Five big problems with confidential containers
and why we need KVM developers to help

Christophe de Dinechin
Senior Principal Software Engineer
dinechin@redhat.com
Agenda

Key topics we are going to cover today

• Quick primer on confidential containers
Agenda

Key topics we are going to cover today

- Quick primer on confidential containers
- Attestation: ensuring you know what's running
Agenda

Key topics we are going to cover today

- Quick primer on confidential containers
- Attestation: ensuring you know what's running
- Performance overhead: A hardware vendor's dream
Agenda
Key topics we are going to cover today

- Quick primer on confidential containers
- Attestation: ensuring you know what's running
- Performance overhead: A hardware vendor's dream
- Image download: Lather, Rinse, Repeat!
Agenda

Key topics we are going to cover today

- Quick primer on confidential containers
- Attestation: ensuring you know what's running
- Performance overhead: A hardware vendor's dream
- Image download: Lather, Rinse, Repeat!
- Access control: Rethinking Kubernetes credentials
Agenda
Key topics we are going to cover today

- Quick primer on confidential containers
- Attestation: ensuring you know what's running
- Performance overhead: A hardware vendor's dream
- Image download: Lather, Rinse, Repeat!
- Access control: Rethinking Kubernetes credentials
- Debuggability: even printf-debugging is out
Confidential Containers

A quick primer
I love cunning containers as much as anyone, but I've found that if I get rid of everything I don't need, I often don't need a container at all.

Gretchen Rubin
A very active project

https://github.com/confidential-containers
Confidential Computing

Why should infrastructure see your data?

Software now runs on hardware you do not own, like a cloud provider

Virtual machine host
Confidential Computing

Why should infrastructure see your data?

Hardware resources are owned by the host

Virtual machine host

Physical resources
Confidential Computing

Why should infrastructure see your data?

Containers carve out resources from the host
Confidential Computing

Why should infrastructure see your data?

Classical sandboxing only protects the host from the containers running on it.
Confidential Computing

Why should infrastructure see your data?

The host can freely peek inside the container, for example read its memory.
Confidential Computing

Why should infrastructure see your data?

For that reason, multiple tenants do not want to share hardware when processing sensitive data.
Confidential Computing

Why should infrastructure see your data?

Data on disk or in network is already encrypted today, so the VM host cannot read it nor tamper with it.
Confidential Computing

Why should infrastructure see your data?

In non-CC architectures, data in memory is not encrypted, so it can be accessed by the host.
Integrity ensures the host cannot corrupt nor poison CPU state or RAM contents

Confidential Computing

Why should infrastructure see your data?
Confidential Computing

Why should infrastructure see your data?

Attestation proves where you are running and what you are running.
Vendor landscape

Different vendors with different approaches?

- AMD: Secure Encrypted Virtualization (SEV)
Vendor landscape

Different vendors with different approaches?

- **AMD: Secure Encrypted Virtualization (SEV)**
  - SEV-ES adds Encrypted State (e.g. CPU register file)

Security is Always a Tradeoff

...but it's very nice to have great options

**Considerations**

- Requires AMD EPYC 7xx2 CPUs
- Requires guest OS support
- vMotion, memory snapshots, hot-add, suspend/resume, Fault Tolerance, clones, and guest integrity not supported
- Support SEV-ES (memory encryption + encrypted register state), not just SEV

**Benefits**

- Workloads gain deep data-in-use protections without modification!
- Coexists with other workloads
- Containers & modern applications (Tanzu) make most operational considerations invisible
- Easy to enable & operate (PowerCLI command for the VM)
Vendor landscape

Different vendors with different approaches?

- AMD: Secure Encrypted Virtualization (SEV)
  - SEV-ES adds Encrypted State (e.g. CPU register file)
  - SEV-SNP adds Secure Nested Pages (integrity protection)
Vendor landscape

Different vendors with different approaches?

- **AMD**: Secure Encrypted Virtualization (SEV)
  - SEV-ES adds Encrypted State (e.g. CPU register file)
  - SEV-SNP adds Secure Nested Pages (integrity protection)

- **Intel**: Trusted Domain Extensions (TDX)
Vendor landscape

Different vendors with different approaches?

- **AMD**: Secure Encrypted Virtualization (SEV)
  - SEV-ES adds Encrypted State (e.g. CPU register file)
  - SEV-SNP adds Secure Nested Pages (integrity protection)

- **Intel**: Trusted Domain Extensions (TDX)

- **IBM S390**: Secure Execution (SE)
Vendor landscape

Different vendors with different approaches?

- **AMD:** Secure Encrypted Virtualization (SEV)
  - SEV-ES adds Encrypted State (e.g. CPU register file)
  - SEV-SNP adds Secure Nested Pages (integrity protection)
- **Intel:** Trusted Domain Extensions (TDX)
- **IBM S390:** Secure Execution (SE)
- **Power:** Protected Execution Facility (PEF)
Vendor landscape

Different vendors with different approaches?

- **AMD**: Secure Encrypted Virtualization (SEV)
  - SEV-ES adds Encrypted State (e.g., CPU register file)
  - SEV-SNP adds Secure Nested Pages (integrity protection)

- **Intel**: Trusted Domain Extensions (TDX)

- **IBM S390**: Secure Execution (SE)

- **Power**: Protected Execution Facility (PEF)

- **Arm**: Confidential Computing Architecture (CCA)
Vendor landscape

Different vendors with different approaches?

- AMD: Secure Encrypted Virtualization (SEV)
  - SEV-ES adds Encrypted State (e.g. CPU register file)
  - SEV-SNP adds Secure Nested Pages (integrity protection)

- Intel: Trusted Domain Extensions (TDX)

- IBM S390: Secure Execution (SE)

- Power: Protected Execution Facility (PEF)

- Arm: Confidential Computing Architecture (CCA)

- All these technologies are based on virtualization
Vendor landscape

Different vendors with different approaches?

