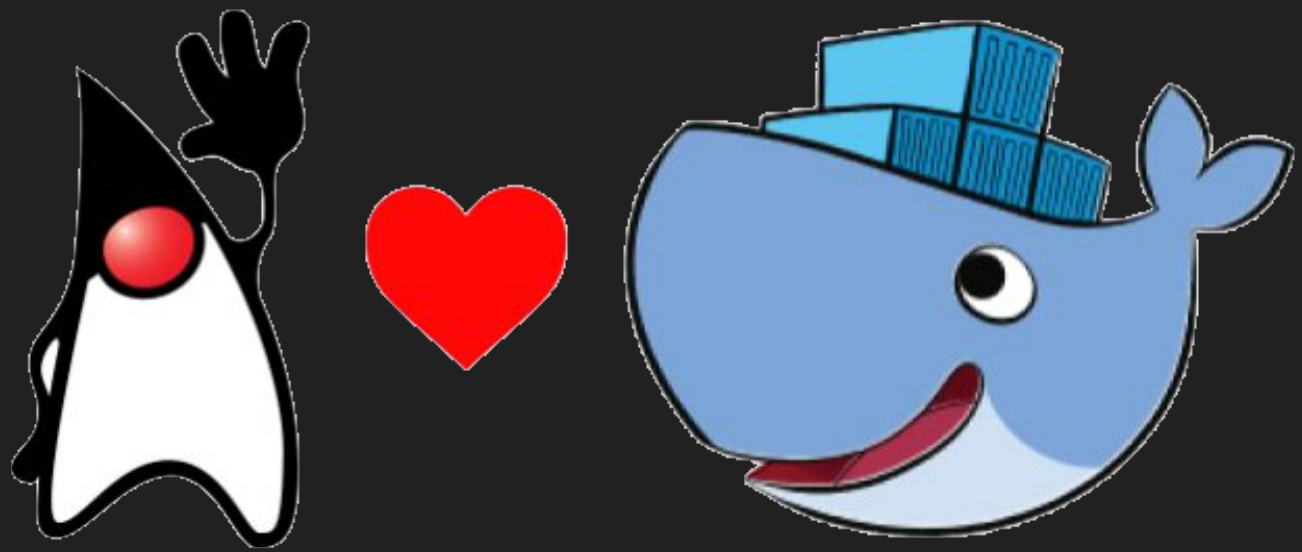
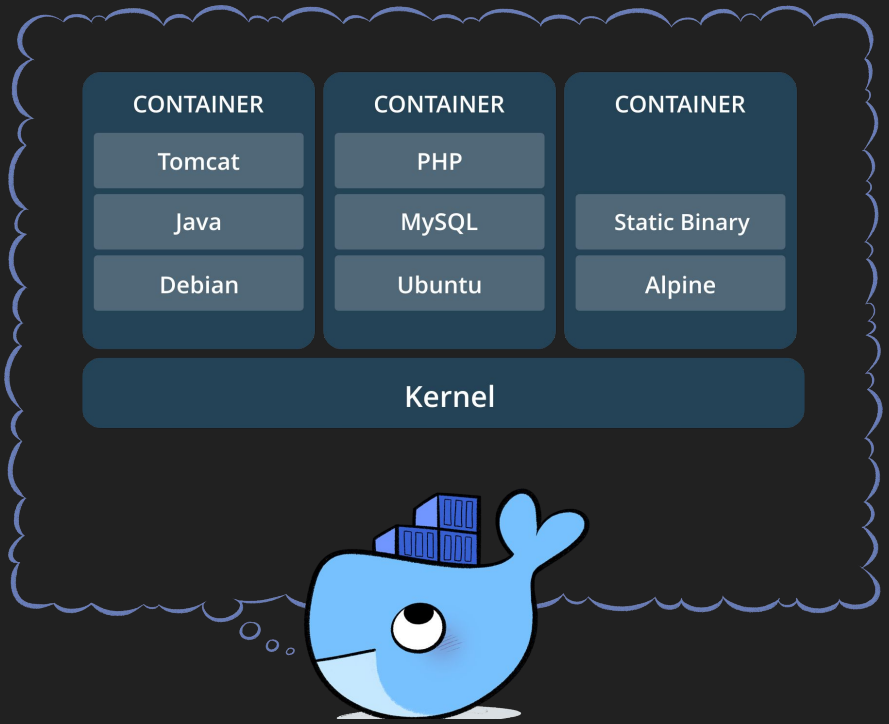
JVM & Docker

