Debugging Java Apps in Containers: No Heavy Welding Gear Required

Daniel Bryant
@danielbryantuk

Steve Poole
@spoole167

Agenda

- Assuming Java and Docker basic knowledge
- You can still use standard Java tooling
- Docker is not as ‘contained’ as we think
- Containers do change some things (a lot)
- Case studies
- Monitoring and logging FTW
- Docker debug tool
- OS debug tools

Who Are We?

Steve Poole

IBM Developer

@spoole167

Making Java Real Since Version 0.9

Open Source Advocate

DevOps Practitioner (whatever that means!)

Driving Change

Daniel Bryant

Principal Consultant,
OpenCredo

@danielbryantuk

"Biz-dev-QA-ops"

Leading change in organisations

All over Docker, Mesos, k8s, Go, Java

InfoQ, DZone, Voxxed contributor

Do We Need the Welding Gear?

Part 1

Simple and even simpler…

Simple example (using spark java, maven and docker)

```
…
public class App {

        public static void main(String[] args) {

                get("/hello", (req, res) -> "Hello World");

        }
}
```

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
   <artifactId>exec-maven-plugin</artifactId>
   <version>1.4.0</version>
    <configuration>
          <mainClass>App</mainClass>
     </configuration>
</plugin>
```

```
<dependency>
    <groupId>com.sparkjava</groupId>
    <artifactId>spark-core</artifactId>
    <version>2.3</version>
 </dependency>
```

Dockerfiles

```
$ docker build -t dukeserver .
```

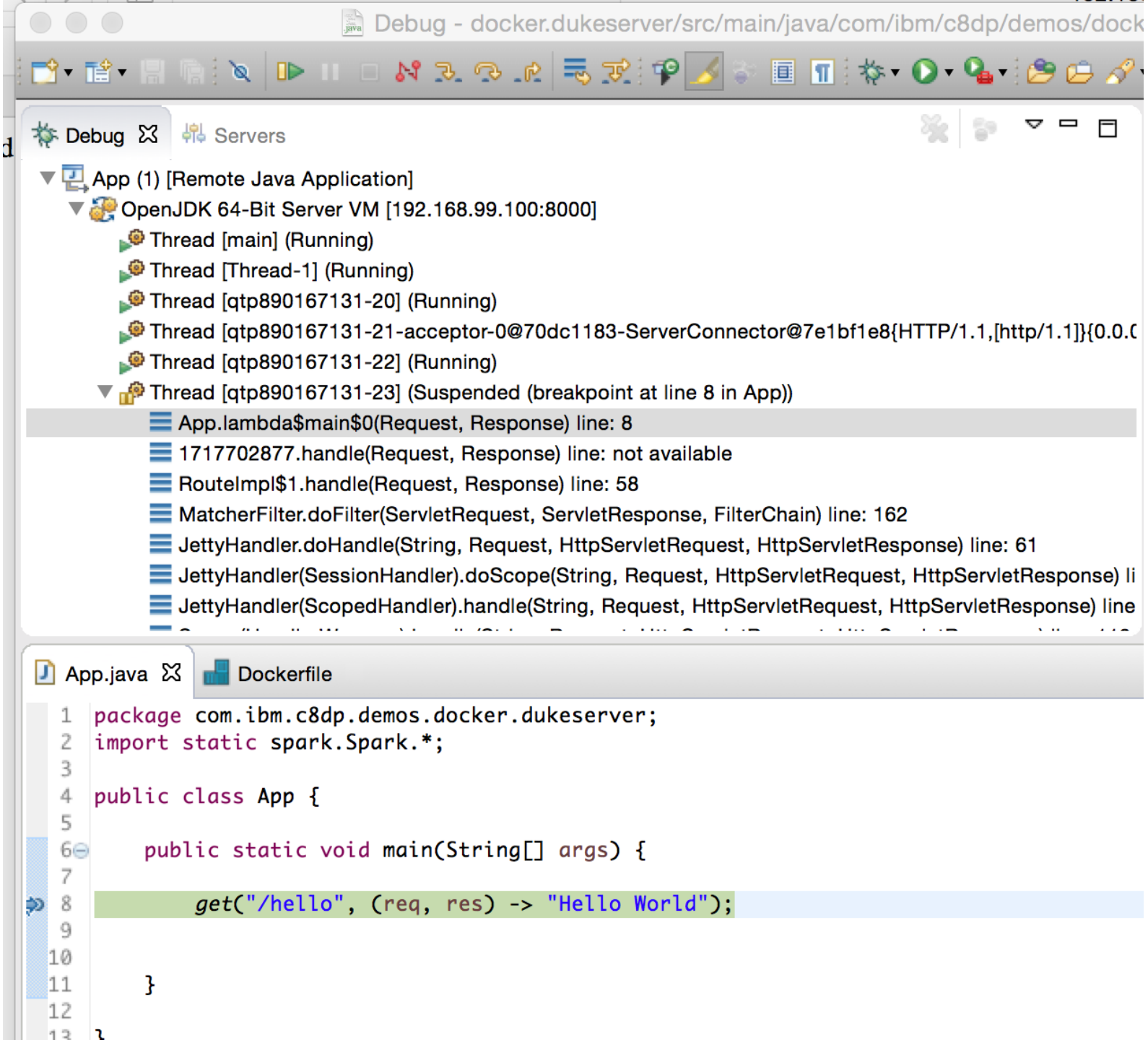
```
FROM maven:3.3.3-jdk-8
COPY . /usr/src/app
WORKDIR /usr/src/app
RUN mvn compile
CMD mvn exec:java
```