- AMD: Secure Encrypted Virtualization (SEV)
  - SEV-ES adds Encrypted State (e.g. CPU register file)
  - SEV-SNP adds Secure Nested Pages (integrity protection)
- Intel: Trusted Domain Extensions (TDX)
- IBM S390: Secure Execution (SE)
- Power: Protected Execution Facility (PEF)
- Arm: Confidential Computing Architecture (CCA)
- All these technologies are based on virtualization
- They all work differently
Kata Containers Overview

Run containers in virtual machines
Kata Containers Overview

Run containers in virtual machines

Kubelet

Container
- Kata Agent
- VM (Linux)
- Hypervisor (qemu)
- Runtime (shim-v2)

Runtime (runc)

CRI (cri-o or containerd)

Images & Volumes

CNI

CSI
Kata Containers Overview

Run containers in virtual machines

Containers ecosystem
Virtual machine isolation
Confidential Containers in Kata

Taking advantage of confidential computing
Confidential Containers in Kata

Taking advantage of confidential computing

- **Kubelet**
- **Container**
  - **Kata Agent**
  - **VM (Linux)**
    - **Hypervisor (qemu)**
    - **Runtime (shim-v2)**
  - **Runtime (runc)**
- **CRI (cri-o or containerd)**
- **Linux kernel**
- **Firmware / Hardware**

**Trusted platform:**
Offers confidentiality guarantees enforced by hardware cryptography

**Host:**
Manages and offers resources used to run containers (CPU, memory, I/O, etc)

**Tenant:**
Confidential area, inaccessible to the host even when running on it
Phase 1: Hardware enablement

Just activating the confidential computing technologies
Phase 1: Hardware enablement

Just activating the confidential computing technologies

Hardware:
Encryption support in memory controller

Firmware:
Special services like physical page validation

Kernel:
Low-level hardware support, e.g. SEV, TDX
Phase 1: Hardware enablement

Just activating the confidential computing technologies

- Kubelet
- Container
- VM (Linux)
- Kata Agent
- Hypervisor (qemu)
- Runtime (runc)
- Runtime (shim-v2)
- Encrypted Images & Volumes
- CRI (cri-o or containerd)
- CNI
- CSI
- Key broker
- Attestation service
- Relying party

Hardware:
- Encryption support in memory controller

Firmware:
- Special services like physical page validation

Kernel:
- Low-level hardware support, e.g. SEV, TDX

Runtime:
- Pass the right options to hypervisor

Hypervisor:
- Setup virtual machines with encryption, etc

Agent:
- Image download, confidential services
Phase 1: Hardware enablement

Just activating the confidential computing technologies

**Complete!**

**Hardware:**
Encryption support in memory controller

**Firmware:**
Special services like physical page validation

**Kernel:**
Low-level hardware support, e.g. SEV, TDX

**Runtime:**
Pass the right options to hypervisor

**Hypervisor:**
Setup virtual machines with encryption, etc

**Agent:**
Image download, confidential services
Phase 2: Securing image pull

Download images from within the guest

Kubelet

Container
  Kata Agent
  VM (Linux)
    Hypervisor (qemu)
    Runtime (shim-v2)

Encrypted Images & Volumes

Relying party
  Key broker
  Attestation service

CRI (cri-o or containerd)

Runtime (runc)

CNI

CSI

Linux kernel

Firmware / Hardware
Phase 2: Securing image pull

Download images from within the guest

- **Pull Image**
  - from inside the guest instead of from the host

- **Store Images**
  - on an encrypted volume, with guest-only decryption keys

- **Architectural changes**
  - to delegate PullImage to the runtime, implemented in `containerd`, but patches rejected
Phase 2: Securing image pull

Download images from within the guest

Pull Image from inside the guest instead of from the host

Store Images on an encrypted volume, with guest-only decryption keys

Architectural changes to delegate PullImage to the runtime, implemented in containerd, but patches rejected
Attestation

Measuring what we run using cryptography

Kubelet

Container
  Kata Agent
  VM (Linux)
    Hypervisor (qemu)
    Runtime (shim-v2)

CRI (cri-o or containerd)

Runtime (runc)

CNI

CSI

Linux kernel

Firmware / Hardware

Key broker

Attestation service

Relying party
Attestation

Measuring what we run using cryptography

Preattestation:
Measure the payload before allowing it to start (original SEV)
Attestation
Measuring what we run using cryptography

Preactestation:
Measure the payload before allowing it to start (original SEV)

Postatestation:
Code in the payload can confirm its identity using the measurement in order to get secrets
Attestation

Measuring what we run using cryptography

**Preadmission:**
Measure the payload before allowing it to start (original SEV)

**Postattribution:**
Code in the payload can confirm its identity using the measurement in order to get secrets

**Workload attestation**
The workload itself is attested, e.g., gets secrets from attestation
How does attestation work?

Challenge / response to deliver secrets

- Container
- Kata Agent
- VM (Linux)
- Hypervisor (qemu)
- Runtime (shim-v2)
- Relying party
  - Key broker
  - Attestation service
How does attestation work?

Challenge / response to deliver secrets

Container

Kata Agent

VM (Linux)

Relying party

Key broker

Attestation service

Cryptographic measurement:
Measurement of relevant memory performed by hardware / firmware
How does attestation work?

Challenge / response to deliver secrets

- Container
- Kata Agent
- VM (Linux)
- Hypervisor (qemu)
- Runtime (shim-v2)
- Relying party
- Key broker
- Attestation service

**Cryptographic measurement:**
Measurement of relevant memory performed by hardware / firmware

**Cryptographic challenge:**
Send proof of identity (salted)
How does attestation work?

Challenge / response to deliver secrets

- Container
  - Kata Agent
  - VM (Linux)
- Relying party
  - Key broker
  - Attestation service

Cryptographic measurement:
Measurement of relevant memory performed by hardware / firmware

Cryptographic challenge:
Send proof of identity (salted)

Secret delivery
Ensure the workload cannot do harm if not attested
How does attestation work?

