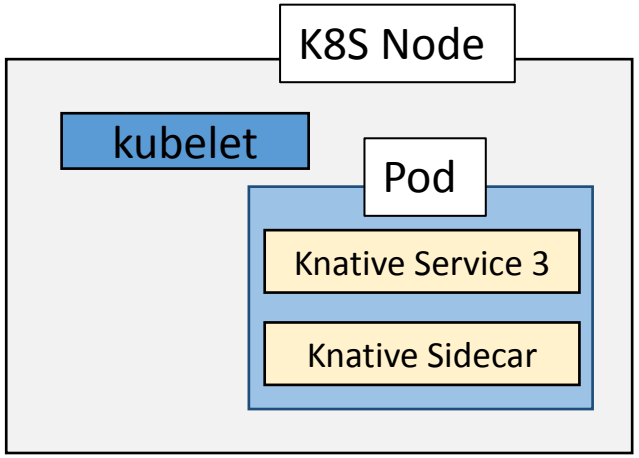
Process-Centric Security

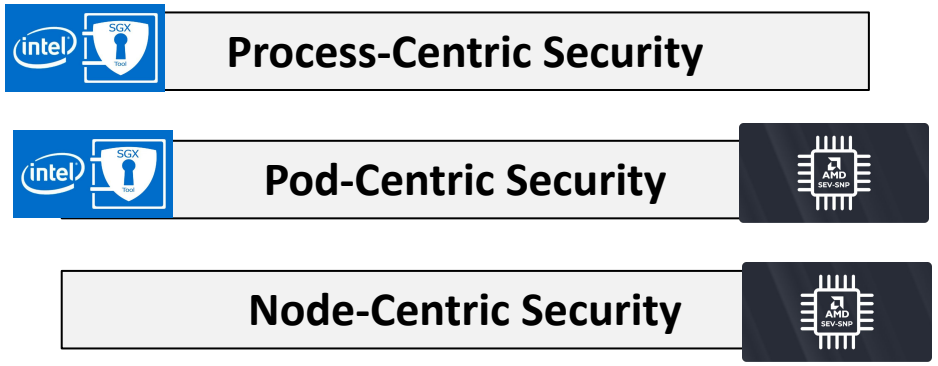
Pod-Centric Security

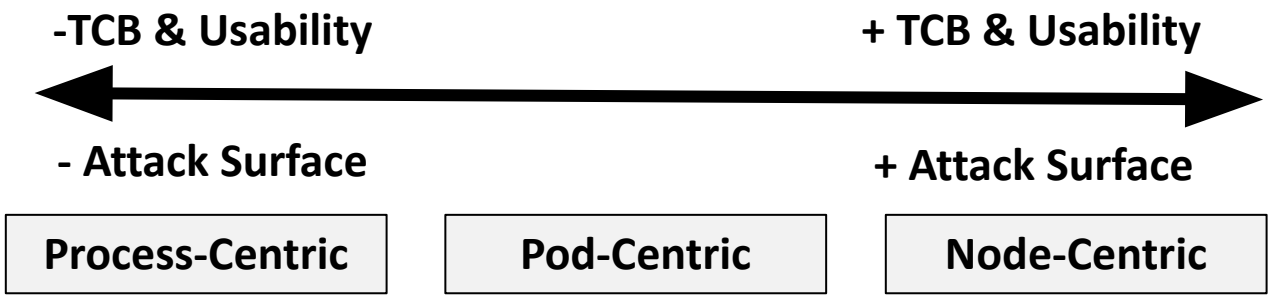
Node-Centric Security

Background: Design Space for Confidential Serverless

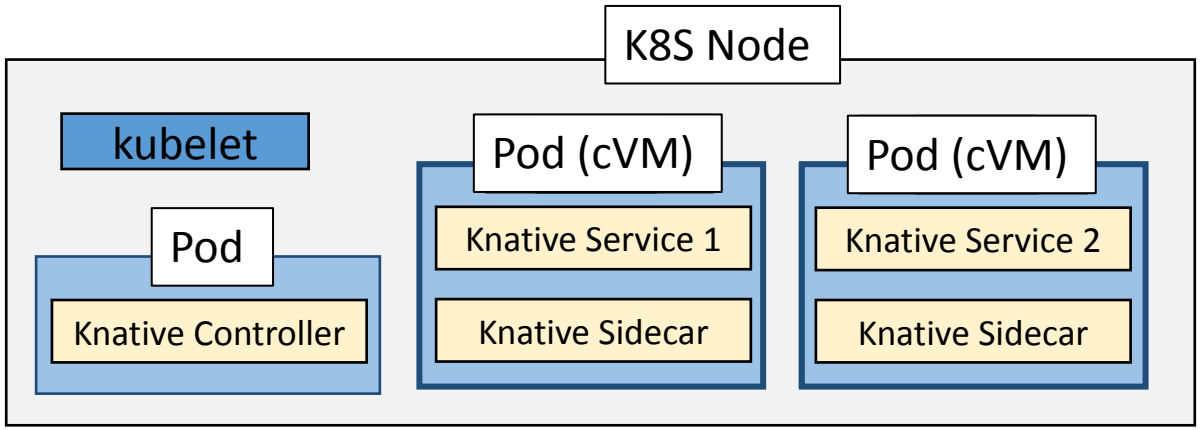


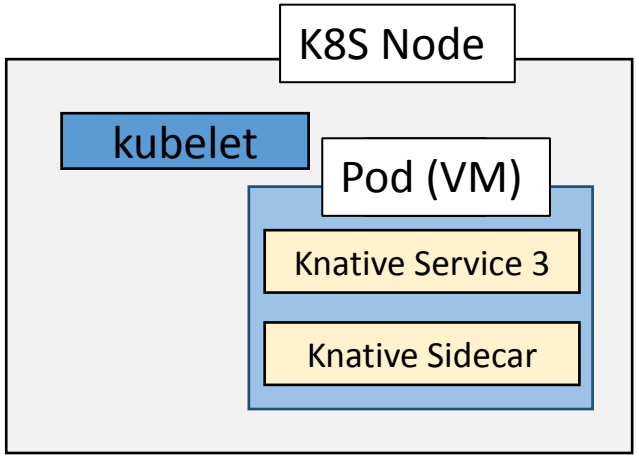






Background: Design Space for Confidential Serverless





~~Process-Centric Security~~

Pod-Centric Security

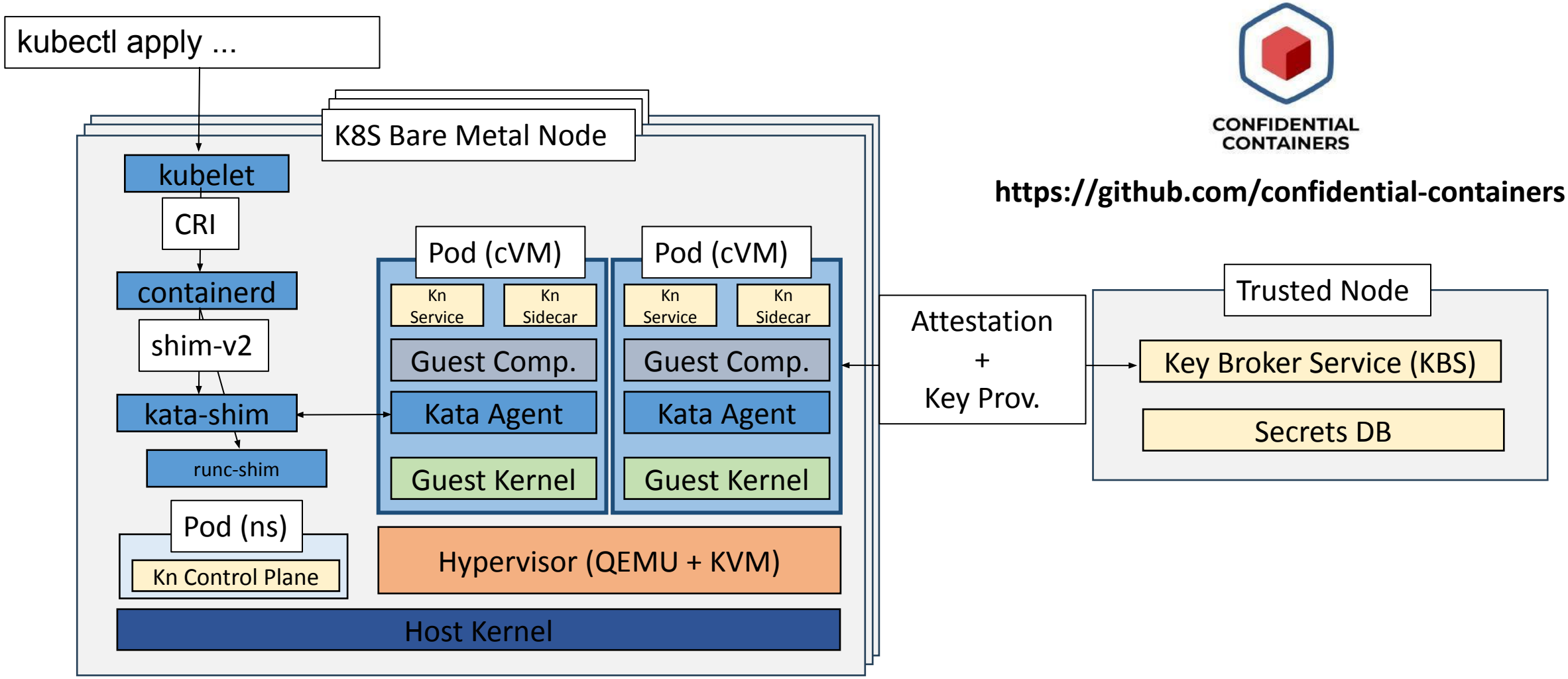
~~Node-Centric Security~~

kata containers

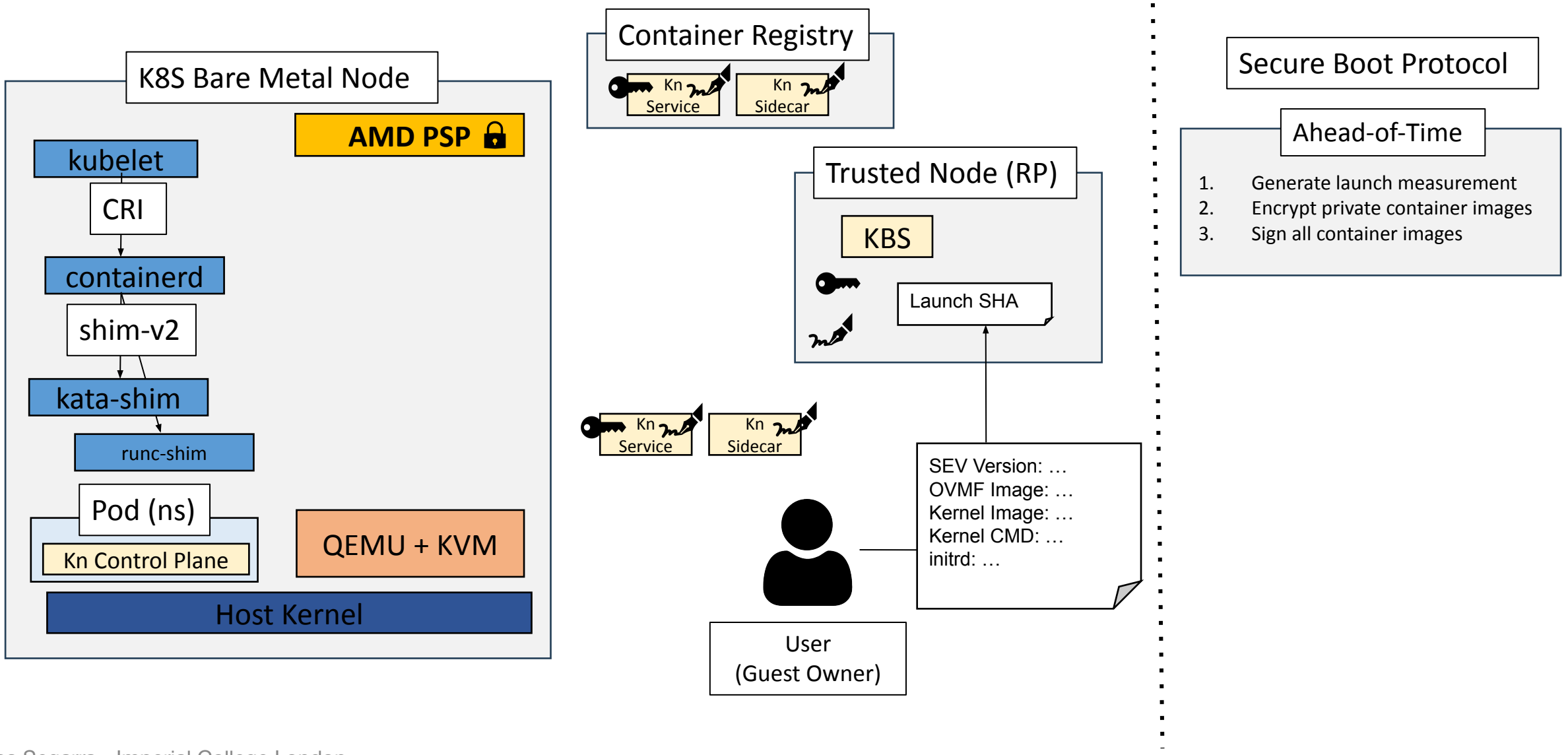
CONFIDENTIAL CONTAINERS