```
$ docker build -t dukeserver-debug -f Dockerfile-debug .
```

```
FROM dukeserver
CMD /usr/share/maven/bin/mvnDebug exec:java
```

Simple

- Treat Docker image as a remote server
  - Enable debugging port in Docker at launch
    - -p 8000:8000
  - Point debugger to image host
    - 192.168.99.100
- And you're done


Debug - docker.dukeserver/src/main/java/com/ibm/c8dp/demos/dock
Debug
Servers
App (1) [Remote Java Application]
OpenJDK 64-Bit Server VM [192.168.99.100:8000]
Thread [main] (Running)
Thread [Thread-1] (Running)
Thread [qtp890167131-20] (Running)
Thread [qtp890167131-21-acceptor-0@70dc1183-ServerConnector@7e1bf1e8{HTTP/1.1,[http/1.1]}{0.0.0
Thread [qtp890167131-22] (Running)
Thread [qtp890167131-23] (Suspended (breakpoint at line 8 in App))
App.lambda$main$0(Request, Response) line: 8
1717702877.handle(Request, Response) line: not available
RouteImpl$1.handle(Request, Response) line: 58
MatcherFilter.doFilter(ServletRequest, ServletResponse, FilterChain) line: 162
JettyHandler.doHandle(String, Request, HttpServletRequest, HttpServletResponse) line: 61
JettyHandler(SessionHandler).doScope(String, Request, HttpServletRequest, HttpServletResponse) li
JettyHandler(ScopedHandler).handle(String, Request, HttpServletRequest, HttpServletResponse) line
App.java
Dockerfile
1 package com.ibm.c8dp.demos.docker.dukeserver;
2 import static spark.Spark.*;
3
4 public class App {
5
6 public static void main(String[] args) {
7
8 get("/hello", (req, res) -> "Hello World");
9
10
11 }
12
13 }

Even simpler

docker ps

```
CONTAINER ID        IMAGE               COMMAND                  PORTS
ce98c30288c0        dukeserver-debug    "/bin/sh -c '/usr/sha"   0.0.0.0:8000->8000/tcp,
                                                                 0.0.0.0:8081->4567/tcp

95043b8003ac        dukeserver          "/bin/sh -c '/usr/sha"   0.0.0.0:8080->4567/tcp
```

docker exec –it <<container>> /bin/bash

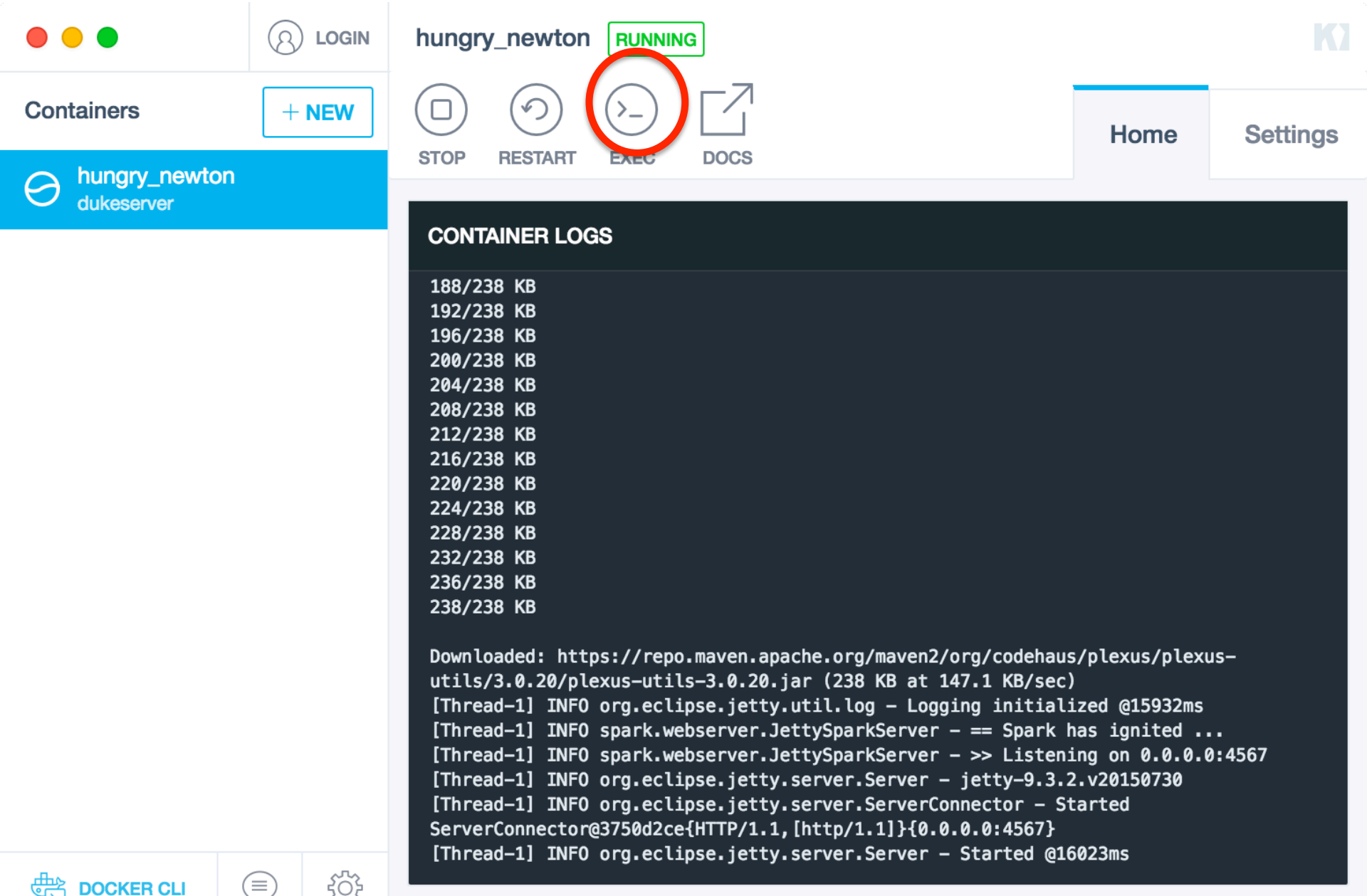
```
root@ce98c30288c0:/usr/src/app#