Challenge / response to deliver secrets

- **Container**
- **Kata Agent**
- **VM (Linux)**
- **Hypervisor (qemu)**
- **Runtime (shim-v2)**

**Cryptographic measurement:** Measurement of relevant memory performed by hardware / firmware

**Cryptographic challenge:** Send proof of identity (salted)

**Secret delivery**
Ensure the workload cannot do harm if not attested

**Remote attestation:**
Can invalidate workloads e.g. if compromised
So, what is the problem?

It looks like you got everything under control
Let's start with attestation

The problem with standards...
Let's start with attestation.

The good thing about standards is that there are so many to choose from.

Andrew Tannenbaum
Attestation is a very general thing

So there is More One Way To Do It™

• Remote Attestation Procedures (RATS)
Attestation is a very general thing

So there is More One Way To Do It™

- Remote Attestation Procedures (RATS)
  - A relatively complex topic in itself

- Conceptual data flow between different RATS roles
Attestation is a very general thing

So there is More One Way To Do It™

- Remote Attestation Procedures (RATS)
  - A relatively complex topic in itself
- Secure Boot and Trusted Platform Module
Attestation is a very general thing

So there is More One Way To Do It™

- Remote Attestation Procedures (RATS)
  - A relatively complex topic in itself

- Secure Boot and Trusted Platform Module
  - How about manufacturing transient virtual TPMs?
Attestation is a very general thing

So there is More One Way To Do It™

- Remote Attestation Procedures (RATS)
  - A relatively complex topic in itself
- Secure Boot and Trusted Platform Module
  - How about manufacturing transient virtual TPMs?
- Confidential Containers have their own attestation
Attestation is a very general thing
So there is More One Way To Do It™

- Remote Attestation Procedures (RATS)
  - A relatively complex topic in itself

- Secure Boot and Trusted Platform Module
  - How about manufacturing transient virtual TPMs?

- Confidential Containers have their own attestation
  - Which needs to plug into all existing technologies
Attestation is a very general thing

So there is More One Way To Do It™

- Remote Attestation Procedures (RATS)
  - A relatively complex topic in itself

- Secure Boot and Trusted Platform Module
  - How about manufacturing transient virtual TPMs?

- Confidential Containers have their own attestation
  - Which needs to plug into all existing technologies

- Hide platform differences from user-space
Attestation and Key Brokering

Attestation is robust only if it hands you secrets
Attestation and Key Brokering

Attestation is robust only if it hands you secrets

- Attestation Service (AS, aka Verifier): Checks your ID
Attestation and Key Brokering

Attestation is robust only if it hands you secrets

- Attestation Service (AS, aka Verifier): Checks your ID
- Key Broker Service (KBS): Hands over secrets
Attestation and Key Brokering

Attestation is robust only if it hands you secrets

- Attestation Service (AS, aka Verifier): Checks your ID
- Key Broker Service (KBS): Hands over secrets
- Your workload must need that secret
Attestation and Key Brokering

Attestation is robust only if it hands you secrets

- Attestation Service (AS, aka Verifier): Checks your ID
- Key Broker Service (KBS): Hands over secrets
- Your workload must need that secret
- KBS protocol defined by Confidential Containers
Attestation and Key Brokering

Attestation is robust only if it hands you secrets

- Attestation Service (AS, aka Verifier): Checks your ID
- Key Broker Service (KBS): Hands over secrets
- Your workload must need that secret

- KBS protocol defined by Confidential Containers
  - ... first implemented by Sergio Lopez for libkrun
Attestation and Key Brokering

Attestation is robust only if it hands you secrets

- Attestation Service (AS, aka Verifier): Checks your ID
- Key Broker Service (KBS): Hands over secrets
- Your workload must need that secret

- KBS protocol defined by Confidential Containers
  - ... first implemented by Sergio Lopez for libkrun

- Services will come from different players

Microsoft Azure Attestation

Microsoft Azure Attestation is a unified solution for remotely verifying the trustworthiness of a platform and integrity of the binaries running inside it. The service supports attestation of the platforms backed by Trusted Platform Modules (TPMs) alongside the ability to attest to the state of Trusted Execution Environments (TEEs) such as Intel® Software Guard Extensions (SGX) enclaves, Virtualization-based Security (VBS) enclaves, Trusted Platform Modules (TPMs), Trusted launch for Azure VMs and Azure confidential VMs.

Attestation is a process for demonstrating that software bin is instantiated on a trusted platform. Remote relying parties can be confident that only such intended software is running on the Azure Attestation is a unified customer-facing service and facilitates attestation.
Attestation and Key Brokering

Attestation is robust only if it hands you secrets

- Attestation Service (AS, aka Verifier): Checks your ID
- Key Broker Service (KBS): Hands over secrets
- Your workload must need that secret

- KBS protocol defined by Confidential Containers
  - ... first implemented by Sergio Lopez for libkrun

- Services will come from different players
  - So we need to define protocols, not just code
Then there is performance

A hardware vendor's dream...
A state-of-the-art calculation requires 100 hours of CPU time on the state-of-the-art computer, independent of the decade.