https://github.com/confidential-containers

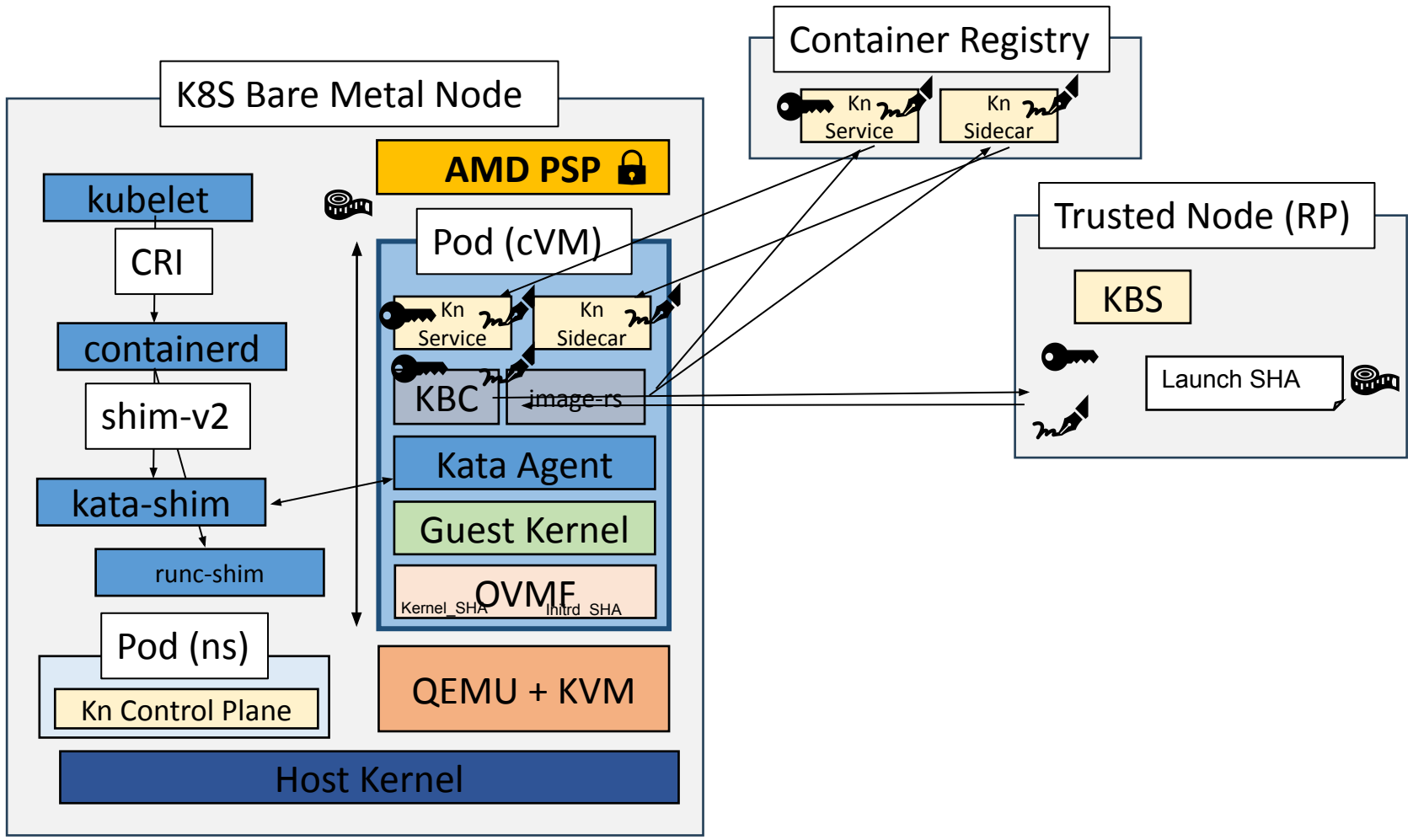
PoC: Knative on Confidential Containers



PoC: Attestation of Knative on CoCo (AMD SEV)



PoC: Attestation of Knative on CoCo (AMD SEV)



Secure Boot Protocol

Ahead-of-Time

1. Generate launch measurement
2. Encrypt private container images
3. Sign all container images

Run-time

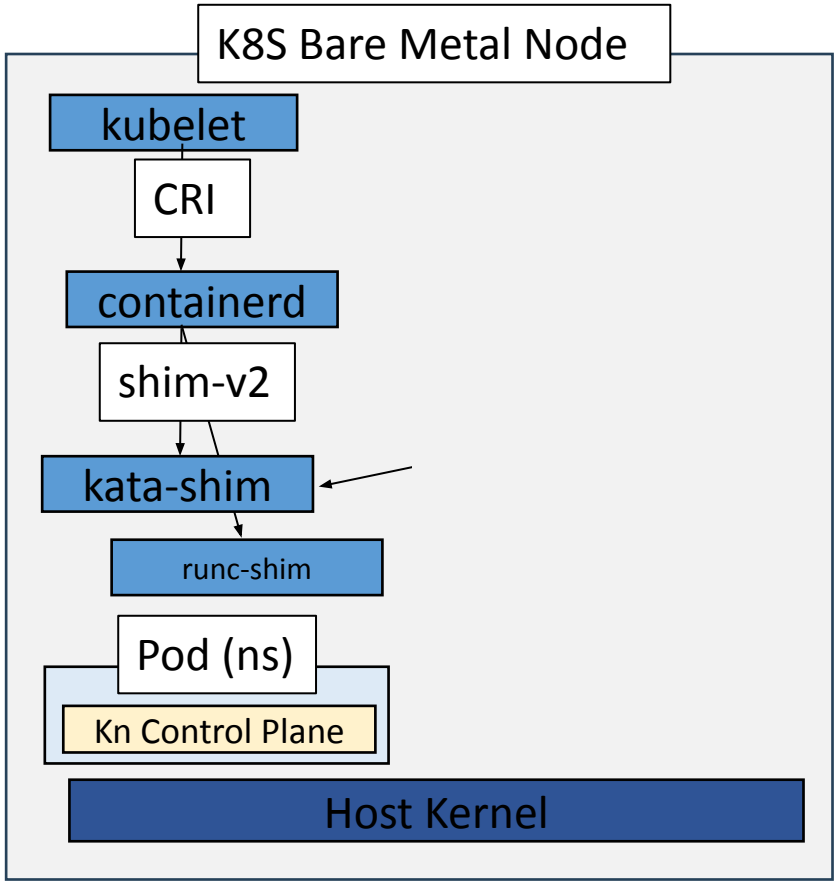
1. cVM pre-attestation
2. OVMF boot
3. Direct measured kernel boot
4. Kata Agent as /init in initrd
5. Pull encrypted/signed images
6. Request key material
7. Validate Image Signature
