Container Friendly Java 9
Ken Sipe
Distribute Application Engineer

Apache Mesos Contributor
Apache Committer Myriad, Open DCOS
Developer: Embedded, C++, Java, Groovy, Grails, C#, GoLang

@KenSipe
Agenda

- What is a Container
- The Heart of Container (CGROUPS / NameSpaces)
- Java Mem / CPU
- Java in a Container
Does Java fit in a container?
Containers
Container
Write Once Run Any Where
High level: it's a lightweight VM

- I can get a shell on it (through SSH or otherwise)
- It "feels" like a VM:
  - own process space
  - own network interface
  - can install packages
  - can run services
## HARDWARE VIRTUALIZATION

<table>
<thead>
<tr>
<th>Year</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>1966-1972</td>
<td>IBM CP/CMS</td>
</tr>
<tr>
<td>1989</td>
<td>Insignia SoftPC</td>
</tr>
<tr>
<td>1997</td>
<td>Connectix VirtualPC</td>
</tr>
<tr>
<td>1999</td>
<td>VMWare Workstation</td>
</tr>
<tr>
<td>2001</td>
<td>IBM AIX LPAR</td>
</tr>
<tr>
<td>2002</td>
<td>Xen</td>
</tr>
<tr>
<td>2006</td>
<td>Amazon EC2</td>
</tr>
<tr>
<td>2007</td>
<td>Sun Logical Domains</td>
</tr>
<tr>
<td>2007</td>
<td>Linux KVM</td>
</tr>
<tr>
<td>2007</td>
<td>InnoTek VirtualBox</td>
</tr>
<tr>
<td>2008</td>
<td>Microsoft Hyper-V</td>
</tr>
</tbody>
</table>
Virtualization Defined
For those more visually inclined...

Traditional Architecture

Virtual Architecture
It's not quite like a VM:
- uses the host kernel
- can't boot a different OS

It's just a bunch of processes visible on the host machine

(contrast with VMs which are opaque)
## PROCESS VIRTUALIZATION

<table>
<thead>
<tr>
<th>Year</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>1979-1982</td>
<td>UNIX Chroot</td>
</tr>
<tr>
<td>1998</td>
<td>FreeBSD Jail</td>
</tr>
<tr>
<td>2001</td>
<td>Parallels Virtuozzo</td>
</tr>
<tr>
<td>2001</td>
<td>Linux-VServer</td>
</tr>
<tr>
<td>2005</td>
<td>Solaris Zones</td>
</tr>
<tr>
<td>2005</td>
<td>OpenVZ</td>
</tr>
<tr>
<td>2008</td>
<td>Linux LXC</td>
</tr>
<tr>
<td>2007+</td>
<td><strong>PAAS:</strong></td>
</tr>
<tr>
<td></td>
<td>Heroku, Joyent, CloudFoundry</td>
</tr>
<tr>
<td>2013</td>
<td>Docker</td>
</tr>
</tbody>
</table>
What’s the difference between containers and virtual machines?
Containers vs Virtual Machines

Containers are isolated, but share OS and, where appropriate, bins/libraries.
Differences between containers and virtual machines

- Weaker isolation in containers
- Containers run near-native speed CPU/IO
- Containers launch in around 0.1 second (libcontainer)
- Less memory overhead
Namespaces provide isolation:
- pid (processes)
- net (network interfaces, routing...)
- ipc (System V IPC)
- mnt (mount points, filesystems)
- uts (hostname)
- user (UIDs)

Control groups control resources:
- cpu (CPU shares)
- cpuacct
- cpuset (limit processes to a CPU)
- memory (swap, dirty pages)
- blkio (throttle reads/writes)
- devices
- net_cls, net_prio: control packet class and priority
- freezer
Control Groups
Control groups

- Resource metering and limiting
  - memory
  - CPU
  - block I/O
  - network*
  - Device node (/dev/*) access control
- freezer
Control groups - Generalities