Edward Teller
The cost of confidentiality

Confidentiality gets in the way of deduplication
The cost of confidentiality

Confidentiality gets in the way of deduplication

• Downloaded images are now encrypted
The cost of confidentiality

Confidentiality gets in the way of deduplication

- Downloaded images are now encrypted
  - So they can't be shared between pods
The cost of confidentiality

Confidentiality gets in the way of deduplication

- Downloaded images are now encrypted
  - So they can't be shared between pods
  - This increases disk and networking costs
The cost of confidentiality

Confidentiality gets in the way of deduplication

- Downloaded images are now encrypted
  - So they can't be shared between pods
  - This increases disk and networking costs
- Disks too must be crypted
The cost of confidentiality
Confidentiality gets in the way of deduplication

- Downloaded images are now encrypted
  - So they can't be shared between pods
  - This increases disk and networking costs

- Disks too must be crypted
  - Therefore, no deduplication
The cost of confidentiality

Confidentiality gets in the way of deduplication

- Downloaded images are now encrypted
  - So they can't be shared between pods
  - This increases disk and networking costs

- Disks too must be encrypted
  - Therefore, no deduplication

- Memory is crypted too
The cost of confidentiality

Confidentiality gets in the way of deduplication

- Downloaded images are now encrypted
  - So they can't be shared between pods
  - This increases disk and networking costs
- Disks too must be crypted
  - Therefore, no deduplication
- Memory is crypted too
  - No kernel same-page merging
The cost of confidentiality

Confidentiality gets in the way of deduplication

- Downloaded images are now encrypted
  - So they can't be shared between pods
  - This increases disk and networking costs

- Disks too must be crypted
  - Therefore, no deduplication

- Memory is crypted too
  - No kernel same-page merging

- Who has a patch for Crypto KSM?
Good news: manageable runtime cost

We encrypt memory on the fly, but it's "cheap"
Image (re)download

The "Lather, Rinse, Repeat" school of programming
History is always repeating itself, but each time the price goes up

Will Durant
Cache + Dedup + Crypto?

Can we get something like Nydus on encrypted blocks?
Cache + Dedup + Crypto?

Can we get something like Nydus on encrypted blocks?
Cache + Dedup + Crypto?
Can we get something like Nydus on encrypted blocks?

The tree looks something like:

\[
\text{alg = sha256, num\_blocks = 32768, block\_size = 4096}
\]

```
[ root ]
 /  ...
[entry_0]
 /  ...
[entry_0_0]  ...  [entry_0_127]  ...  [entry_1_127]
 /  ...
[entry_0_0]  ...  [entry_0_127]  ...  [entry_1_127]
 /  ...
blk_0  ...  blk_127  blk_16256  blk_16383  blk_32640  ...  blk_32767
```

Cache + Dedup + Crypto?

Can we get something like Nydus on encrypted blocks?

- To attest or decrypt, you need the entire layer

Encrypted layers (at least one)

Clear text layers (still attested)
Cache + Dedup + Crypto?
Can we get something like Nydus on encrypted blocks?

Building encrypted & signed images

- **Generating an image** that cannot be used without a secret
- **Use OCI standard** encrypted image format – which is still actively developed
- **Build pipeline now has two outputs**, one to image registry, the other to key broker service

Security realms

- **Standard**: Normal security applies (e.g. corporate)
- **Public**: Untrusted infrastructure, encryption applies
- **Owner**: Trusted infrastructure
Cache + Dedup + Crypto?
Can we get something like Nydus on encrypted blocks?

Use the host as a block cache for image downloads by the guest

- Use **device mapper** in guest to deal with (block level) encryption and integrity
- Use **nbdkit** on the host side to deliver blocks, with local caching

Instead of storing a whole encrypted layer on local disk, we use the host + nbdkit as a cache of encrypted blocks.

This makes it possible for the host to manage the associated disk space more efficiently, e.g. share blocks across pods using the same image.

The underlying host-guest relationship becomes closer to what exists in regular kata, but using virtio-blk instead of virtiofs.
Access Control

We just need to rewrite the Kubernetes access-control model from scratch
For every problem, there is a solution that is simple, clean and wrong

Henry Louis Mencken
Going from two to three
It's the beginning of counting, folks

Red Hat Marketplace

Kubelet

Container
  Kata Agent
  VM (Linux)

Runtime (runc)

CRI (cri-o or containerd)

Linux kernel

Firmware / Hardware

Relying party
  Key broker
  Attestation service

Trusted platform:
Offers confidentiality guarantees enforced by hardware cryptography

Host:
Manages and offers resources used to run containers (CPU, memory, I/O, etc)

Tenant:
Confidential area, inaccessible to the host even when running on it
The host/tenant split API

Let's just decide which credentials we use on the fly

Cluster user: kubectl $blah $args

Host user with host credentials

kubelet

etcd

CRIOW

containerd

kata-shim-v3

Tenant user with tenant credentials

magic-smoke (TBD)

kata-shim-v3

VMM

(Cloud-Hypervisor/JEMU)

kata-agent

initrd

kernel

No vsock
Debaggability

The FBI school of debugging
A nation can't solve what the press won't let it perceive

Julian Assange
Tell us what you need to know and we'll teach you why you don't need it.
Tell us what you need to know
and we'll teach you why you don't need it

It's not a suggestion, it's a rule
Really, we mean it

if you `panic()`, we offer a cryptographic, hardware-enforced guarantee that you won't see the logs

Any good ideas on how to address this in a sensible manner is welcome
Conclusion

Confidential Containers are an opportunity but a real challenge in terms of everything
Key takeaways

High cost of confidentiality and sandboxing

- Release 1 of confidential containers is around the corner
- Practical, portable, uniform attestation is a challenge
- Performance issues are mostly vastly enhanced resource usage
- Image download is a huge contributor to this
- Access control has to be re-thought (not in release 1)
- Debuggability or confidentiality, pick one
Thank you

Now is a good time for questions

This Tao3D presentation is available at https://github.com/c3d/presentations (branch kvm-forum-2022)
Overflow sides
https://github.com/c3d/presentations

This Tao3D presentation is available at https://github.com/c3d/presentations (branch kvm-forum-2022)
Attestation process