root@ce98c30288c0:/usr/src/app# ps -A
  PID TTY          TIME CMD
    1 ?        00:00:00 sh
    6 ?        00:00:00 mvnDebug
    8 ?        00:00:00 java
   40 ?        00:00:00 bash
   47 ?        00:00:00 ps
```

Even, even simpler

For Mac and Windows

Even, even simpler



Remote Debugging

- Working with JMX and Docker
  - Remote debugging
  - Jconsole / VisualVM

- Great Instructions
  - ptmccarthy.github.io/2014/07/24/remote-jmx-with-docker/

Gotchas

```
java -Dcom.sun.management.jmxremote \
  -Dcom.sun.management.jmxremote.port=9010 \
  -Dcom.sun.management.jmxremote.rmi.port=9010 \
  -Dcom.sun.management.jmxremote.local.only=false \
  -Dcom.sun.management.jmxremote.authenticate=false \
  -Dcom.sun.management.jmxremote.ssl=false \
  -jar MyApp.jar
```

- Ports must be mapped at ‘docker run’
  - docker run –d -p 8080:8080 -p 9010:9010

Java Debug Tooling

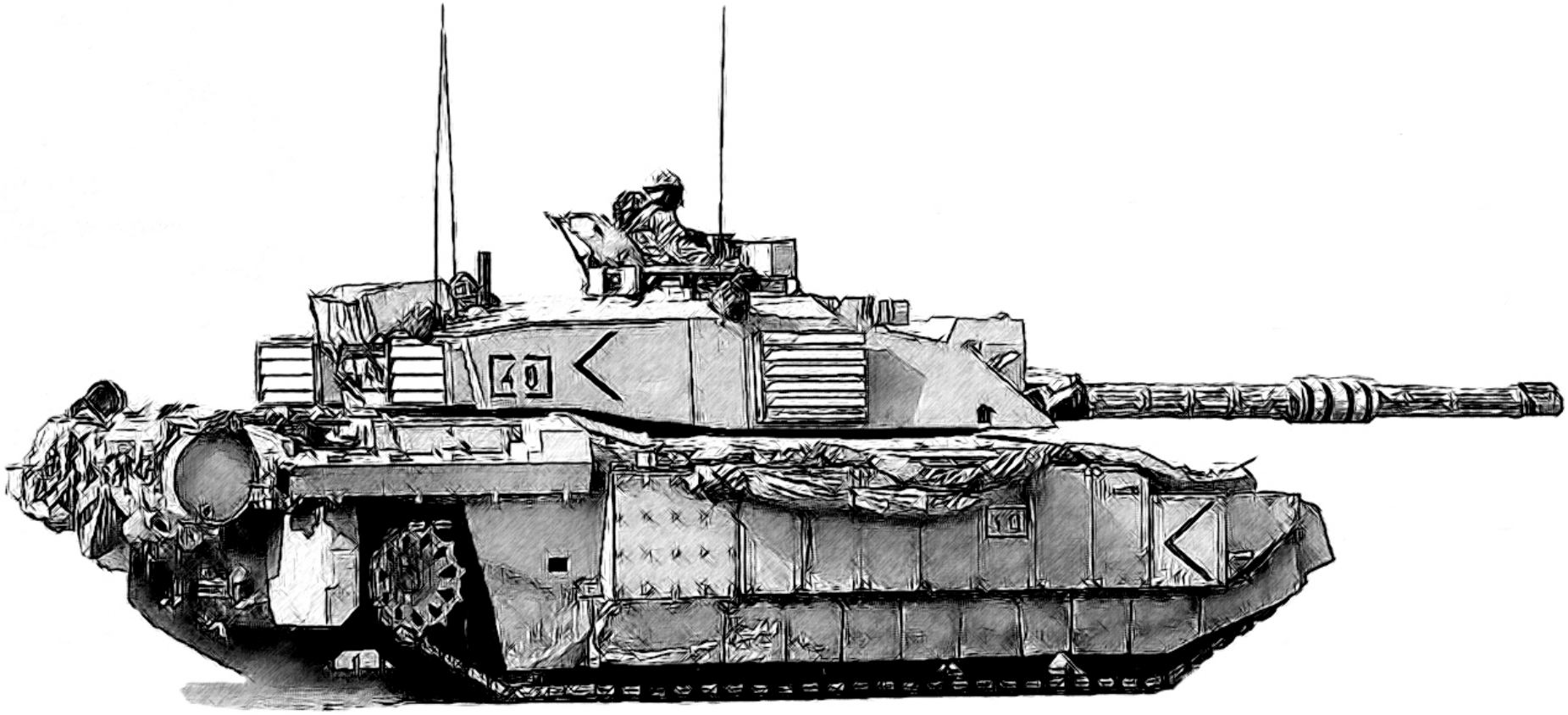
- docker exec –it <<container>> /bin/bash
- jps
  - Local VM id (lvmid)
- jstat
  - JVM Statistics
  - -class
  - -compiler
  - -gcutil
- jstack
  - Stack trace

Interlude: some simple truths about Docker

Is this your view of a container?

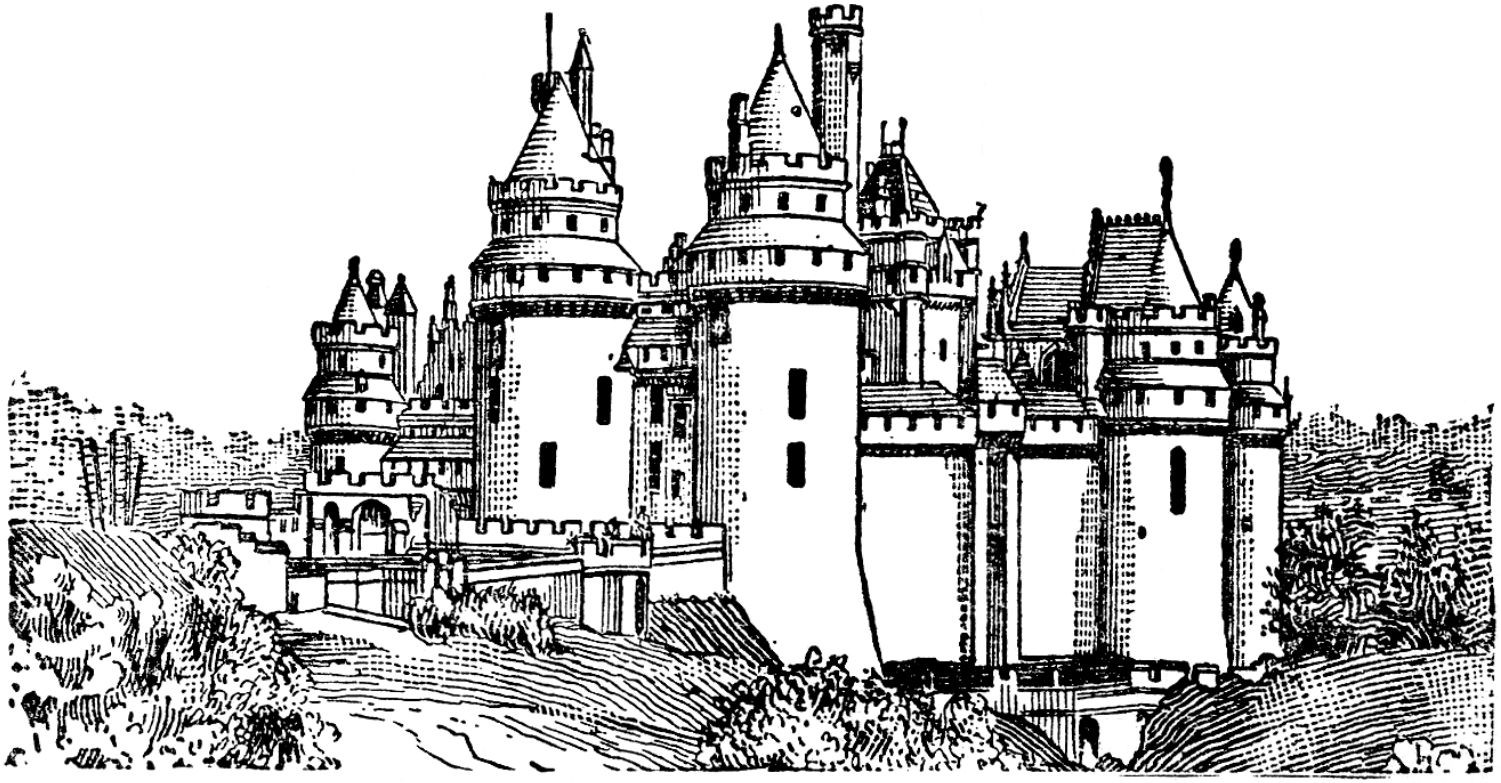
https://www.flickr.com/photos/45131642@N00/

Do you equate one with this?



https://www.flickr.com/photos/don-stewart/

Or this?



Sorry – it’s more like:

https://www.flickr.com/photos/smoovey/

The reality of containers

Welcome to the World of Containers

- Restrictions (typically)
  - Minimal OS
  - Limited resources
  - JRE vs JDK (with Java 9, minimal JDK?)
- JVM doesn't always respect resource limits
  - cgroup/namespace awareness
- Clustering/autoscaling/microscaling
  - Constant restarts and reallocation

…and the World of ‘Cloud’

- Everything is on the network
  - Seriously, everything…
- Noisy neighbours
  - Containers and VMs
- No writing to ‘local’ disk
  - Typically emphemeral

The Hardest Part of Debugging

- Finding the problem is difficult…