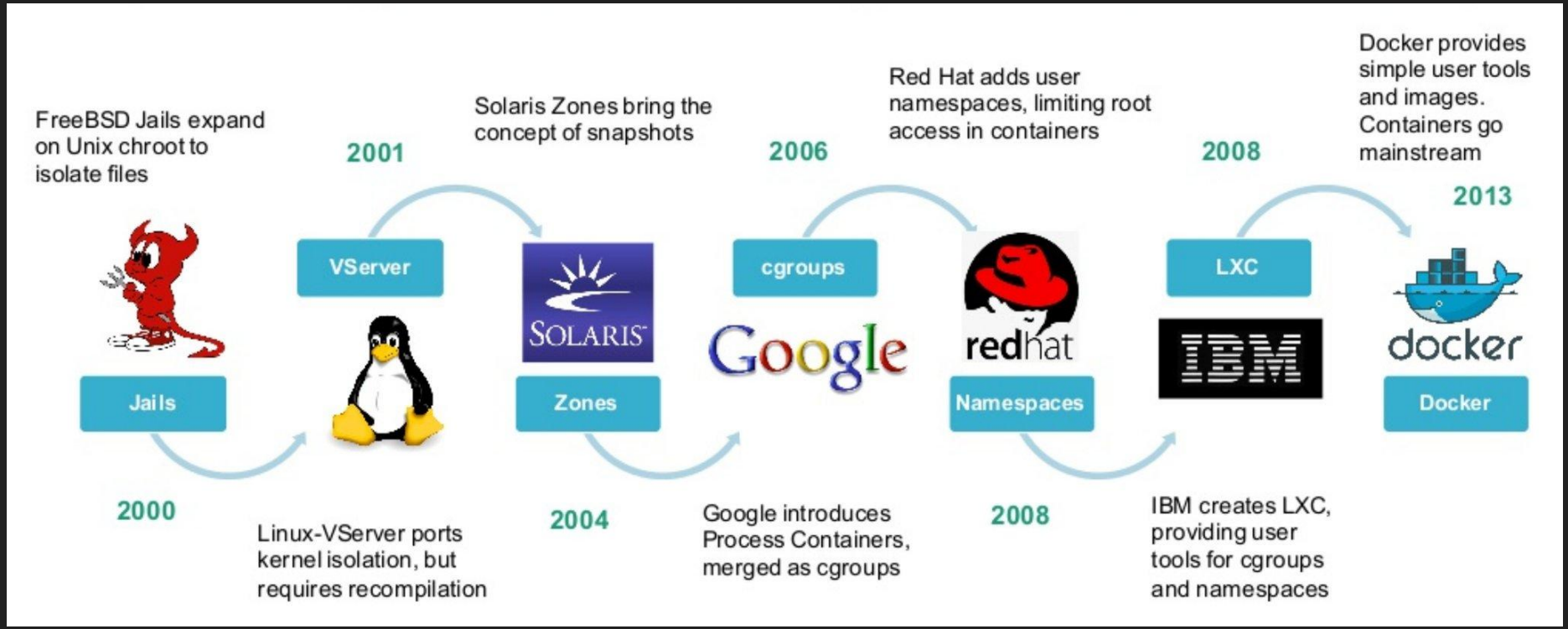


O que é docker?