Exchanging keys with the relying party
Immutable Pods

You cannot change what you certified

- **Hot Plugging** is currently used to add memory, CPU or devices to the pod
  - The Pod APIs do not give us the information about container sizes
  - Resources are dynamically added at container creation time
  - This adds a lot of complexity to the runtime, and inefficiency (e.g. fat page tables)
- **Integrity** is hard to guarantee if you can change the configuration at runtime
  - Memory hot-plugging or ballooning mechanisms conflict with encryption / validation
  - Devices, notably pass-through PCI devices with DMA, are also problematic
- **Immutable Pods** are fully defined ahead of time, before booting the virtual machine
  - This requires many changes in the existing Kubernetes APIs
  - Existing APIs may put things “in the wrong place”, e.g. send logs to the host.
  - This will simplify and optimize the non-confidential case, e.g. remove hot-plugging
Two Control Planes
Host and Tenant security realms

- **Tenants** need their own isolated administrative realm (logs, container metrics, ...)
- **Hosts** manage physical resources (pod creation/destruction, raw disks, H/W metrics...)

Diagram:
- Host user with host credentials
  - kubelet
  - etcd
  - CRIQ containerd
  - kata-shim-v3
  - VMM
  - Create Pod with complete description (not piecewise)

- Confidential VM
  - Container
    - kata-agent
    - initrd
    - kernel
  - No vsock

- Tenant user with tenant credentials
  - kubelet
  - etcd
  - CRIQ containerd
  - kata-shim-v3
  - VMM
  - All APIs except pod lifetime (create, kill) go through confidential channel
Performance considerations

Impact of running virtualized
Memory density

How many containers can you fit?
Memory usage

Death by a thousand cuts

Host kernel and user-space programs
Memory usage

Death by a thousand cuts

```
podman run -it --rm fedora bash
```
Memory usage

Running bash

```
podman run --it --rm fedora bash
```

Actual workload you are running

Process monitoring your container

Host kernel and user-space programs
Memory usage

Running a web server

```
podman run -dt --pod new:nginx -p8080:80 nginx
```

- Actual workload you are running
- Process monitoring your container
- Host kernel and user-space programs
Memory usage

Kubernetes overhead

```
kubectl apply -f nginx.yaml
```

- **2G**
  - 8.4MB `nginx`
    - Actual workload you are running

- **1G**
  - 2.5MB `Common`
    - Process monitoring your container

  - 110MB `Kubelet`
  - 950MB `Host`

- **Host kernel and user-space programs**

- **Top-level control process**
Memory usage

Kubernetes overhead

```
kubectl apply -f nginx.yaml
```

2G

- **8.4MB**: nginx
  - Actual workload you are running

1G

- **2.5MB**: Common
- **27.9MB**: CRI-O
- **110MB**: Kubelet
- **950MB**: Host

- **Process monitoring your container**
- **Container runtime interface**
- **Top-level control process**
- **Host kernel and user-space programs**
Memory usage

Running 100 containers
(yaml file with 100x nginx)

Actual workload you are running

Process monitoring your container

Container runtime interface
Top-level control process
Host kernel and user-space programs
Memory usage

Overhead varies with number of containers

```
kubectl apply -f nginx.yaml
```

Actual workload you are running

- 8.4MB nginx
- 2.5MB Common
- 28MB CRI-O
- 110MB Kubelet
- 950MB Host

Process monitoring your container

Container runtime interface

Top-level control process

Host kernel and user-space programs
Memory usage

Kata adds even more overhead

`podman run --it --rm --runtime kata fedora bash`

Actual workload you are running

- 8.4MB: Nginx
- 2.5MB: Common
- 25MB: Kata runtime
- 28MB: CRI-O
- 110MB: Kubelet
- 950MB: Host

Process monitoring your container
Kata runtime (transient)
Container runtime interface
Top-level control process
Host kernel and user-space programs
Memory usage

Kata adds even more overhead

```
podman run -it --rm --runtime kata fedora bash
```

Actual workload you are running

I/O interface for container

Process monitoring your container

Kata runtime (transient)

Container runtime interface

Top-level control process

Host kernel and user-space programs
Memory usage

Kata adds even more overhead

```
podman run --it --rm --runtime kata fedora bash
```

Actual workload you are running

Virtual machine monitor
I/O interface for container

Process monitoring your container
Kata runtime (transient)
Container runtime interface
Top-level control process
Host kernel and user-space programs
Memory usage

Kata adds even more overhead

```
podman run -it --rm --runtime kata fedora bash
```

Actual workload you are running

1G

- 8.4MB nginx
- 30MB QEMU
- 28MB CRI-O
- 110MB Kubelet
- 950MB Host

2G

- 16.8MB Kata Shim
- 2.5MB Common
- 25MB Kata Runtime

Virtual machine monitor
I/O interface for container
Host file access from guest
Process monitoring your container
Kata runtime (transient)
Container runtime interface
Top-level control process
Host kernel and user-space programs
Memory usage

Kata adds even more overhead

```
podman run --it --rm --runtime kata fedora bash
```

- Actual workload you are running
- Guest OS kernel, data, user-space
- Virtual machine monitor
- I/O interface for container
- Host file access from guest
- Process monitoring your container
- Kata runtime (transient)
- Container runtime interface
- Top-level control process
- Host kernel and user-space programs
Memory usage

Kata adds even more overhead

```
podman run -it --rm --runtime kata fedora bash
```

- **Actual workload you are running**
- **Kata agent in the guest**
- **Guest OS kernel, data, user-space**
- **Virtual machine monitor**
- **I/O interface for container**
- **Host file access from guest**
- **Process monitoring your container**
- **Kata runtime (transient)**
- **Container runtime interface**
- **Top-level control process**
- **Host kernel and user-space programs**
Memory usage