- A distributed system only makes this harder!
- Cloud and containers embrace transient
- …yep, we've got a challenge on our hands!

Case Studies

Case Study: The Perfect Crime

- Symptom
  - Container crashing spectularly trashing filesystem
- Diagnostics
  - Err…
- Problem
  - No logs to debug
- Resolution
  - Write logs to mounted directory
  - Ship logs via logstash

Case Study: The Slow Consumer

- Symptom
  - Suddenly realised we had loads of “consumers”
- Diagnostics
  - Examine container (docker stats)
  - Docker exec (top, vmstat, df –h)
  - docker exec with jstat
- Problem
  - GC issues (easy fix)
- Resolution
  - Ship metrics to InfluxDB (with Telegraf and Grafana)
  - Alerting and information radiators

Aggregation: Sick Cattle, Not Sick Pets

Case Study: App Crashing

- Symptom
  - Spring Boot app slow to respond/crashing
- Diagnostics
  - Examine host (top, vmstat, df –h)
  - Examine container (docker stats)
  - Docker exec (top, vmstat, df –h)
- Problem
  - No disk space for docker logging
- Resolution
  - Increase disk space (move logs to mount)

Case Study: Container Builds Failing

- Symptom
  - Jenkins builds sporadically failing – insufficient space
- Diagnostics
  - Examine host (df –h)
  - Inodes! (df –i)
- Problem
  - Old containers taking lots of inodes
- Resolution
  - Clear old containers
  - docker run --rm

Case Study: Application Dieing

- Symptom
  - Application boots, then crashes, container killed
- Diagnostics
  - Examine app logs
  - Examine host (top, vmstat, df –h)
  - Examine container (docker stats)
  - dmesg | egrep -i 'killed process'
- Problem
  - Only allowed Xmx memory limit (no overhead)
- Resolution
  - Set memory limit = Heap (Xmx) + Metaspace + JVM

Case Study: App Not Starting

- Symptom
  - Spring boot application not starting (or slow)
- Diagnostics
  - The usual suspects
  - jstack – blocked on SecureRandom
- Problem
  - /dev/random not so good on containers
- Resolution
  - -Djava.security.egd=file:/dev/urandom