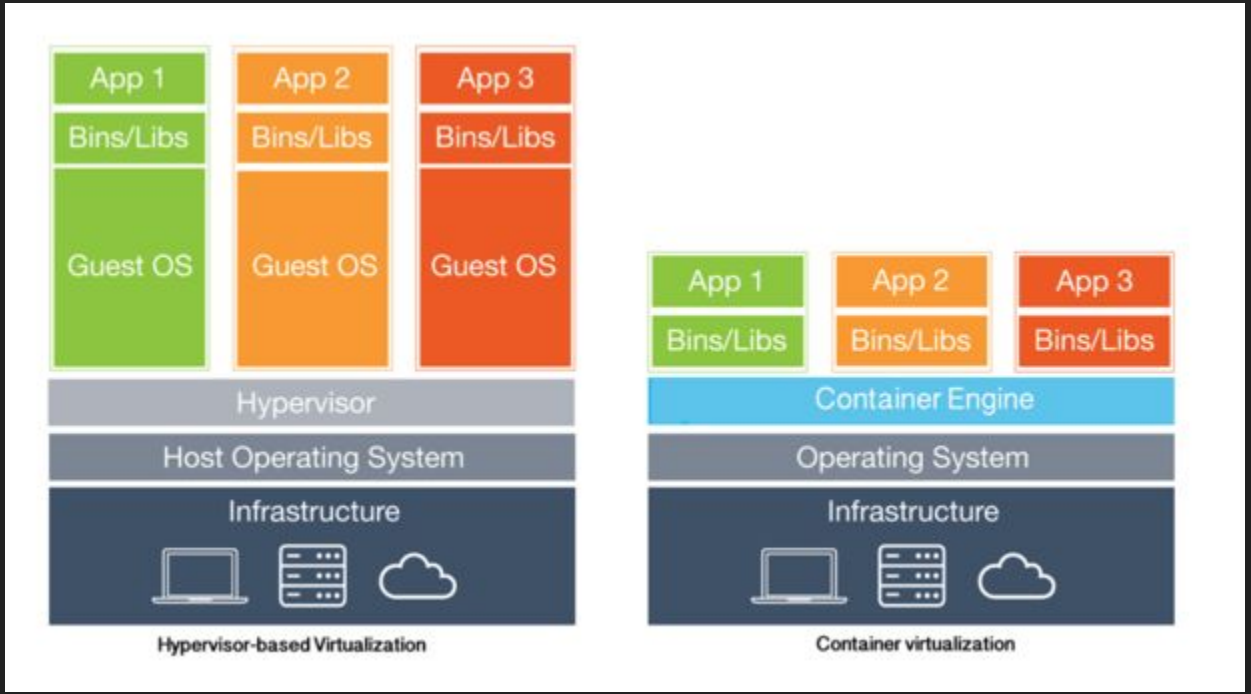


- Tecnologia de containerização para criação e gerenciamento de containers
- Open-source

História dos containers



Virtualização e Containers



Virtualização e Containers

```
FROM ubuntu
MAINTAINER Kimbro Staken

RUN apt-get install -y software-properties-common python
RUN add-apt-repository ppa:chris-lea/node.js
RUN echo "deb http://us.archive.ubuntu.com/ubuntu/ precise universe" >> /etc/apt/sources.list
RUN apt-get update
RUN apt-get install -y nodejs
#RUN apt-get install -y nodejs=0.6.12~dfsg1-1ubuntu1
RUN mkdir /var/www

ADD app.js /var/www/app.js

CMD ["/usr/bin/node", "/var/www/app.js"]
```

Porque Docker?

- Produtividade
- Isolamento
- Gerenciamento de ambientes
- Integração contínua

Por trás das "câmeras"

CGroups e Namespaces

- CGroups
  - cpu share
  - cpuset
  - memory
  - block I/O
  - devices

- Namespaces
  - pid(processos)
  - network(routing, interfaces)
  - ipc
  - mnt(mount points, fs)
  - uts(hostname)
  - user

Java

Porque o Java falha?

- Runtime.getRuntime().maxMemory()
- Runtime.getRuntime().avaiableProcessors()

- Acesso à todos os recursos da máquina Host, não respeita o cgroups

JVM e containers

- Memória
- CPU
- Disk I/O

Ergonomia da JVM

(https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/index.html)

- initial heap size = 1/64 da memória disponível no host
- max heap size = 1/4 da memória disponível no host
- JIT optimizations
- Garbage collector
- Thread management

Memória

Max Heap Size não definido(-Xmx)

- default MaxHeapSize = 25% da memória total
- JVM não reconhece cgroups

Exemplo:

- memória do host: 8Gb
- memória máxima do container: 1Gb
- default heap size = 2Gb

Memória

Memória do container < memória do Java(Heap + Stack)

- O tamanho máximo da heap não é o máximo de memória usada
- Depende da natureza da sua aplicação. Um ponto de partida seria 70% da memória

Exemplo:

- memória máxima do container: 1GB
- max heap size = 1GB
- new heap size = 700MB

Memória

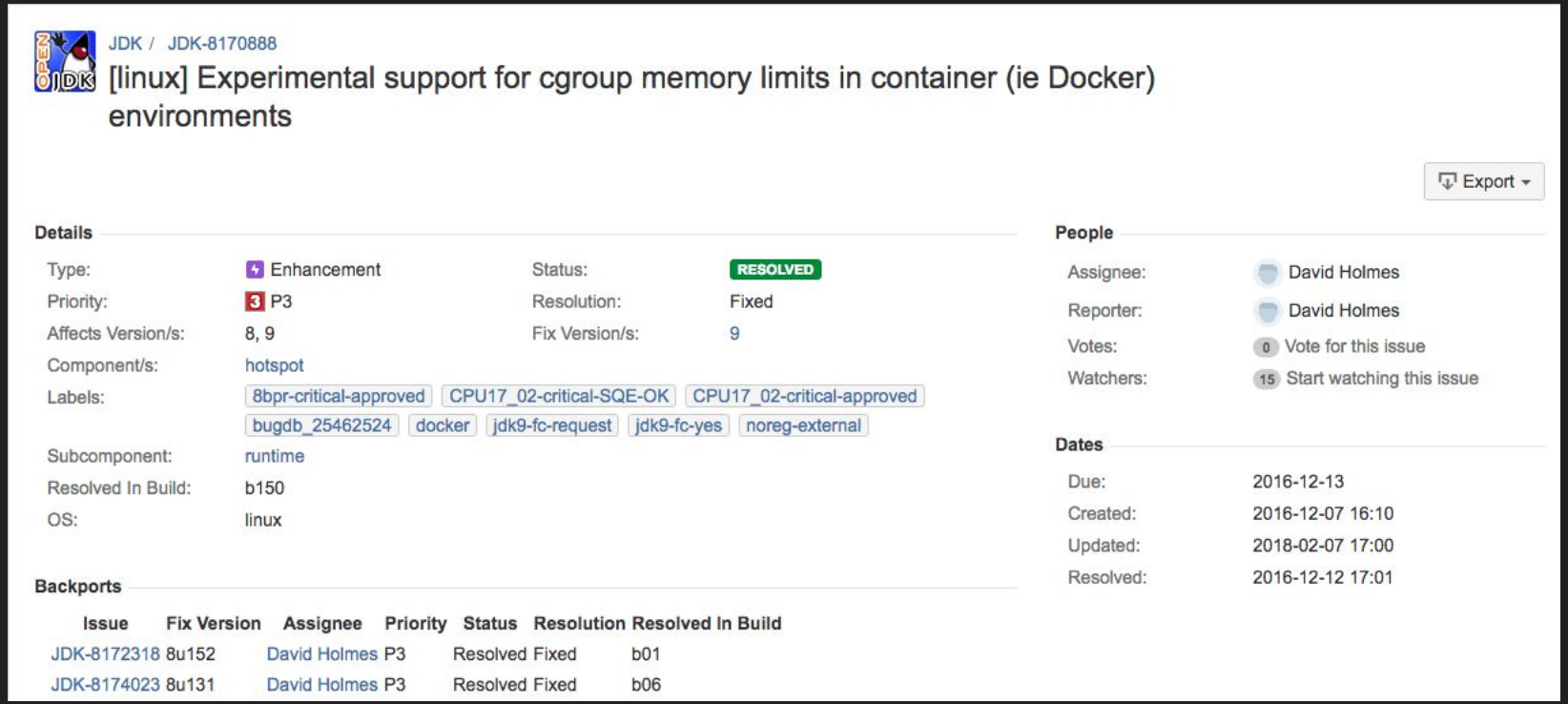
partição de SWAP inexistente

- Sua máquina de produção tem swap?
- SWAP limit default de 2 * memory limit

Exemplo:

- memória máxima do container: 1GB
- max container swap = 2GB

Heap patched Java 8u131



JDK / JDK-8170888

[linux] Experimental support for cgroup memory limits in container (ie Docker) environments

Export

Details

Type:	Enhancement	Status:	RESOLVED
Priority:	P3	Resolution:	Fixed
Affects Version/s:	8, 9	Fix Version/s:	9
Component/s:	hotspot		
Labels:	8bpr-critical-approved, CPU17_02-critical-SQE-OK, CPU17_02-critical-approved, bugdb_25462524, docker, jdk9-fc-request, jdk9-fc-yes, noreg-external		
Subcomponent:	runtime		
Resolved In Build:	b150		
OS:	linux		

People

Assignee:	David Holmes
Reporter:	David Holmes
Votes:	0 Vote for this issue
Watchers:	15 Start watching this issue

Dates

Due:	2016-12-13
Created:	2016-12-07 16:10
Updated:	2018-02-07 17:00
Resolved:	2016-12-12 17:01

Backports

Issue	Fix Version	Assignee	Priority	Status	Resolution	Resolved In Build
JDK-8172318	8u152	David Holmes	P3	Resolved	Fixed	b01
JDK-8174023	8u131	David Holmes	P3	Resolved	Fixed	b06

Memória

JDK 8u131 e JDK 9:

- -XX:+UnlockExperimentalVMOptions
- -XX:+UseCGroupMemoryLimitForHeap

JDK 10

- Suporte total ao Docker
- Não há necessário das flags anteriores

CPU

- Baixa performance de GC e paralelismo
- JVM não reconhece cgroups!

Exemplo:

- total host cores: 8
- max containers core = 1
- max jvm cores = 8

Solução:

- -XX:ParallelGCThreads
- -XX:ConGCThreads
- -Djava.util.concurrent.ForkJoinPool.common.parallelism

Disk I/O

- Performance SecureRandom
- Exaustão da entropia do host, causando block do /dev/random
- Use /dev/urandom para evitar bloqueios

Solução:

- -Djava.security.egd=file:/dev/urandom

Demo!

Demo Time!

- docker run -m 100MB openjdk:8u121 java -XX:+PrintFlagsFinal -version | grep MaxHeapSize
- docker run -m 100MB openjdk:8u131 java -XX:+PrintFlagsFinal -version | grep MaxHeapSize
- docker run -m 100MB openjdk:8u131 java -XX:+PrintFlagsFinal -XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -version | grep MaxHeapSize
- docker run -m 100MB openjdk:10 java -XX:+PrintFlagsFinal -version | grep MaxHeapSize