Kata Runtime is transient

```
podman run -it --rm --runtime kata fedora bash
```

- **Actual workload you are running**
- **Kata agent in the guest**
- **Guest OS kernel, data, user-space**
- **Virtual machine monitor**
- **I/O interface for container**
- **Host file access from guest**
- **Process monitoring your container**

- **Container runtime interface**
- **Top-level control process**
- **Host kernel and user-space programs**
Memory usage

I/Os consume memory (buffers, etc)

`fedc07a13# dnf install --y procps-ng`

- **Actual workload you are running**
- **Kata agent in the guest**
- **Guest OS kernel, data, user-space**
- **Virtual machine monitor**
- **I/O interface for container**
- **Host file access from guest**
- **Process monitoring your container**

- **Container runtime interface**
- **Top-level control process**
- **Host kernel and user-space programs**

Possible double accounting of the same physical memory between guest and virtiofsd
Memory usage

Running 100 containers
(runtimeClass kata, 100x nginx)

- Actual workload you are running
- Kata agent in the guest
- Guest OS kernel, data, user-space
- Virtual machine monitor
- I/O interface for container
- Host file access from guest
- Process monitoring your container

Container runtime interface
- Top-level control process
- Host kernel and user-space programs

Possible double accounting of the same physical memory between guest and virtiofsd
Memory usage

Running 100 containers
(runtimeClass kata, 100x nginx)

- Actual workload you are running
- Kata agent in the guest
- Guest OS kernel, data, user-space
- Virtual machine monitor
- I/O interface for container
- Host file access from guest
- Process monitoring your container

Possible double accounting of the same physical memory between guest and virtiofsd

- Container runtime interface
- Top-level control process
- Host kernel and user-space programs
Memory usage

Running 100 containers
(runtimeClass kata, 100x nginx)

- Actual workload you are running
- Kata agent in the guest
- Guest OS kernel, data, user-space
- Virtual machine monitor
- I/O interface for container
- Host file access from guest
- Process monitoring your container
- Container runtime interface
- Top-level control process
- Host kernel and user-space programs
Memory usage

Running 100 containers
(runtimeClass kata, 100x nginx)

- Actual workload you are running
- Kata agent in the guest
- Guest OS kernel, data, user-space
- Virtual machine monitor
- I/O interface for container
- Host file access from guest
- Process monitoring your container
- Container runtime interface
- Top-level control process
- Host kernel and user-space programs
Memory usage

Kata Containers 2.0: Containerd Shim v2

Containerd-shim-kata-v2

- Actual workload you are running
- Kata agent in the guest
- Guest OS kernel, data, user-space
- Virtual machine monitor
- I/O interface for container
- Host file access from guest

- Container runtime interface
- Top-level control process
- Host kernel and user-space programs
Memory - Comparing runtimes

Cost of the runtime to run 10 instances of nginx
Memory - Comparing runtimes

Cost of the runtime to run 10 instances of nginx
Memory - Comparing runtimes

Cost of the runtime to run 10 instances of nginx

[Diagram showing memory usage for different runtimes: CRIO, Kubernetes, One pod, Kata 1.12]
Memory - Comparing runtimes

Cost of the runtime to run 10 instances of nginx
Memory - Comparing runtimes

Cost of the runtime to run 10 instances of nginx

- CRI-O
- Kubernetes
- Kata Onepod
- Kata 1.12 Tenpods
- Kata 1.12 Onepod
- Kata 2.0
Memory - Comparing runtimes

Cost of the runtime to run 10 instances of nginx
Memory - Comparing runtimes

Cost of the runtime to run 10 instances of nginx

- Kata requires a lot of memory for qemu, guest kernel and virtiofs.
- Reusing pods can help, but is not common practice.
Memory - disk caches

Perform a dnf install inside the running pods
Memory - disk caches

Perform a dnf install inside the running pods
Memory - disk caches

Perform a dnf install inside the running pods
Memory - disk caches

Perform a dnf install inside the running pods

- One pod, Kata: 561M
- Ten pods, Kata: 921M
- One pod, after I/Os: 911M
- Ten pods, after I/Os: 3961M
Memory - disk caches

Perform a dnf install inside the running pods

- I/O buffers consume a lot of memory.
- DAX allows host and guests to share that memory
Memory overhead - It's bad