```
$ jstack 2616

2014-09-04 07:33:30
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.20-b23 mixed mode):

"Attach Listener" #14 daemon prio=9 os_prio=0 tid=0x00007fb678001000 nid=0xaae waiting on condition
[0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"localhost-startStop-1" #13 daemon prio=5 os_prio=0 tid=0x00007fb65000d000 nid=0xa46 runnable
[0x00007fb662fec000]
   java.lang.Thread.State: RUNNABLE
    at java.io.FileInputStream.readBytes(Native Method)
    at java.io.FileInputStream.read(FileInputStream.java:246)
    at sun.security.provider.SeedGenerator$URLSeedGenerator.getSeedBytes(SeedGenerator.java:539)
    at sun.security.provider.SeedGenerator.generateSeed(SeedGenerator.java:144)
    at sun.security.provider.SecureRandom$SeederHolder.<clinit>(SecureRandom.java:192)
    at sun.security.provider.SecureRandom.engineNextBytes(SecureRandom.java:210)
    - locked <0x00000000f15173a0> (a sun.security.provider.SecureRandom)
    at java.security.SecureRandom.nextBytes(SecureRandom.java:457)
    - locked <0x00000000f15176c0> (a java.security.SecureRandom)
    at java.security.SecureRandom.next(SecureRandom.java:480)
    at java.util.Random.nextInt(Random.java:329)
    at org.apache.catalina.util.SessionIdGenerator.createSecureRandom(SessionIdGenerator.java:246)
```

Case Study: Rollouts Broke Comms

- Symptom
  - New app deployed with new IP added to ELB
  - Existing apps couldn't talk to it
- Diagnostics
  - dig service.qa.mystore.com (looks good)
  - Create simple Java service that echoes IPs
- Problem
  - Java's wonky caching of DNS
  - Pre Java 7 caches indefinitely
  - Post Java 7 'implementation specific' unless SM installed
- Resolution
  - -Dsun.net.inetaddr.ttl=0

Case Study: Slow Communications

- Symptom
  - Some apps would drip feed data onto the wire
  - Existing apps couldn't talk to it
- Diagnostics
  - Vmstat
  - Saw high cpu 'wa'
- Problem
  - Packed high contention containers on same host
- Resolution
  - Spread containers out