- `/sys/fs/cgroup`
- Each subsystem (memory, CPU...) has a hierarchy (tree)
- Each process belongs to exactly 1 node in each hierarchy
- Each hierarchy starts with 1 node (the root)
- Each node = group of processes (sharing the same resources)
View on DCOS on CoreOS

```
root@261ce7bb2b33:/# ls -l /sys/fs/cgroup/
total 0
  dr-xr-xr-x 5 root root 0 Jun 1 21:17 blkio
  lrwxrwxrwx 1 root root 11 May 31 15:38 cpu → cpu,cpuacct
  dr-xr-xr-x 6 root root 0 Jun 1 21:17 cpu,cpuacct
  lrwxrwxrwx 1 root root 11 May 31 15:38 cpuacct → cpu,cpuacct
  dr-xr-xr-x 3 root root 0 Jun 1 21:17 cpuset
  dr-xr-xr-x 5 root root 0 Jun 1 21:17 devices
  dr-xr-xr-x 4 root root 0 Jun 1 21:17 freezer
  dr-xr-xr-x 6 root root 0 May 31 20:37 memory
  lrwxrwxrwx 1 root root 16 May 31 15:38 net_cls → net_cls,net_prio
  dr-xr-xr-x 2 root root 0 Jun 1 21:17 net_cls,net_prio
  lrwxrwxrwx 1 root root 16 May 31 15:38 net_prio → net_cls,net_prio
  dr-xr-xr-x 2 root root 0 Jun 1 21:17 perf_event
  dr-xr-xr-x 5 root root 0 May 31 21:16 systemd
root@261ce7bb2b33:/#
```
Memory cgroup: accounting

- Metrics: swap, total rss, # pages in/out
- Keeps track of pages used by each group:
  - file (read/write/mmap from block devices)
  - anonymous (stack, heap, anonymous mmap)
  - active (recently accessed)
  - inactive (candidate for eviction)
• Each group can have **hard** and **soft** limits
• Soft limits are not enforced
• Hard limits will trigger a per-group OOM killer
• Limits can be set for physical, kernel, total memory

```
docker run -it --rm -m 128m fedora bash
```
journalctl -f _TRANSPORT=kernel

Cpu cgroup

- Metrics: cpuacct.stats user | system
- Allows to set weights
- Limitations based on type
- Options:
  - CPU Shares
  - CPU Sets
CPU Shares

- Priority Weighting across all the cores
- default value is 1024

```bash
docker run -it --rm -c 512 stress ...
```
CPU Shares

- `sudo cgcreate -g cpu:A`
- `sudo cgcreate -g cpu:B`
- `cgroup A: sudo cgset -r cpu.shares=768 A 75%`
- `cgroup B: sudo cgset -r cpu.shares=256 B 25%`
CPU Sets

- **Pin** groups to **specific CPU(s)**
- **Reserve CPUs** for specific apps
- Avoid processes bouncing between CPUs
- Also relevant for NUMA systems

```
docker run -it -cpuset=0,4,6 stress
```
Blkio cgroup

- Limit: Bandwidth limits R/W, async / sync
- Limit: Weighted 10 10 1000
- Metrics: iops served, waiting and queued
Devices cgroup