Huge effort needed to make things better
Memory overhead - It's bad

Huge effort needed to make things better

- Too many processes

| Processes | | | | | | |
| --- | --- | --- | --- | --- | --- | |
| 3949656 root 20 0 79.4m 2.2m 1.9m S 0.0 0.0 0.0 0.0 0:00.00 | - common | | | | | |
| 3949761 root 20 0 85.1m 4.3m 3.8m S 0.0 0.0 0.0 0.0 0:00.00 | - virtiofsd | | | | | |
| 3949788 root 20 0 1071.8m 250.6m 242.4m S 0.0 1.6 0.0 0.0 0:01.41 | - virtiofsd | | | | | |
| 3949781 root 20 0 2379.4m 594.9m 568.0m S 0.3 3.7 0.0 0.0 0:17.99 | - qemu-system-x86 | | | | | |
| 3952158 root 20 0 1351.8m 14.5m 8.7m S 0.0 0.1 0:00.00 | - keta-shim | | | | | |
| 3953366 root 20 0 79.4m 2.2m 2.0m S 0.0 0.0 0.0 0.0 0:00.00 | - common | | | | | |
| 3953461 root 20 0 1207.8m 14.8m 9.1m S 0.0 0.1 0:00.00 | - keta-shim | | | | | |
| 3954774 root 20 0 79.4m 2.3m 2.0m S 0.0 0.0 0.0 0.0 0:00.00 | - common | | | | | |
| 3954896 root 20 0 1278.7m 14.9m 9.1m S 0.0 0.1 0:00.00 | - keta-shim | | | | | |
| 3956157 root 20 0 79.4m 2.3m 2.1m S 0.0 0.0 0.0 0.0 0:00.00 | - common | | | | | |
| 3956270 root 20 0 1350.9m 14.5m 8.9m S 0.0 0.1 0:00.00 | - keta-shim | | | | | |
| 3957451 root 20 0 79.4m 2.2m 1.9m S 0.0 0.0 0.0 0.0 0:00.00 | - common | | | | | |
| 3957566 root 20 0 1350.7m 14.6m 9.1m S 0.0 0.1 0:00.00 | - common | | | | | |
| 3958776 root 20 0 79.4m 2.3m 2.1m S 0.0 0.0 0.0 0.0 0:00.00 | - keta-shim | | | | | |
| 3958891 root 20 0 1350.7m 14.3m 8.6m S 0.0 0.1 0:00.00 | - keta-shim | | | | | |
| 3960230 root 20 0 79.4m 2.2m 2.0m S 0.0 0.0 0.0 0.0 0:00.00 | - common | | | | | |
| 3960373 root 20 0 1422.7m 14.7m 9.1m S 0.0 0.1 0:00.00 | - keta-shim | | | | | |
| 3961632 root 20 0 79.4m 2.3m 2.0m S 0.0 0.0 0.0 0.0 0:00.00 | - common | | | | | |
| 3961757 root 20 0 1207.8m 14.0m 8.4m S 0.0 0.1 0:00.00 | - keta-shim | | | | | |
| 3962991 root 20 0 79.4m 2.3m 2.1m S 0.0 0.0 0.0 0.0 0:00.00 | - common | | | | | |
| 3963137 root 20 0 1278.4m 14.6m 9.0m S 0.0 0.1 0:00.00 | - keta-shim | | | | | |
| 3964472 root 20 0 79.4m 2.2m 2.0m S 0.0 0.0 0.0 0.0 0:00.00 | - common | | | | | |
| 3965589 root 20 0 1279.5m 14.5m 8.6m S 0.0 0.1 0:00.00 | - keta-shim | | | | | |
| 3966529 root 20 0 79.4m 2.2m 2.0m S 0.0 0.0 0.0 0.0 0:00.00 | - common | | | | | |
| 3966963 root 20 0 1278.7m 14.7m 9.1m S 0.0 0.1 0:00.00 | - keta-shim | | | | | |
| 3968240 root 20 0 79.4m 2.2m 2.0m S 0.0 0.0 0.0 0.0 0:00.00 | - common | | | | | |
| 3968379 root 20 0 1278.7m 14.4m 8.6m S 0.0 0.1 0:00.00 | - keta-shim | | | | | |
Memory overhead - It's bad

Huge effort needed to make things better

- Too many processes
  - Partially addressed by 2.0
Memory overhead - It's bad
Huge effort needed to make things better

- Too many processes
  - Partially addressed by 2.0
  - ... but single point of failure

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU Alloc</th>
<th>Memory Alloc</th>
<th>Swap</th>
<th>RSS</th>
<th>VmSize</th>
<th>Times</th>
<th>User</th>
<th>System</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>779277</td>
<td>530.9m</td>
<td>1.7m</td>
<td>0.0</td>
<td>0.4</td>
<td>5:20.41</td>
<td>crio</td>
<td>0.0</td>
<td>0.3</td>
<td>0.0</td>
</tr>
<tr>
<td>797519</td>
<td>530.9m</td>
<td>1.7m</td>
<td>0.0</td>
<td>0.4</td>
<td>5:20.41</td>
<td>containerd-shim</td>
<td>0.0</td>
<td>0.3</td>
<td>0.0</td>
</tr>
<tr>
<td>787528</td>
<td>530.9m</td>
<td>1.7m</td>
<td>0.0</td>
<td>0.4</td>
<td>5:20.41</td>
<td>virtiofsd</td>
<td>0.0</td>
<td>0.3</td>
<td>0.0</td>
</tr>
<tr>
<td>787543</td>
<td>530.9m</td>
<td>1.7m</td>
<td>0.0</td>
<td>0.4</td>
<td>5:20.41</td>
<td>virtiofsd</td>
<td>0.0</td>
<td>0.3</td>
<td>0.0</td>
</tr>
<tr>
<td>797538</td>
<td>530.9m</td>
<td>1.7m</td>
<td>0.0</td>
<td>0.4</td>
<td>5:20.41</td>
<td>qemu-system-x86</td>
<td>0.0</td>
<td>0.3</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Memory overhead - It's bad

Huge effort needed to make things better

- Too many processes
  - Partially addressed by 2.0
  - ... but single point of failure
- A lot of memory in buffer caches
Memory overhead - It's bad

Huge effort needed to make things better

- Too many processes
  - Partially addressed by 2.0
  - ... but single point of failure

- A lot of memory in buffer caches
  - DAX allows host buffer caches to be used
Boot time
How fast does a container start?
Boot time overhead

Time to boot containers

1s

0.5

0.3s

ix busybox on
runc
Boot time overhead

Time to boot containers

- ix busybox on: 0.3s
- ix busybox on: 7.1s
- Kata: 0.3s
Boot time overhead

Time to boot containers

- ix/busybox on
- ix/busybox on
- Kata
- 20x/busybox on
- runc

- 0.3s
- 7.1s
- 7.9s
Boot time overhead

Time to boot containers

- ix/busybox: 0.3s
- IX/busybox: 7.1s
- Kata: 7.9s
- 2x/busybox: 14.3s
- 20x/busybox: 14.3s

Confidential Containers - Distrusting your VM host
Boot time overhead

Time to boot containers

- Pod boot time is relatively constant (about 7s here)
- But can you run multiple containers in a single pod?
Run time overhead