```
$vmstat

procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
......
 0  0 169936 545976 119436 2414452    0    0   980   272 2578  3342  0  1 73 26
 1  3 169936 544036 119436 2416012    0    0   696   890 2528  3778  1  3 72 24
 0  3 169936 544308 119436 2416012    0    0    68  2342 2833  2143  0  1 61 37
 0  3 169936 544160 119436 2416012    0    0    60  2806 2924  2276  0  1 65 34
 0  1 169936 544288 119436 2416272    0    0    24  3356 1816  2009  0  1 63 36
 0  1 169936 543692 119488 2416480    0    0   218   926 2031  2441  0  1 67 32
```

Our Learnings

Why Instrument?

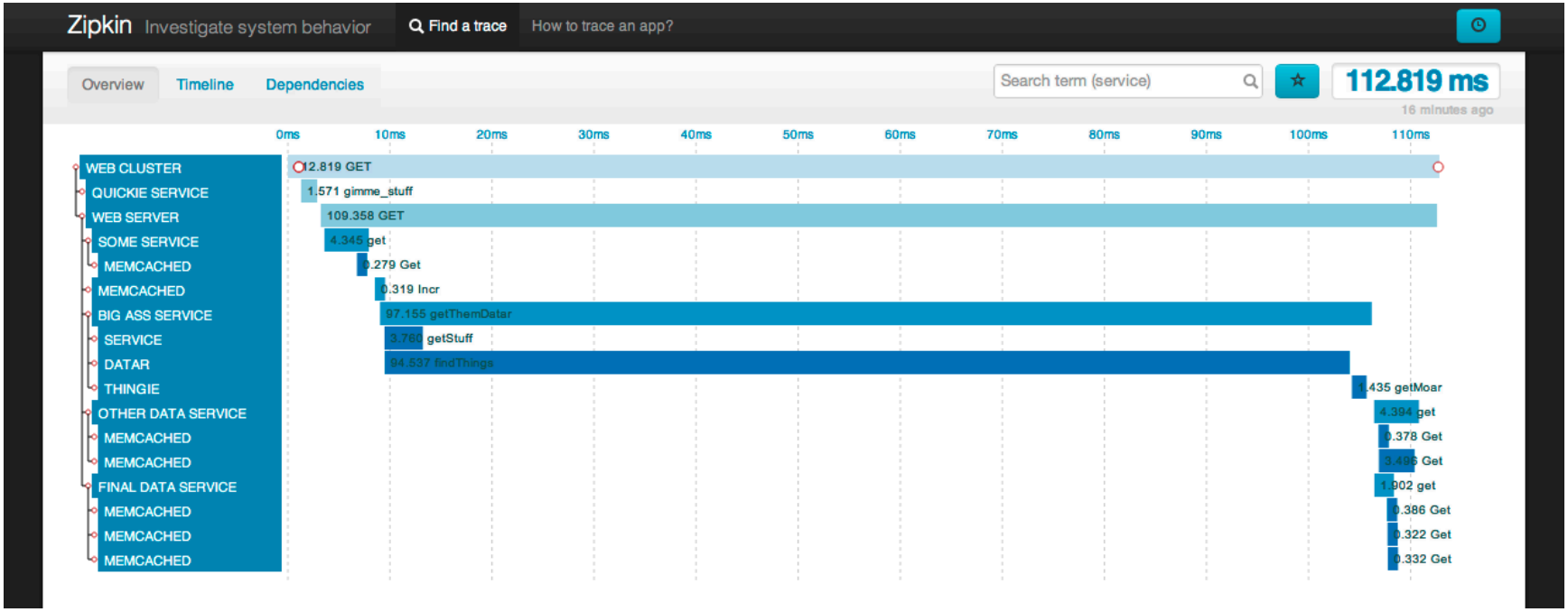
- Post Mortem
  - Containers are gone and so is all their state…
- Aggregation (or not)
  - coherent view of your application behaviour when it's distributed (microservices or not)
  - lots of containers running
- Looking for hints and smoking guns..

What to Instrument

Instrumentation

- Instrument the code/application
  - codehale-metrics, Spring Boot Actuator etc
- Instrument the OS
  - Collectd, munin, SAR
- Instrument the **system**
  - Zipkin, App Dynamics etc

The Value of System Monitoring

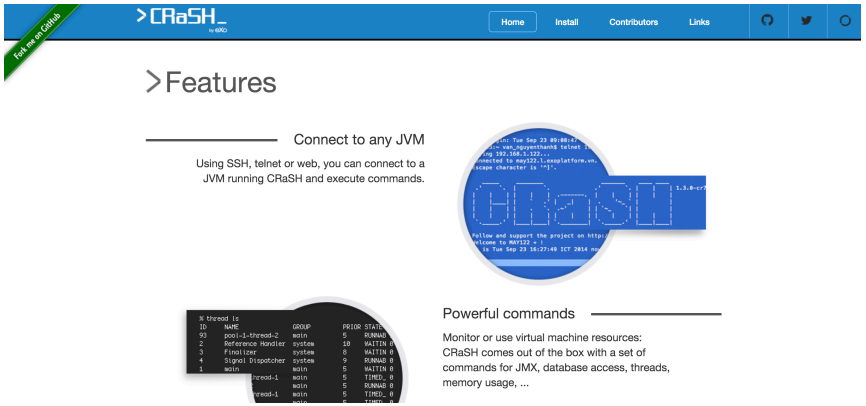


github.com/openzipkin/zipkin

github.com/openzipkin/docker-zipkin

Monitoring

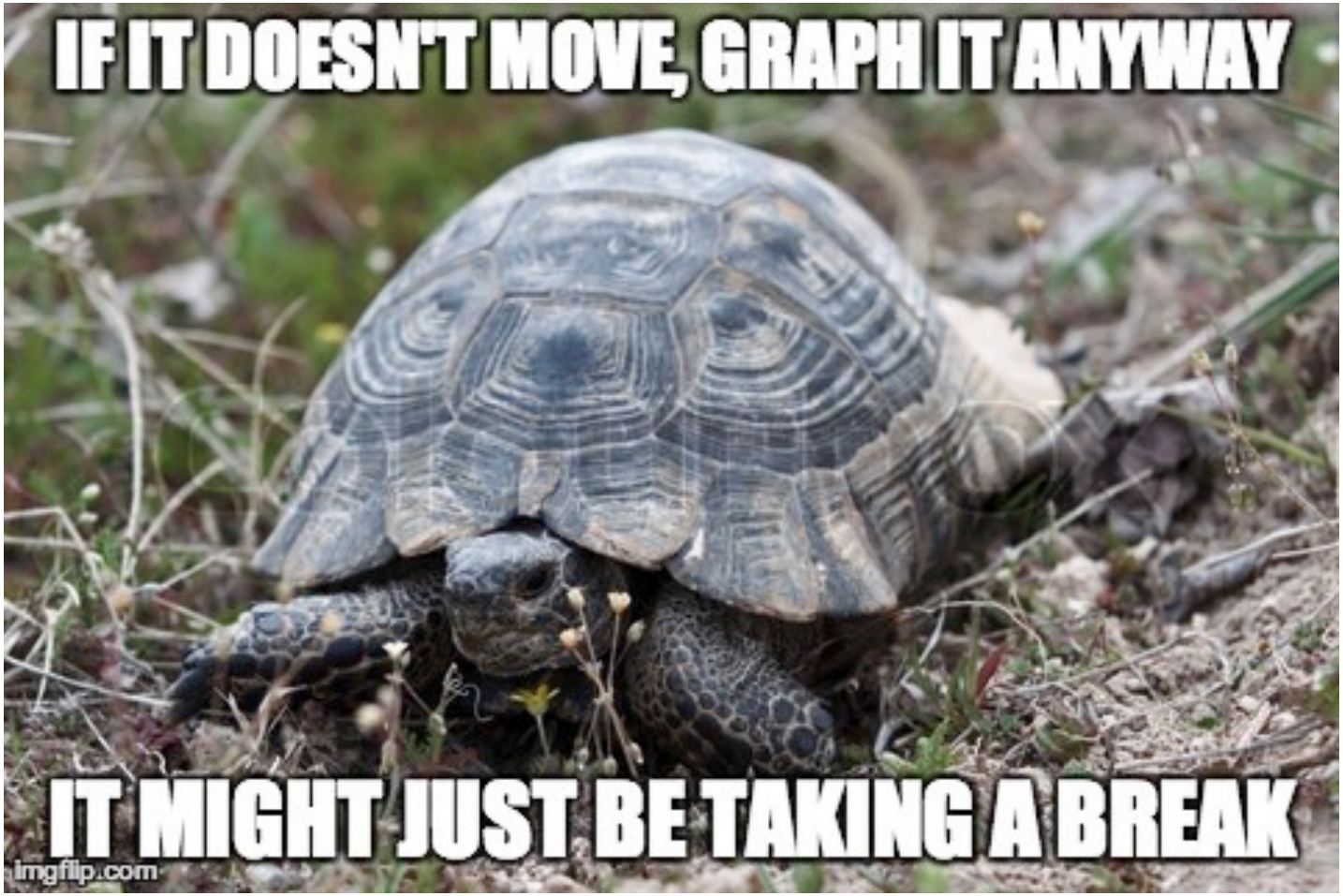