8. Decrypt Layers

Evaluation

We want to evaluate the feasibility of our PoC according to the three key metrics we identified for serverless:

	1. Cold Start Times	2. Warm Start Times	3. Instantiation Throughput
RUNC	6s	1s	1 fps
kata containers	7s	2s	0.5 fps
CONFIDENTIAL CONTAINERS	??	??	??

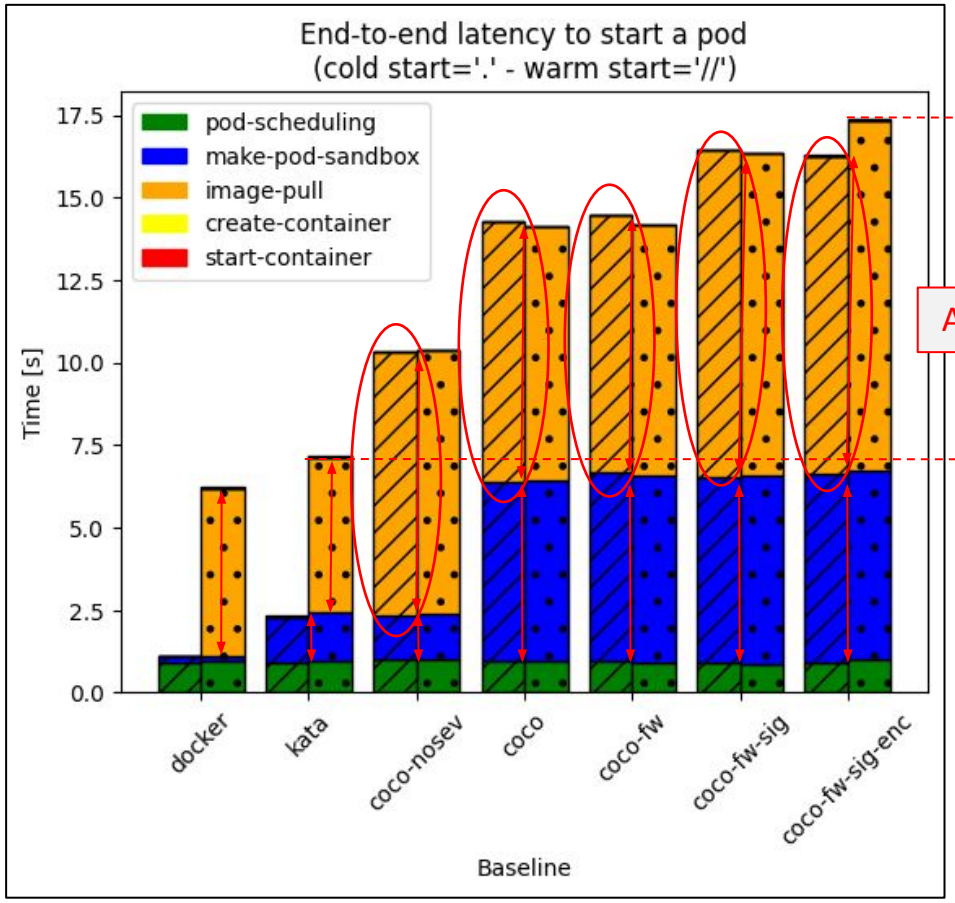
Evaluation: Baselines



0. **docker (i.e. runc):** no VMs
1. **kata:** VMs
2. **coco-nosev:** + pull in guest
3. **coco-nosev-ovmf:** + OVMF
4. **coco:** + SEV
5. **coco-fw:** + HW att
6. **coco-fw-sig:** + image signature
7. **coco-fw-sig-enc:** + image enc.