Shutting down also takes time

20x busybox on runc

7.9s
Run time overhead

Shutting down also takes time

- 20x busybox on
- runc
- 20x busybox on Kata

15s
10s
5s

7.9s
14.3s
Run time overhead

Shutting down also takes time

- 20x busybox on run: 7.9s
- 20x busybox on Kata: 14.3s
- Run 20x busybox: 27.3s
Run time overhead

Shutting down also takes time

- 20x busybox on: 7.9s
- runc: 14.3s
- Kata: 27.3s
- Run 20x busybox: 28.4s
- Run 20x alpine: 28.4s
Run time overhead

Shutting down also takes time

- 20x busybox on run: 7.9s
- 20x busybox on Kata: 14.3s
- Run 20x busybox: 27.3s
- Run 20x alpine: 28.4s
- Run 20x fedora: 29.5s
Run time overhead

Shutting down also takes time

- Shutting down a Kata pod also takes time
- With Kata, the container base you use is not as relevant
Disk I/O

Disk performance with virtiofs: improved, but still not good
Disk read overhead

Copy files from the local file system

1m
30s
34.7s
5s
runc
Disk read overhead

Copy files from the local file system

- runc: 5s
- runc+blk: 4.9s
- Total: 34.7s
Disk read overhead

Copy files from the local file system

- `runc`: 5s
- `runc+blk`: 4.9s
- `Kata+virtiofs`: 9.7s
- `Kata+blk`: 20.3s
- `Kata+virtiofs`: 75.7s
Disk read overhead

Copy files from the local file system

- runc: 5s, 34.7s
- runc+blk: 4.9s, 20.3s
- Kata+virtiofs: 9.7s, 75.7s
- Kata+9pfs: 10.7s, 172.7s
Disk read overhead

Copy files from the local file system
Disk read overhead

Copy files from the local file system

- File reads are markedly slower with Kata
- virtiofs improves a lot over 9pfs
- Kata benefits less from blk than runc
Disk write overhead

Write a 1.0G file from /dev/random

20s
10s
15.1s
5.1s
runc

Page 45 Confidential Containers - Distrusting your VM host 9/9/2022 - 20:37:20
Disk write overhead

Write a 1.0G file from /dev/random
Disk write overhead

Write a 1.0G file from /dev/random

- runc: 15.1s
- Kata+virtiofs: 10.2s
- Kata+9p.fs: 5.2s
- Total: 18.3s
Disk write overhead

Write a 1.0G file from /dev/random

- runc: 15.1s
- Kata+virtiofs: 10.2s
- Kata+9pfs: 18.3s
- runc+blk: 5.2s
Disk write overhead

Write a 1.0G file from /dev/random

- runc: 15.1s
- Kata+virtiofs: 10.2s
- Kata+9p.fs: 5.9s
- runc+blk: 5.2s
- rewrite-runc+blk: 4.7s

Page 45.4  Confidential Containers - Distrusting your VM host  9/9/2022 - 20:37:27
Disk write overhead

Write a 1.0G file from /dev/random

- Disk write overhead:
  - runc: 15.1s
  - Kata+virtiofs: 10.2s
  - Kata+9pfs: 5.9s
  - runc+blk: 5.2s
  - rewrite runc+blk: 4.7s
  - Kata+blk: 5s

Page 45.5
Confidential Containers - Distrusting your VM host

9/9/2022 - 20:37:28
Disk write overhead

Write a 1.0G file from /dev/random

- runc: 5.1s
- Kata+virtfs: 5.9s
- Kata+9p.fs: 5.2s
- reWrite runc+blk: 4.7s
- Kata+blk: 5s
- reWrite Kata+blk: 5.1s
Disk write overhead

Write a 1.0G file from /dev/random

- Kata with virtiofs outperforms runc here
- Overwriting a file is more expensive
- The effect is less marked with Kata
Networking I/O

Multiple networking configurations
At least some of them should perform well
Network overhead

Read or write a 1G file from local server

1m

30s

run read

42.3s

8.0s
Network overhead

Read or write a 1G file from local server

- nuc read: 42.3s
- data read: 43.3s
Network overhead

Read or write a 1G file from local server

- runc read: 42.3s
- kdat aread: 43.3s
- runc write: 41.7s

Page 47.2 Confidential Containers - Distrusting your VM host
Network overhead

Read or write a 1G file from local server

- runc read: 8.8s
- kata read: 6.1s
- runc write: 2.9s
- kata write: 3.7s
Network overhead

Read or write a 1G file from local server

<table>
<thead>
<tr>
<th></th>
<th>runc read</th>
<th>kata read</th>
<th>runc write</th>
<th>kata write</th>
<th>runc read host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>8.0s</td>
<td>6.1s</td>
<td>2.9s</td>
<td>3.7s</td>
<td>27.5s</td>
</tr>
</tbody>
</table>

1m
30s
Network overhead

Read or write a 1G file from local server

- runc read
- kata read
- runc write
- kata write
- runc read host
- kata read host

Times:
- runc read: 42.3s
- kata read: 43.3s
- runc write: 41.7s
- kata write: 43.4s
- runc read host: 38.3s
- kata read host: 56.3s
Network overhead

Read or write a 1G file from local server

- runc read: 8.8s
- kata read: 6.1s
- runc write: 2.9s
- kata write: 3.7s
- runc read host: 38.3s
- kata read host: 56.3s
- runc write host: 293.3s
Network overhead

Read or write a 1G file from local server
Network overhead

Read or write a 1G file from local server

- Kata overhead is modest
  - Lower system usage in guest (moved to host)
- Host networking is rather bad for Kata
  - Writing is really bad
Network bandwidth

Path length increase make it harder to saturate link
Network bandwidth

Path length increase make it harder to saturate link
Network bandwidth

Path length increase make it harder to saturate link

- Hot-plugged vCPUs don't help much
- Increased path length, notably on the host