- Controls what the group can do on device nodes
- Permissions include read/write/mknod
- Typical use:
  - allow /dev/{ty,zero,random,nul}...
  - deny everything else
- A few interesting nodes:
  - /dev/net/tun(network interface manipulation)
  - /dev/fuse(filesystems in user space)
  - /dev/kvm(VMs in containers, yay inception!)
  - /dev/dri(GPU)
When a process is created, it is placed in the same groups as its parent.
Groups are materialized by one (or multiple) pseudo-fs (typically mounted in /sys/fs/cgroup).
core@ip-10-0-2-26 ~ $ systemd-cgls memory
memory:
├─ /usr/lib/systemd/systemd --switched-root --system --deserialize 12
│   └─ mesos
│       └─ slave
│           └─ 1 /opt/mesosphere/packages/mesos--1eb02db6541da1a30af9fcbd9dc8e492351b8716/bin/mesos-slave
│           └─ 9640 mesos-logrotate-logger --help=false --log_filename=/var/lib/mesos/slave/slaves/77f7bbfb-5c51-42a1-8f90-28740c68450d-0000/executors/sleep.3df1f8e0-276f-11e6-b
│           └─ frameworks/77f7bbfb-5c51-42a1-8f90-28740c68450d-0000/executors/sleep.3df1f8e0-276f-11e6-b
│           └─ 9641 mesos-logrotate-logger --help=false --log_filename=/var/lib/mesos/slave/slaves/77f7bbfb-5c51-42a1-8f90-28740c68450d-0000/executors/sleep.3df1f8e0-276f-11e6-b
│           └─ frameworks/77f7bbfb-5c51-42a1-8f90-28740c68450d-0000/executors/sleep.3df1f8e0-276f-11e6-b
│           └─ 10625 mesos-logrotate-logger --help=false --log_filename=/var/lib/mesos/slave/slaves/77f7bbfb-5c51-42a1-8f90-28740c68450d-0000/executors/docker-sleep.2d2504c1-2770
│           └─ frameworks/77f7bbfb-5c51-42a1-8f90-28740c68450d-0000/executors/docker-sleep.2d2504c1-2770
│           └─ 10626 mesos-logrotate-logger --help=false --log_filename=/var/lib/mesos/slave/slaves/77f7bbfb-5c51-42a1-8f90-28740c68450d-0000/executors/docker-sleep.2d2504c1-2770
│           └─ frameworks/77f7bbfb-5c51-42a1-8f90-28740c68450d-0000/executors/docker-sleep.2d2504c1-2770
│           └─ 10627 mesos-docker-executor --container=mesos-77f7bbfb-5c51-42a1-8f90-28740c68450d-S0.a0c1787b-3cd4-4b4a-aa47-87b01036130f --docker_socket=/var/run/docker.sock --help=false --launcher_dir=/opt/mesos
│           └─ frameworks/77f7bbfb-5c51-42a1-8f90-28740c68450d-S0.a0c1787b-3cd4-4b4a-aa47-87b01036130f --docker_socket=/var/run/docker.sock --help=false --launcher_dir=/opt/mesos
│           └─ 10659 docker --H unix://var/run/docker.sock run --cpu-shares 1024 --memory 134217728 --e MARATHON_APP_VERSION=2016-05-31T20:42:22.763Z --e HOST=10.0.2.26 --e MARATHON_APP_RESOURCE_CPUS=1.0 --e MARATHON_APP_DOCKER_IMAGE
│           └─ frameworks/77f7bbfb-5c51-42a1-8f90-28740c68450d-S0.a0c1787b-3cd4-4b4a-aa47-87b01036130f --docker_socket=/var/run/docker.sock --help=false --launcher_dir=/opt/mesos
│           └─ 14772 mesos-logrotate-logger --help=false --log_filename=/var/lib/mesos/slave/slaves/77f7bbfb-5c51-42a1-8f90-28740c68450d-0000/executors/hdfs.6a7079c3-2841-11e6-b4
│           └─ frameworks/77f7bbfb-5c51-42a1-8f90-28740c68450d-0000/executors/hdfs.6a7079c3-2841-11e6-b4
│           └─ 14773 mesos-logrotate-logger --help=false --log_filename=/var/lib/mesos/slave/slaves/77f7bbfb-5c51-42a1-8f90-28740c68450d-0000/executors/hdfs.6a7079c3-2841-11e6-b4
│           └─ frameworks/77f7bbfb-5c51-42a1-8f90-28740c68450d-0000/executors/hdfs.6a7079c3-2841-11e6-b4
│           └─ 17369 mesos-logrotate-logger --help=false --log_filename=/var/lib/mesos/slave/slaves/77f7bbfb-5c51-42a1-8f90-28740c68450d-0001/executors/task.journalnode.journalnode
│           └─ frameworks/77f7bbfb-5c51-42a1-8f90-28740c68450d-0001/executors/task.journalnode.journalnode
│           └─ 17370 mesos-logrotate-logger --help=false --log_filename=/var/lib/mesos/slave/slaves/77f7bbfb-5c51-42a1-8f90-28740c68450d-0001/executors/task.journalnode.journalnode
│           └─ frameworks/77f7bbfb-5c51-42a1-8f90-28740c68450d-0001/executors/task.journalnode.journalnode
│           └─ 5c4cde35-f267-4a67-9b31-72b6fa77dec0
│           └─ 9642 /opt/mesosphere/packages/mesos--1eb02db6541da1a30af9fcbd9dc8e492351b8716/libexec/mesos/mesos-executor
│           └─ frameworks/77f7bbfb-5c51-42a1-8f90-28740c68450d-0001/executors/mesos/mesos-executor
│           └─ 9674 sh -c while ;; do echo "sandbox: $PWD"; sleep 120; done
core@ip-10-0-2-26 ~ $ cat /proc/9674/cgroup
9:freezer:/mesos/5c4cde35-f267-4a67-9b31-72b6fa77dec0
8:devices:/system.slice/dcos-mesos-slave.service
7:blkio:/system.slice/dcos-mesos-slave.service
6:net_cls,net_prio:
5:cpu,cpuaacct:/mesos/5c4cde35-f267-4a67-9b31-72b6fa77dec0
4:cpuset:
3:perf_event:
2:memory:/mesos/5c4cde35-f267-4a67-9b31-72b6fa77dec0
1:name=systemd:/mesos_executors.slice
### systemd-cgtop