Knative Service is a simple "Hello World" in Python

Evaluation: Cold/Warm Starts



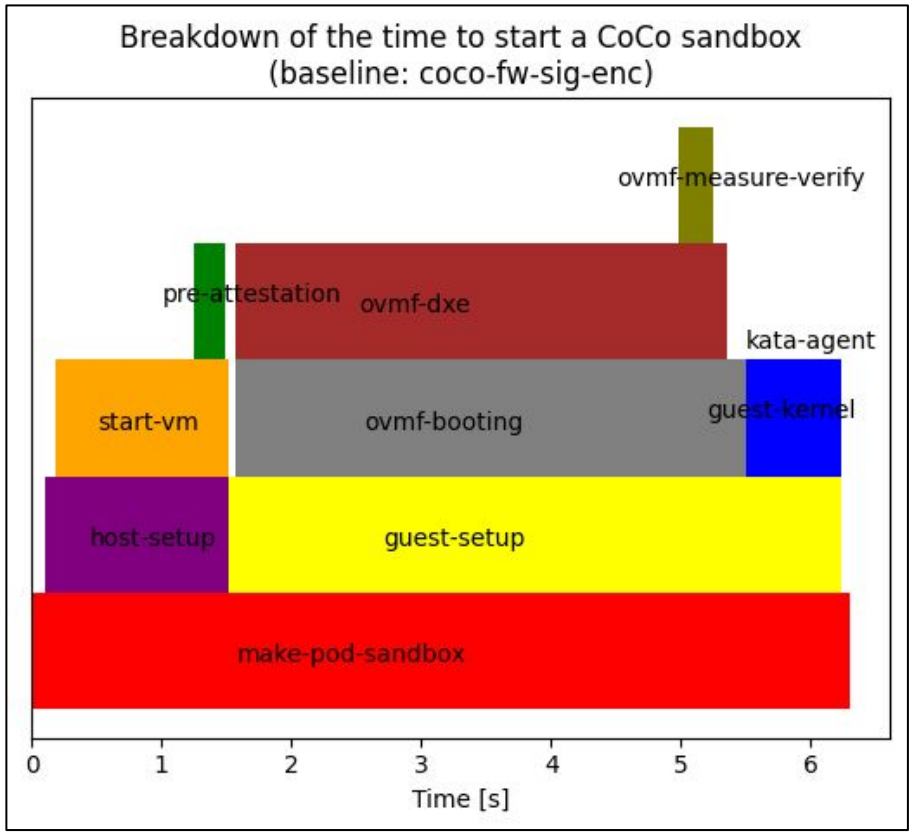
Additional 10 s!

Observations:

1. Why is VM start-up 4x slower with SEV?
2. Why is image pulling 2-3x slower w.r.t docker?
3. Why are there no warm starts?

Let us address these questions one by one

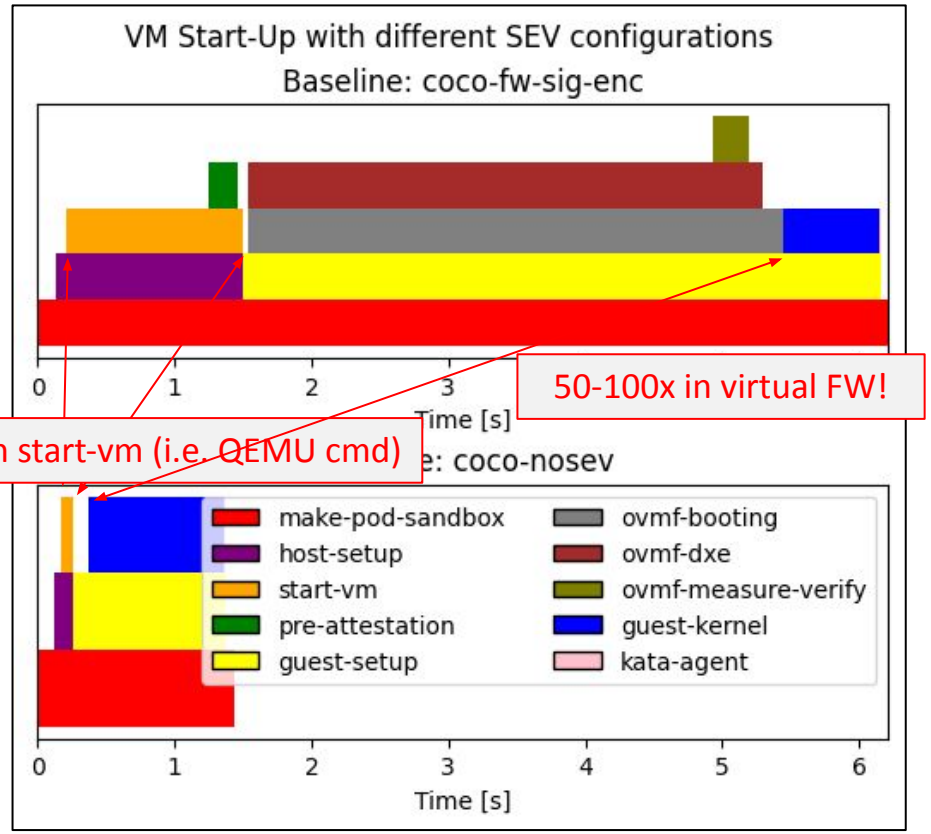
Evaluation: VM Start-Up in detail



Q1: Why is VM start-up 3x slower with SEV?

A: It seems that we spend a lot of time in OVMF...

Evaluation: VM Start-Up in detail

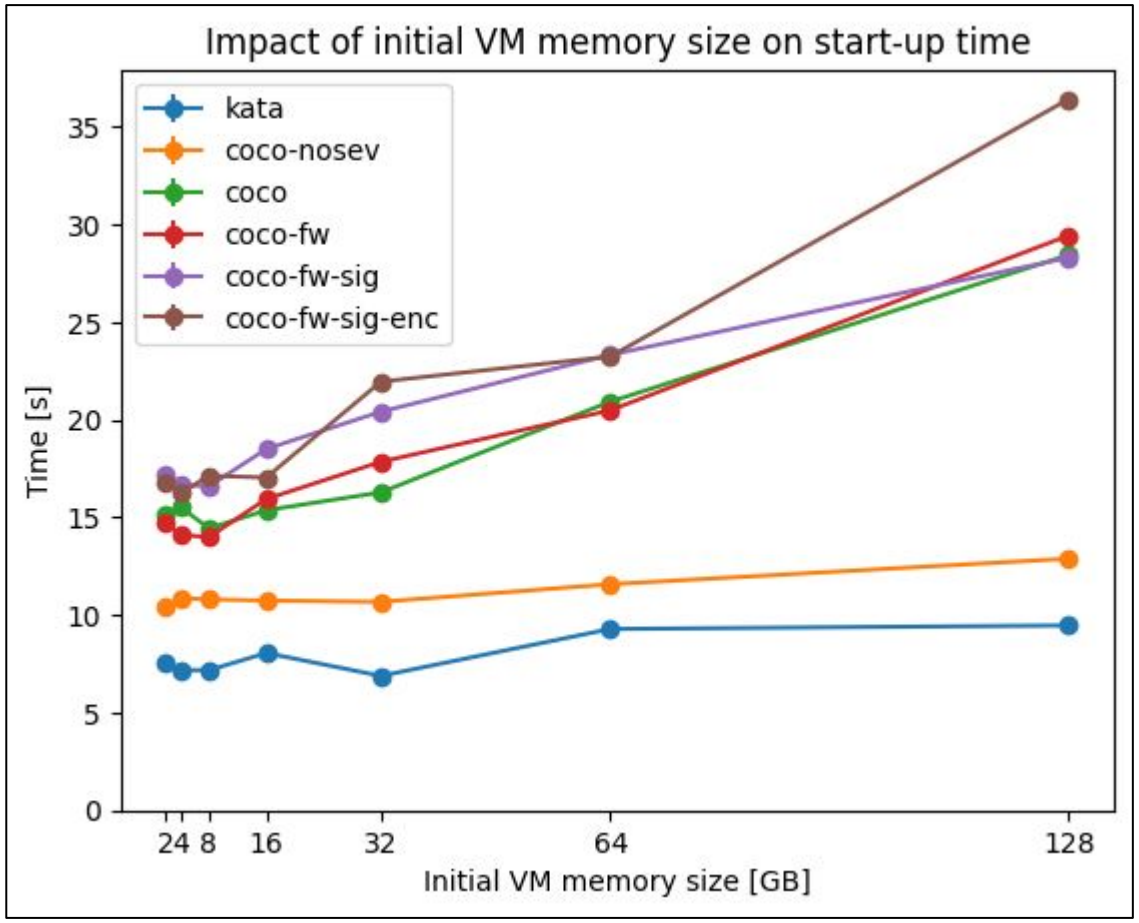


Q1: Why is VM start-up 3x slower with SEV?