- In-situ monitoring
  - Curl docker stats endpoint
  - CRaSH shell



- Use monitoring tools
  - InfluxDB, Telegraf, Prometheus
  - Datadog, AWS CloudWatch, Grafana

Graphing

Graphing


IF IT DOESN'T MOVE, GRAPH IT ANYWAY
IT MIGHT JUST BE TAKING A BREAK
imgflip.com

Logging

- docker logs are your friend
- Rotate app log files to mounted dir
  - Otherwise they disappear!
- The ElasticSearch-Logstash-Kibana stack
  - github.com/deviantony/docker-elk
- Log like an operator

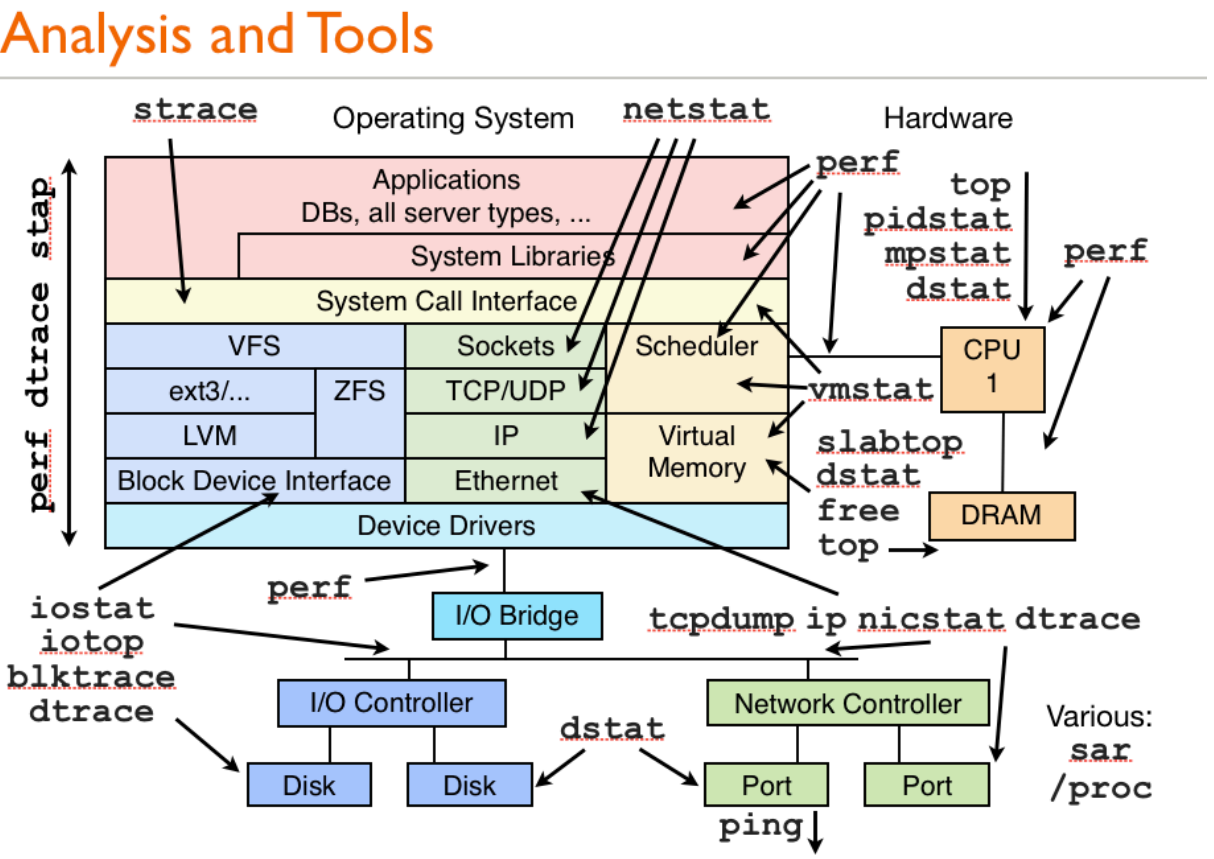
Quick summary of docker/OS debug tooling

Looking Inside the Container

Docker Debug Tools

- docker stats
  - Live stream of CPU and memory
- docker info
  - Displays system-wide information
- docker inspect
  - Detailed information on a container
  - $ docker inspect --format='{{.NetworkSettings.IPAddress}}' $INSTANCE_ID
  - $ docker inspect --format='{{.LogPath}}' $INSTANCE_ID
  - $ docker inspect --format='{{range $p, $conf := .NetworkSettings.Ports}} {{$p}} -> {{(index $conf 0).HostPort}} {{end}}' $INSTANCE_ID

OS Debugging Tools



www.joyent.com/blog/linux-performance-analysis-and-tools-brendan-gregg-s-talk-at-scale-11x

OS Debugging Tools

- Top, htop,
- ps,
- mpstat,
- free, df -h
- vmstat,
- iostat
- /proc filesystem
  - meminfo and vmstat not cgroup aware!

Networking Rulez