<table>
<thead>
<tr>
<th>Path</th>
<th>Tasks</th>
<th>%CPU</th>
<th>Memory</th>
<th>Input/s</th>
<th>Output/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>71</td>
<td>3.1</td>
<td>13.9G</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/mesos</td>
<td>-</td>
<td>1.7</td>
<td>13.4G</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice</td>
<td>2</td>
<td>1.4</td>
<td>439.4M</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/system-spread.slice</td>
<td>4</td>
<td>0.6</td>
<td>180.0M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/dos-minuteman.service</td>
<td>11</td>
<td>0.5</td>
<td>12.8G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/dos-minuteman.service</td>
<td>4</td>
<td>0.3</td>
<td>132.2M</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/dos-minuteman.service</td>
<td>5</td>
<td>0.3</td>
<td>485.5M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/dos-minuteman.service</td>
<td>3</td>
<td>0.2</td>
<td>2.9M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/dos-minuteman.service</td>
<td>5</td>
<td>0.1</td>
<td>71.0M</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/docker.service</td>
<td>1</td>
<td>0.0</td>
<td>31.1M</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/docker.service</td>
<td>1</td>
<td>0.0</td>
<td>132.0K</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/mesos_executors.slice</td>
<td>22</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/dbus.service</td>
<td>1</td>
<td>-</td>
<td>1.6M</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/dos-minuteman.service</td>
<td>1</td>
<td>-</td>
<td>7.2M</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/dos-minuteman.service</td>
<td>1</td>
<td>-</td>
<td>7.2M</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/docker-4a27e60e8ac0f659d6125a700b052a5601885247fe14772c673d8c57a9a242bf.scope</td>
<td>23</td>
<td>-</td>
<td>1.8M</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/docker-9a27e60e8ac0f659d6125a700b052a5601885247fe14772c673d8c57a9a242bf.scope</td>
<td>1</td>
<td>-</td>
<td>616.0K</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/docker-9a27e60e8ac0f659d6125a700b052a5601885247fe14772c673d8c57a9a242bf.scope</td>
<td>2</td>
<td>-</td>
<td>324.0K</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/policy.service</td>
<td>1</td>
<td>-</td>
<td>13.3M</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/system-getty.slice/getty@tty1.service</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/system-getty.slice/serial-getty/tty5@.service</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/system-sshd.slice/ssh@0-10.0.2.26:22-10.0.4.64:37914.service</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/system-x2dcloudinit.slice</td>
<td>-</td>
<td>-</td>
<td>28.0K</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/system-x2dcloudinit.slice</td>
<td>1</td>
<td>-</td>
<td>50.6M</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/system-x2dcloudinit.slice</td>
<td>1</td>
<td>-</td>
<td>812.0K</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/system-x2dcloudinit.slice</td>
<td>1</td>
<td>-</td>
<td>1.0M</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/system-x2dcloudinit.slice</td>
<td>1</td>
<td>-</td>
<td>536.0K</td>
<td>0B</td>
<td>0B</td>
</tr>
<tr>
<td>/system.slice/system-x2dcloudinit.slice</td>
<td>1</td>
<td>-</td>
<td>2.5M</td>
<td>0B</td>
<td>0B</td>
</tr>
</tbody>
</table>