start-vm: Provisioning guest memory (pages introduces 1-2 extra seconds (for 2GB of memory)

virtual-fw: OVMF DXE driver initialization introduces 3-4 extra seconds

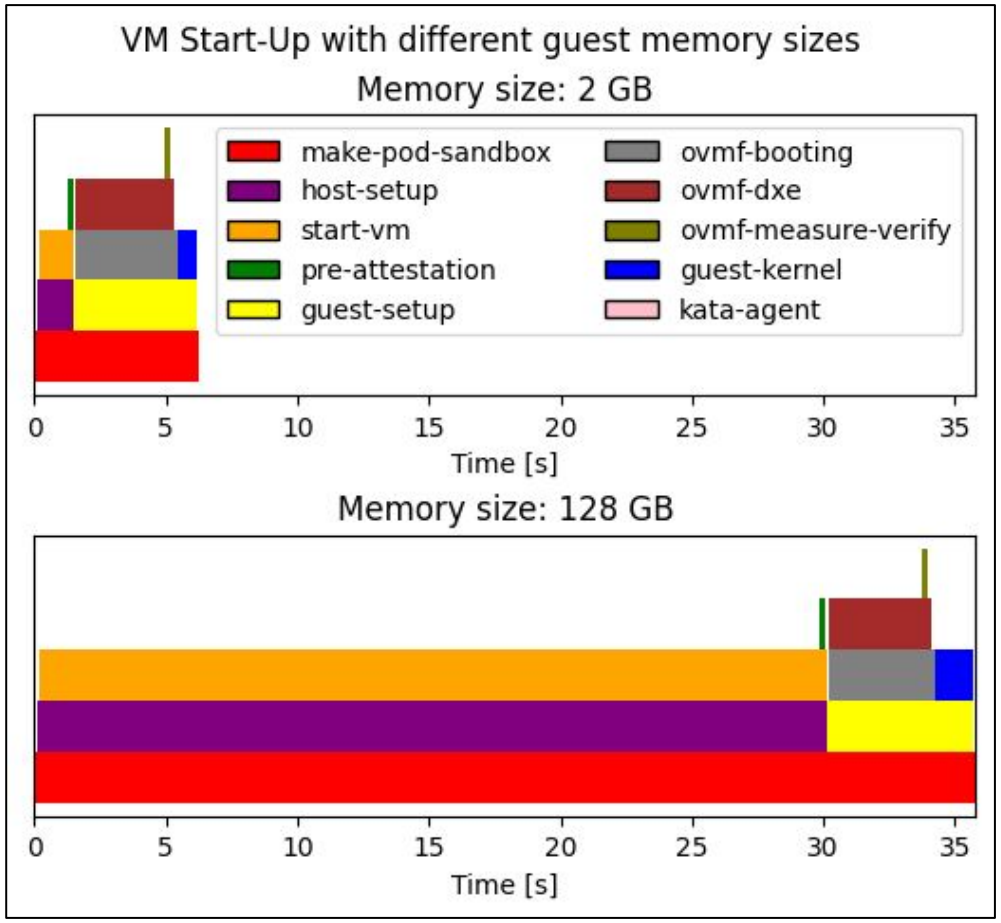
Evaluation: VM Start-Up in detail (ctd)



Q1: Why is VM start-up 3x slower with SEV?

A: During the start-vm phase, QEMU provisions all the memory pages assigned to the guest

Evaluation: VM Start-Up in detail (ctd)



Q1: Why is VM start-up 3x slower with SEV?

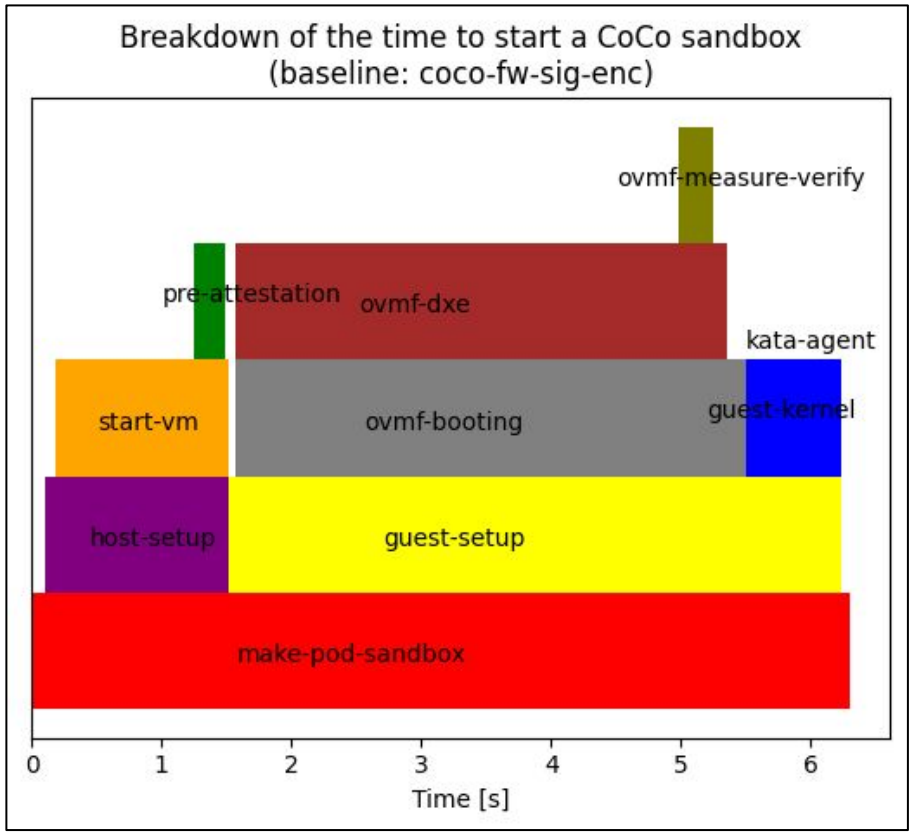
A: During the start-vm phase, the PSP provisions all the memory pages assigned to the guest

Suggested Solution:

- Can we assign memory pages lazily, off the hot-path?

Serverless CoCo Task 1: Optimize cVM provisioning

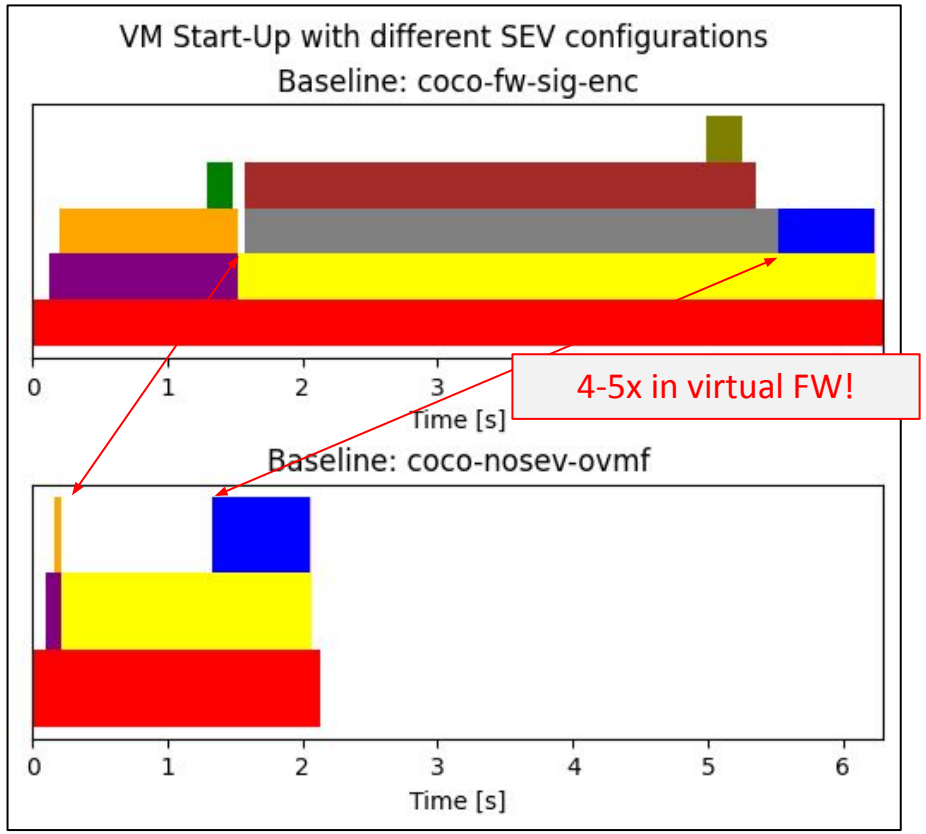
Evaluation: VM Start-Up in detail



Q1: Why is VM start-up 3x slower with SEV?

