Kata Containers: the speed of containers, security of VMs - even in a nested environment!

Eric Ernst, Intel
K. Y Srinivasan, Microsoft
Shiny Sebastian, Intel
Agenda

• Overview of Kata
• Nested use case
• KVM on Hyper-V
• A look at Kata nested
Process A

Process B

Process C

Namespaces

LINUX KERNEL

CPU

MEMORY

NETWORK

STORAGE
VIRTUAL MACHINE CREATED with KVM/QEMU

Actively working on QEMU

No EMULATION

✓ New enlightened machine type
✓ Based on QEMU
✓ See talk C KVM-forum on Thursday

VIRTUAL MACHINE

PROCESS A

NAMESPACEs

LINUX KERNEL A

VIRTUAL MACHINE

PROCESS B

NAMESPACEs

LINUX KERNEL B

VIRTUAL MACHINE

PROCESS C

NAMESPACEs

LINUX KERNEL C

VIRTUAL MACHINE

HARDWARE VIRTUALIZATION

HARDWARE VIRTUALIZATION

HARDWARE VIRTUALIZATION

LINUX KERNEL
Kata-runtime
Kata-runtime

Container
Kata-agent
Kernel
hypervisor

vssock

virtual Machine

(oci)
Nested Use Case
Nested Kata use case

- How many?
  - What’s an appropriate pool size for tenant 1, or a particular workload?
  - Does each untrusted workload need its own VM?
- Infrastructure work:
  - Need to manage virtual machine creation, and tag each per workload/tenant.
  - May need to size virtual machines conservatively based on what a workload *could* need
  - SDN, SDS and fabric overheads with spinning up VMs
Nested Kata use case

- How many?
- What's an appropriate pool size for tenant 1, or a particular workload?
- Does each untrusted workload need its own VM?

Infrastructure work:
- Needs to manage virtual machine creation, and tag each per workload/tenant.
- May need to size virtual machines conservatively based on what a workload *could* need.
- SDN, SDS and fabric overheads with spinning up VMs.

Better utilization of resources:
- No need to size VM based on workload needs.
- No need to size node pool based on potential tenants potential need.
- Pool is shared at a finer granularity.
Nested Kata use case

- Better utilization of resources:
  - No need to size VM based on workload needs
  - No need to size node pool based on potential tenants potential need
  - Pool is shared at a finer granularity
KVM on Hyper-V
Enlightenment

Storage

Network

⇒ Fast Data Path (VM Bus)
Nested Kata
Nested Kata: CPU Measurements

Approximately 3% degradation seen when running with varying number of threads on prime number calculation workload.

<table>
<thead>
<tr>
<th>number of threads</th>
<th>container in L1</th>
<th>container in L2 (kata)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.969</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.968</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.966</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0.976</td>
</tr>
</tbody>
</table>
Nested Kata: Network I/O - setup
Nested Kata: Network I/O - setup
Nested Kata: Network I/O - setup
Nested Kata: Network I/O - setup
Nested Kata Network I/O

<table>
<thead>
<tr>
<th></th>
<th>10 Gbps</th>
<th>25 Gbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Container</td>
<td>7.30%</td>
<td>20.70%</td>
</tr>
<tr>
<td>L2 Container (Kata)</td>
<td>10.70%</td>
<td>27.70%</td>
</tr>
</tbody>
</table>
Nested Kata Network I/O

**Fixed bandwidth CPU Utilization: L1, L2**

<table>
<thead>
<tr>
<th></th>
<th>10 Gbps</th>
<th>25 Gbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Container</td>
<td>7.30%</td>
<td>20.70%</td>
</tr>
<tr>
<td>L2 Container (Kata)</td>
<td>10.70%</td>
<td>27.70%</td>
</tr>
</tbody>
</table>

*measured utilization in Lo*
Nested Kata: Storage I/O - setup

- FID CONTAINER
- BLK STORAGE STACK
- /dev/sda
- Azure DS16-v3 (L1)
- Azure PREMIUM SSD
- L2 virtual machine
### Nested Kata Storage I/O

- Nested is relatively expensive
- High amount of iowait observed in L1 during L2 random read testing

<table>
<thead>
<tr>
<th></th>
<th>Random Read</th>
<th>Random Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Container</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>L2 Container (Kata)</td>
<td>11.7</td>
<td>9.1</td>
</tr>
</tbody>
</table>

**Chart:**
- **CPU Utilization:** L1, L2
- **Bandwidth:** 100 MB/s (6400 IOPS)

**Graph Details:**
- Y-axis: CPU Utilization
- X-axis: Random Read, Random Write

**Table:**
- L1 Container: 0.6
- L2 Container (Kata): 11.7

**Graph Description:**
- The graph shows the CPU utilization for Random Read and Random Write operations.
- The y-axis represents the CPU utilization ranging from 0 to 14.
- The graph indicates higher CPU utilization for L2 Container (Kata) compared to L1 Container for both read and write operations.
Nested Kata Storage I/O

- Nested is relatively expensive
- High amount of iowait observed in L1 during L2 random read testing

<table>
<thead>
<tr>
<th></th>
<th>Random Read</th>
<th>Random Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Container</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>L2 Container (Kata)</td>
<td>11.7</td>
<td>9.1</td>
</tr>
</tbody>
</table>

100 MB/s (6400 IOPS) bandwidth CPU Utilization: L1, L2

CPU Utilization:
- L1: 1.8 cores
- L2: 1.5 cores

*measured utilization in L1
Reducing Kata’s footprint

- Minimal kernel
- Minimal rootfs
- Minimally configured QEMU
Reducing Kata’s footprint

- Minimal kernel
- Minimal rootfs
- Minimally configured QEMU
- DAX/NVDIMM
Reducing Kata’s footprint

- Minimal kernel
- Minimal rootfs
- Minimally configured QEMU
- DAX/NVDIMM
- De-duplicating memory
### Nested Kata container density

<table>
<thead>
<tr>
<th>Dockerhub workload</th>
<th>Memory footprint</th>
<th>Containers/GB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Kata</td>
<td>runc</td>
</tr>
<tr>
<td>busybox (small)</td>
<td>93.2 MB</td>
<td>682 KB</td>
</tr>
<tr>
<td>mysql (medium)</td>
<td>135.5 MB</td>
<td>160.8 MB</td>
</tr>
<tr>
<td>elasticsearch (large)</td>
<td>2.5 GB</td>
<td>2.2 GB</td>
</tr>
</tbody>
</table>

“it depends”
Summary, next steps
Next Steps

- Nesting:
  - Continued improvements for KVM on Hyper-V
  - Optimizations for L2:
    - Investigate more efficient L2 storage options
    - General efficiency improvements to minimize nested cost

- Kata:
  - Improvements on density as well as security
  - Released support for NEMU
Where can you get Kata?

• Dockerhub:
  • katadocker/kata-deploy

• Packages:
  • Clear linux, Snap
  • Built for CentOS, Fedora, SLES, RHEL, Ubuntu

• Running on public Cloud:
  • ACS-engine support in Azure
  • Anywhere bare-metal or nested virtualization is supported, including AWS, Azure, GCP, packet.net, Vexxhost