`core@ip-10-0-2-26 ~ $`
CGroup + Docker

- You can find all the information about the CPU for a specific container under /sys/fs/cgroup/cpu/system.slice/docker-$FULL_CONTAINER_ID.scope/, for example:

  ```
  $ ls /sys/fs/cgroup/cpu/system.slice/
docker-6935854d444d78abe52d629cb9d680334751a0cda82e11d2610e041d77a62b3f.scope/
cgroup.clone_children cpuacct.usage_percpu cpu.rt_runtime_us tasks
cgroup.procs cpu.cfs_period_us cpu.shares
cpuacct.stat cpu.cfs_quota_us cpu.stat
cpuacct.usage cpu.rt_period_us notify_on_release
  ```
Namespaces
Namespaces

- Provide processes with their own view of the system
- Cgroups = limits how much you can use; namespaces = limits what you can see (and therefore use)
- Multiple namespaces:
  - pid, net, mnt, uts, ipc, user
- Each process is in one namespace of each type
Processes within a PID namespace only see processes in the same PID namespace.

Each PID namespace has its own numbering (starting at 1).

When PID 1 goes away, the whole namespace is killed.
Mnt namespace

- **Mounts**
  - read-only
  - shared - share subsequent mounts in a shared mount
  - slave - only source can share with target but target mounts are private
  - private - non-sharing of subsequent mounts in a shared mount
Typical Container Uses of MNT

- Mount RO /usr
- Private /tmp
Processes within a given network namespace get their own private network stack, including:

- network interfaces (including lo)
- routing tables
- iptables rules
- sockets (s, netstat)
Typical Container Uses of NET

- Root namespace
- Bridging
- Private namespace with socket activation
uts namespace

- gethostname
- sethostname
IPC namespace

- Allows a process (or group of processes) to have own:
  - IPC semaphores
  - IPC message queues
  - IPC shared memory
User namespace

- Allows to map UID/GID; e.g.:
  - UID 0 → 1999 in container C1 is mapped to UID 10000 → 11999 on host
  - UID 0 → 1999 in container C2 is mapped to UID 12000 → 13999 on host

- Avoids extra configuration in containers
  - UID 0 (root) can be squashed to a non-privileged user
Namespaces are created with the clone() system call (i.e. with extra flags when creating a new process)

Namespaces are materialized by pseudo-files in

/proc/<pid>/ns

When the last process of a namespace exits, it is destroyed (but can be preserved by bind-mounting the pseudo-file)
cgroups and namespaces

- Can be used independently
- Use a few cgroups if you just need resource isolation
- Simulate a network of routers with network namespaces
- Put the debugger in a container's namespaces, but not its cgroups (to not use its resource quotas)
- Setup a network interface in an isolated environment, then move it to another
Containers
You are **always** in a container

- Even when you don't run containers ...

  - **you are in a container**
LXC

- Set of user-land tools
- A container is a directory in /var/lib/lxc
- Small config file + root filesystem
- Early versions had no support to move images around
- Requires significant amount of elbow grease (easy for sysadmins/ops, hard for devs)
Docker Engine

- Daemon controlled by REST(ish) API
- First versions shelled out to LXC,
- now uses its own lib container runtime
- Manages containers, images, builds, and more
• rkt, runC
  • Focus on container execution (no API, no image management, no build, etc.)
  • They implement different specifications: rkt implements appc (App Container)
  • runC implements OCP (Open Container Project), leverages Docker's libcontainer
• Mesos Container
Container Future

- containerd
Which container is right for you?

