No More Turtles

Alternative to Nesting

Mengmei Ye (mye@ibm.com), Angelo Ruocco (ang@zurich.ibm.com), Daniele Buono (dbuono@us.ibm.com), James Bottomley (iejb@linux.ibm.com), Hubertus Franke (frankeh@us.ibm.com)
Industry problem

- There's an increasing need from Kubernetes customers to spawn VMs:
  - Kubevirt: VM-based workloads scheduled through Kubernetes
  - kata-containers et al: increase container boundaries through VMs
- This creates a significant problem in common cloud-based Kubernetes deployments, where worker nodes are themselves deployed in VM
Present options and drawbacks

1. VMs on Baremetal systems
   - Expensive
   - Very low customizability

2. Nested VMs
   - Confidentiality (TDX/SEV/PVM encryption unavailable)
   - Security (large codebase for nested = higher chance of bugs)
   - Low performance
   - Not allowed by most Cloud Providers
Solution – Flatten the hierarchy

- A (Primary) VM is able to ask the host to spawn a (Secondary) VM
- The Primary VM is able to access the Secondary VM
- The Primary VM has some basic control over the Secondary VM
Solution – Flatten the hierarchy

- Standard VM
  - Nested VM
    - Virtualization software
  - Virtualization software

- Primary VM
  - Secondary VM
    - Virtualization software
    - Virtualization software
Flatten the hierarchy – Challenges

• Security
  – The rest of the system should not be affected by secondary VMs. Resources must be carved out from the Primary VM
  – Primary VM only has access to some pre-defined actions. The control plane is kept in L0

• Isolation
  – Communication channel Host <-> Primary <-> Secondary has to be only accessible by the Primary VM
  – Secondary VMs must be invisible to VMs in other namespaces
Solution – Flatten the hierarchy

SecVM Image Disk

SecVM Template

Virtualization Software

SecVM Image

SecVM Daemon

L0

SecVM Template

SecVM Image

SecVM Daemon

L1

SecVM Image Disk

PrimaryVM

SecondaryVM

PrimaryVM

SecVM Image Disk

SecVM Image Disk
### Solution – Flatten the hierarchy

<table>
<thead>
<tr>
<th>Std VM + Nested</th>
<th>Primary + Secondary VMs</th>
<th>Baremetal + Std VMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cheap</td>
<td>Cheap (?)</td>
<td>Expensive</td>
</tr>
<tr>
<td>High flexibility</td>
<td>Medium flexibility</td>
<td>Low flexibility</td>
</tr>
<tr>
<td>No encryption</td>
<td>Encryption</td>
<td>Encryption</td>
</tr>
<tr>
<td>No true device passthrough</td>
<td>True device passthrough</td>
<td>True device passthrough</td>
</tr>
<tr>
<td>Slow(ish)</td>
<td>Fast</td>
<td>Fast</td>
</tr>
</tbody>
</table>
The Secondary VM Daemon

• Talks to the Primary VM via VSOCK
• Controls the Secondary VMs
  – Create
  – Modify
  – Destroy
  – Show/List
Enforcing the limits – Cgroup

• Primary VM, Secondary VMs and Daemon live inside a cgroup with memory and cpu limits
Enforcing the limits – Storage

- Primary VMs have an additional disk where they put the images for Secondary VMs.
- The disk gets unplugged from the primary VM, mounted on host, the images is copied, and disk shrunk to new size before re-plug.
Enforcing the limits – Network

• A virtual network is created for each Primary - Secondary VMs partition.
• All the requests go to the same physical interface accessible by the Primary VM
Proof of Concept – Why Libvirt

- Open Source
- Supports multiple hypervisors
- Uses cgroups via systemd integration, easy to add implementation for limits enforcement
- Easy way to create and add virtual networks
- Easy, standard way to attach-detach devices at runtime
Secondary VM – Libvirt

DEMO
Future work

• Define standard APIs Host <-> Primary VMs
• Clean up – upstream code
• Improve cgroup <-> libvirt synergy
• Improve PrimaryVM isolation in host
  – Improve cgroup cpuset
  – Enforce guarantees over shared resources
• Evaluate alternative storage solutions