A: It seems that we spend a lot of time in OVMF...

Evaluation: VM Start-Up in detail



Q1: Why is VM start-up 3x slower with SEV?

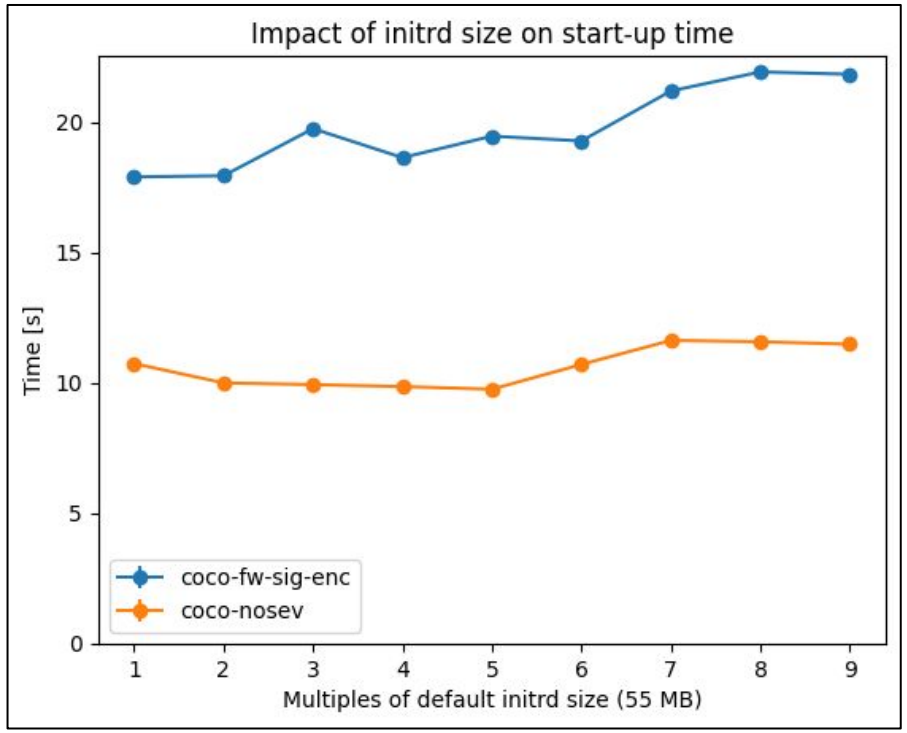
A: It seems that we spend a lot of time in OVMF...

A: Compared to a non-SEV VM (w/ OVMF) we spend:

Q: What is the difference between SEV/non-SEV OVMF?

A: For SEV, we measure and verify kernel/initrd/cmdline

Evaluation: VM Start-Up in detail



Q1: Why is VM start-up 3x slower with SEV?

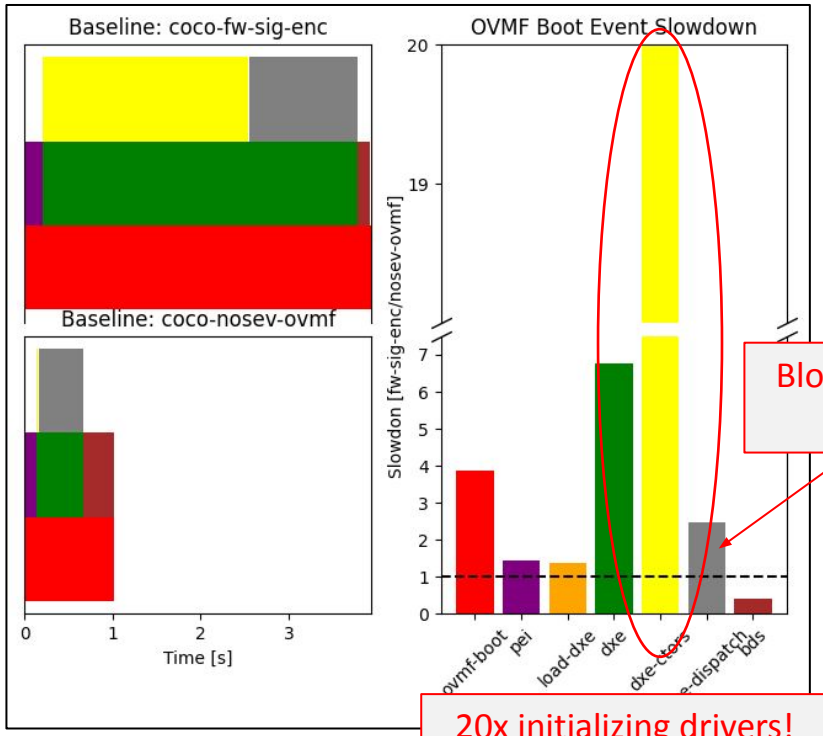
A: It seems that we spend a lot of time in OVMF...

A: Compared to a non-SEV VM (w/ OVMF) we spend:

Q: What is the difference between SEV/non-SEV OVMF?

A: For SEV, we measure and verify kernel/initrd/cmdline

Evaluation: VM Start-Up in detail



Q1: Why is VM start-up 3x slower with SEV?

A: It seems that we spend a lot of time in OVMF...

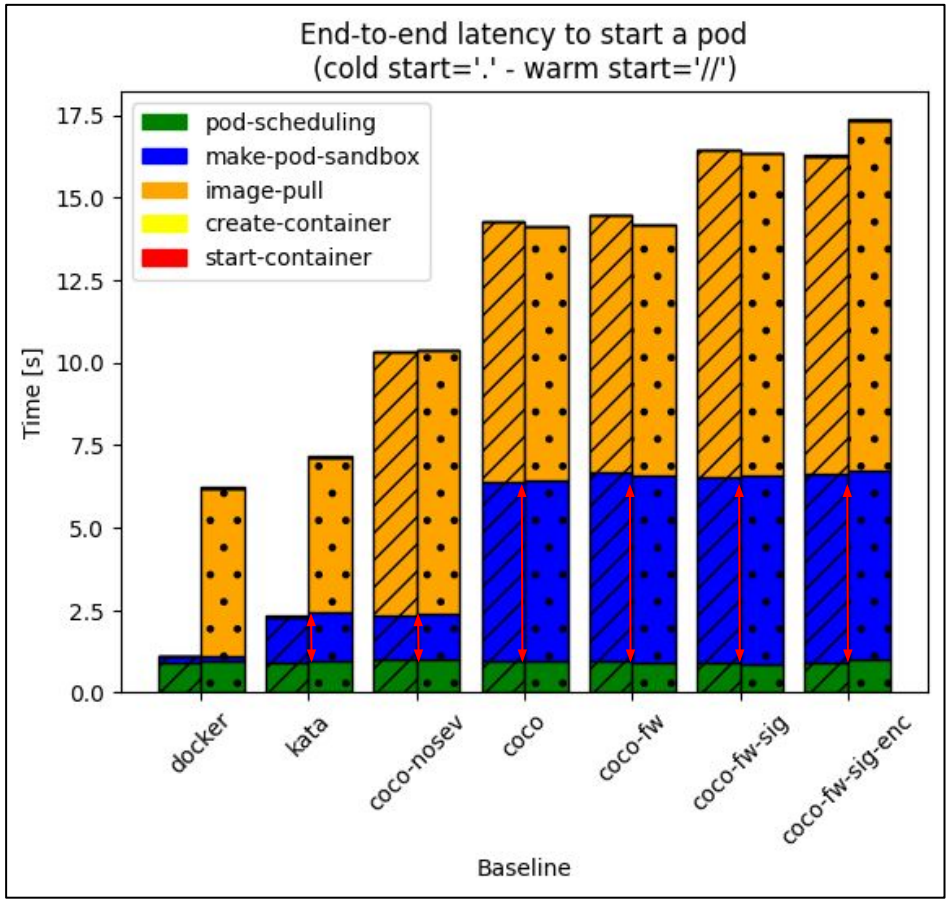
A: Compared to a non-SEV VM (w/ OVMF) we spend:

Blob measurement/verification happens here

This behaviour is unexpected (we are in contact w/ AMD about it)

20x initializing drivers!

Evaluation: Cold/Warm Starts



Observations:

1. **Why is VM start-up 4x slower with SEV?**
2. Why is image pulling 2-3x slower w.r.t docker?
3. Why are there no warm starts?

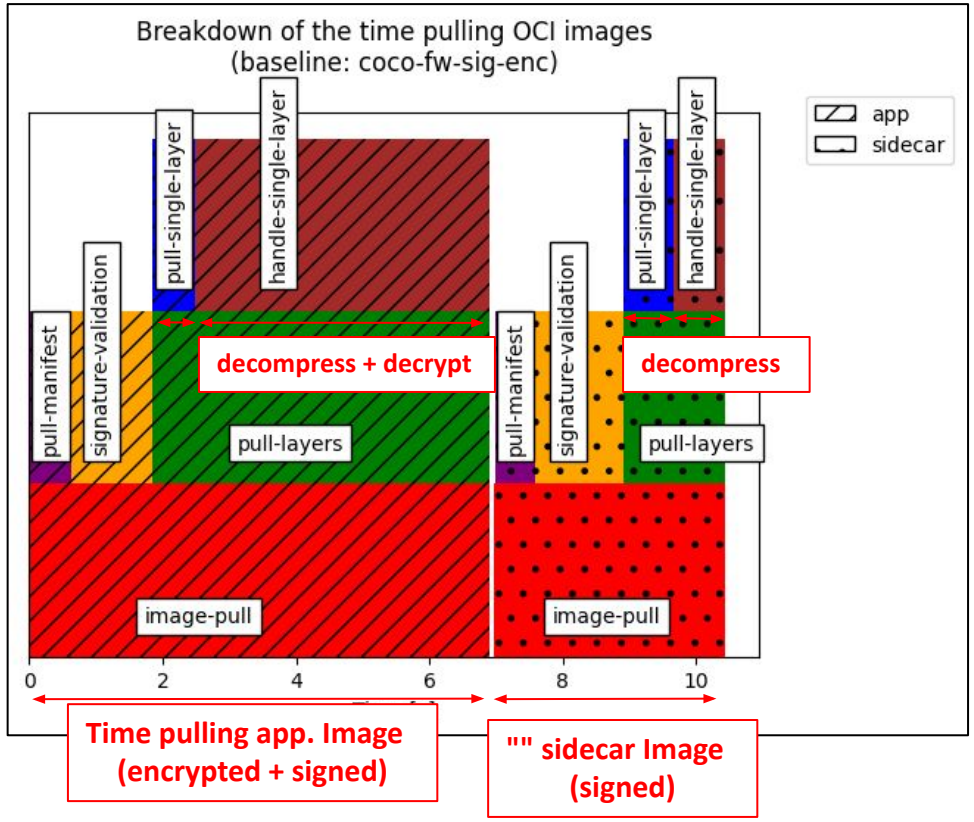
Problem: Provisioning guest memory pages introduces 1-2 extra seconds (for 2GB of memory)

Solution: Hot-Plug guest memory pages (or provision off the hot path)

Problem: OVMF DXE driver initialization introduces 3-4 extra seconds

Solution: Not clear! Talk to AMD folks!

Evaluation: VM Start-Up in detail

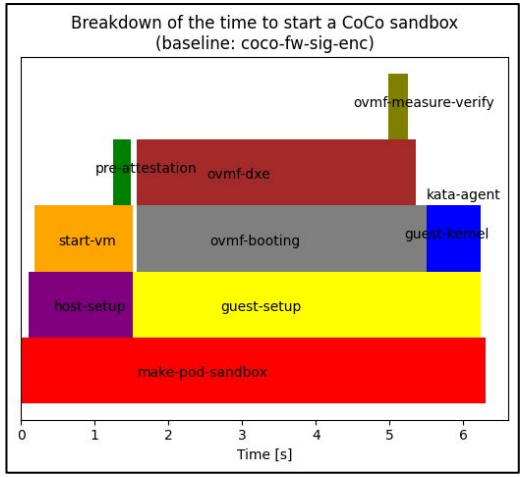


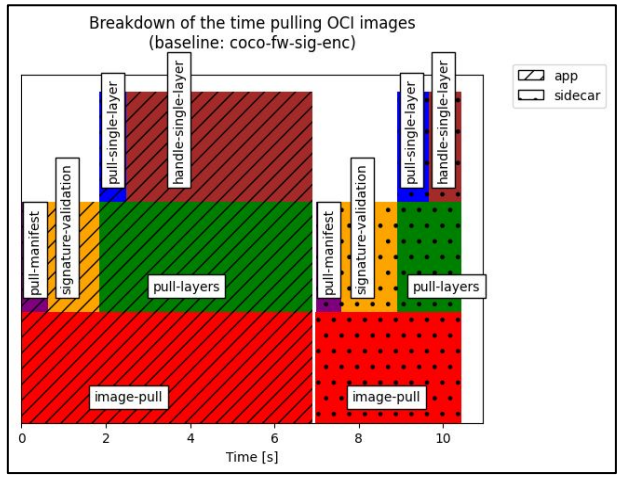
Q2: Why is image-pulling 2x slower w.r.t Docker?

A: containerd's PullImage becomes blocking!

A(ctd): Decrypting image layers is the bottleneck!

Evaluation: VM Start-Up in detail





Q3: Why are there no warm starts?

A: SEV guests are cryptographically bound to one "guest owner"