- They all use the same kernel features
- Performance will be exactly the same
Let's Talk Java
• Java Language + Java Specification + **Java Runtime**
Java Runtime Considerations

- Memory
- JIT Optimizations
- Mode: client vs. server
- Thread Management
- GC
Java Memory Impact on a Machine

- Native JRE
- Heap
- Perm / meta
- JIT bytecode
- JNI
- NIO
- Threads
Java Memory

Eden  SS0  SS1  Old Space

-Xms / -Xmx
Java Memory

-XX:MaxPermSize
# Java Memory

<table>
<thead>
<tr>
<th>Eden</th>
<th>SS0</th>
<th>SS1</th>
<th>Old Space</th>
<th>Perm Space</th>
</tr>
</thead>
</table>

- `-XX:SurvivorRatio`
- `-XX:MaxNewSize`
- `-XX:NewRatio`
Metaspace
Congratulations!! Leaky classloaders now effect the entire machine (or the container)
NIO with Native Memory Buffer
Threads

- Every thread in an application requires memory to store its stack
- Default sizes usually: 512K to 1024K / thread
client vs server mode

- JIT differences
- larger old spaces
- code cache (JRE 7)
  - client = 32M
  - server = 48M
  - JRE8 = 96M
Sets the default # threads for GC
Initial value of the ForkJoin Pool

-Djava.util.concurrent.ForkJoinPool.common.parallelism=1
-XX:ParallelGCThreads
-XX:ConcGCThreads
JRE initializations based on core count

- JIT compiler threads
- HotSpot thresholds and optimizations
- Number of thread in the common fork-join pool
- and more…
Time To Safe Point (TTSP)

- JIT Optimization
- Another Pause Opportunity

- Affected when the number of threads outnumbers the number of cores
Bringing it together
Where Java Gets it’s Information

- Memory limitations are naive and most be set
Where Java Gets it’s CPU Information

- JDK 7/8 - resources from sysconf
  ```c
  sysconf(_SC_NPROCESSORS_ONLN);
  ```
- JDK 9 - sched_getaffinity
  - accounts for cpusets
Java with CPU Set

- CPUSSET
  - pin to specific CPUs

- `Runtime.getRuntime().availableProcessors();`  == # cores assigned

```bash
docker run -ti --cpuset=0,4,6 ...
```
Java with CPU Share

- CPU Share
  - Priority Weighting across all the cores
  - `Runtime.getRuntime().availableProcessors() == # cores on node`

```
docker run -ti -c 512 ...
```
Java and CPU Shares

- Land on a 32 core box
  - 32 cores are seen by the JRE
  - 32 threads set by default for ForkJoinPool
"We resisted making the JVM resource pool aware on Solaris and that turned out to be a win because Zones became dominant.

But now we face a similar problem with cgroups on Linux and environments like Docker.

We were lucky that cpusets are reflected in the output of sched_getaffinity - a nice simple change."
•“But memory constraints are far more problematic and may not even be queryable in general.

•If there are no API's to tell the VM the real resource story what is the VM supposed to do? I don't have any answers to that.”

•“When the environment lies to the VM about what is available it makes it very hard for the VM to try to adjust.”
“The good thing about docker containers (and some other like containers) is that they don’t hide the underlying hardware from processes like VM technology does.”

“The bad thing about docker containers (and some other like containers) is that they don’t hide the underlying hardware from processes like VM technology does.”

Kirk Pepperdine
Java 8/9

- Java 8u131
  - -XX:+UnlockExperimentalVMOptions
  - -XX:+UseCGroupMemoryLimitForHeap
  - -XX:+UseStringDeduplication

https://docs.oracle.com/javase/9/gctuning/garbage-first-garbage-collector.htm#JSGCT-GUID-D74F3CC7-CC9F-45B5-B03D-510AEF2DAC
References

- Java Memory
- Java Performance
- IBM: Thanks for the memory
THANK YOU!
GREAT INDIAN DEVELOPER SUMMIT 2019

Conference: April 23-26, Bangalore

Register early and get the best discounts!

www.developersummit.com  @greatindiandev  bit.ly/gidslinkedin