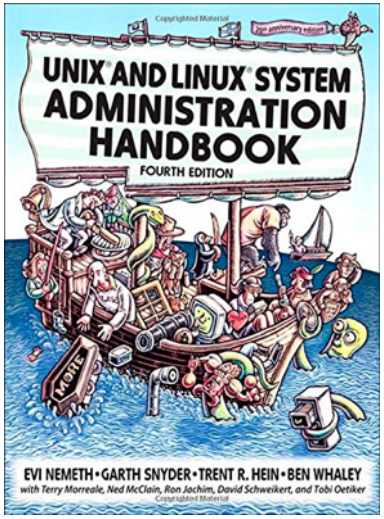
- tcpdump,
- netstat, ntop
- dig (not nslookup)
- ping, tracert
- Lsof –u <<username>>

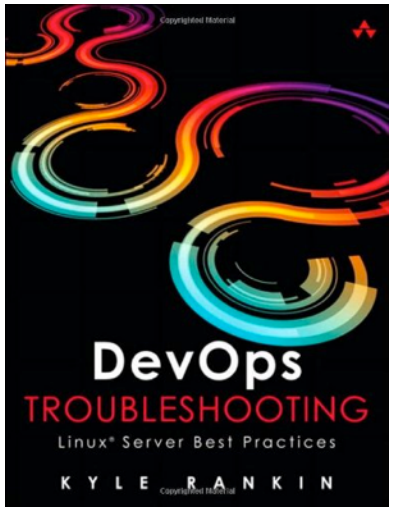
Let’s wrap this up…

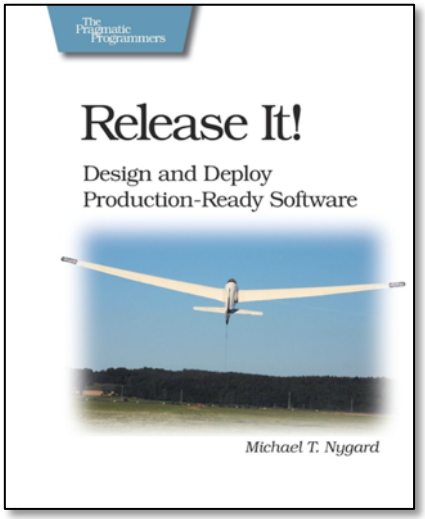
Summary

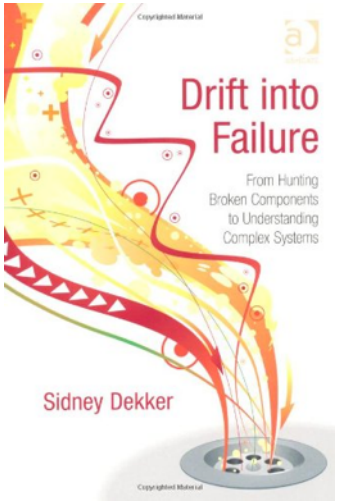
- Debugging is still an essential skill
  - rebooting containers doesn't cut it
- Isolating (targeting) the issue is vital
  - Distributing tracing (Zipkin etc)
- Build your debugging toolbox
  - Java + docker + OS debugging
- Monitoring and logging FTW

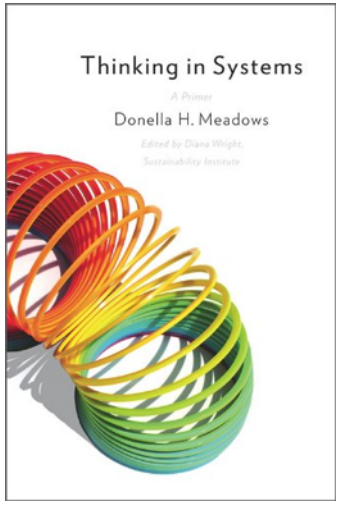
Books

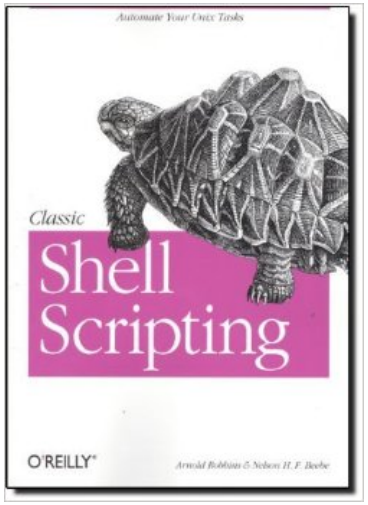

UNIX AND LINUX SYSTEM ADMINISTRATION HANDBOOK
FOURTH EDITION
EVI NEMETH • GARTH SNYDER • TRENT R. HEIN • BEN WHALEY
with Terry Morreale, Ned McClain, Ron Jachim, David Schweikert, and Tobi Oetiker


DevOps
TROUBLESHOOTING
Linux® Server Best Practices
KYLE RANKIN


The Pragmatic Programmers
Release It!
Design and Deploy
Production-Ready Software
Michael T. Nygard


Drift into Failure
From Hunting Broken Components to Understanding Complex Systems
Sidney Dekker


Thinking in Systems
A Primer
Donella H. Meadows


Classic
Shell Scripting
O'REILLY

Thanks! Questions?

@danielbryantuk - @spoole167

Disclaimer: We've made best efforts with this advice, but this is a rapidly developing space