A: Cannot rely on the host to mount container images

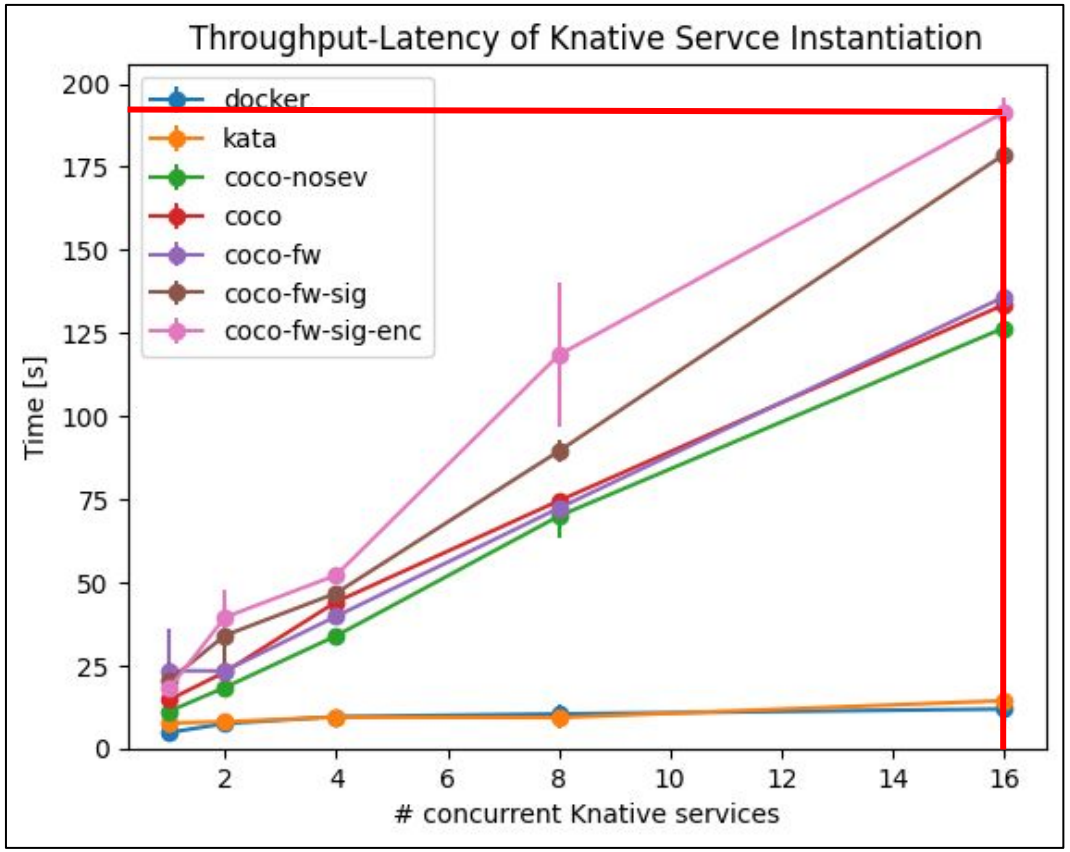
A: Cannot easily share (or lazy load) encrypted image layers

Suggested Solutions:

- Use the KBS as trusted relying-party in VM pre-warm
- Freeze the Kata Agent until pre-warmed VM is assigned
- Encrypted block-based lazy image loading (Nydus)
- Label image layers as encrypted or not

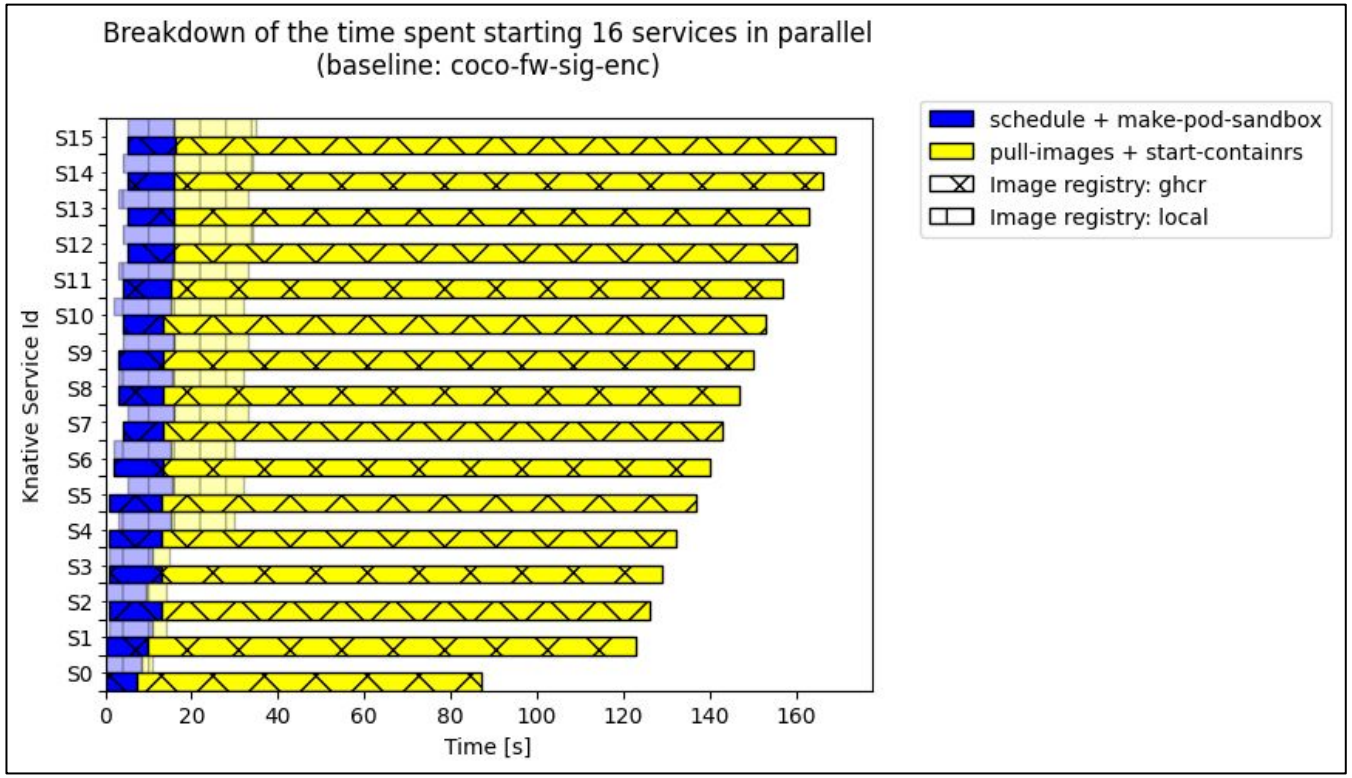
Serverless CoCo Task 3: Design Secure CoCo sandbox re-use strategies

Evaluation: Instantiation Throughput



Starting 16 concurrent functions takes > 3' !!

Evaluation: Instantiation Throughput (ctd.)



Q: Why Starting 16 concurrent functions takes > 3'?

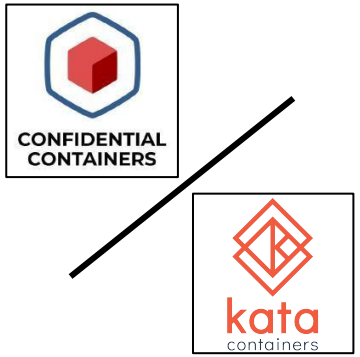
A: We are being throttled by the registry!

Evaluation

We want to evaluate the feasibility of our PoC according to the three key metrics we identified for serverless:

	1. Cold Start Times	2. Warm Start Times	3. Instantiation Throughput
RUNC	6s	1s	1 fps
kata containers	7s	2s	0.5 fps
CONFIDENTIAL CONTAINERS	17.5 s	17.5 s	~ 0.1 cps

FYP CoCo: Summary



Slowdown

1. Cold Start Times	2. Warm Start Times	3. Instantiation Throughput
2.5x	**8.75x**	**5x**
cVM Start-Up Overhead 1) Guest Memory Pages Provisioning 2) OVMF DXE Initialization	**No CoCo pre-warming** 1) SEV Guests <-> owner	**Registry Throttling** 1) If all CoCo's pull from the guest, cannot scale w/out pass-through cache 2) Will benefit from improvements in warm starts
Guest-Side Image Pulling 1) Serial (per-ctr) pulling 2) Image Layer Decryption	**No Image Re-Use** 1) Cannot mount images from host 2) Cannot share images between tenants 3) Cannot lazy load images	

Imperial College London

LSDS
Large-Scale Data & Systems Group

Serverless Confidential Containers: Challenges and Opportunities

Carlos Segarra

(w/ Tobin Feldman-Fitzthum and Daniele Buono)

Large-Scale Data & Systems (LSDS) Group - Imperial College London

Visiting IBM TJ Watson (Sep'23 – Nov'23)

https://carlossegarra.com
<cs1620@ic.ac.uk>

CLOUD STARS

IBM TJ Watson – Thursday, November 16th 2023

FYP CoCo: Summary

Confidential Containers are a very promising technology for zero-friction adoption of confidential computing in the cloud. However...

still very far from being usable in challenging environments like serverless!

Good news is that there is a lot of low-hanging fruit!

Serverless CoCo Task 1: Optimize cVM provisioning

Serverless CoCo Task 2: Optimize Image Pulling Time

Serverless CoCo Task 3: Design Secure CoCo sandbox re-use strategies

Serverless CoCo Task 4: Improve Scalability of CoCo sandbox provisioning