Java w dokerze

Dobre praktyki pracy z dockerem i javą

Agenda

- Applikacja Demo
- Sygnały w dockerze
- Wybór bazowego obrazu
- Warstwy aplikacji
- Wieloetapowy build
- Ergonomia javy
- Bezpieczeństwo
- Google JIB
- Extra
- Podsumowanie

Swtórzmy aplikacje

Aplikacja użytkowników

- Serwis użytkownika
- REST Api
- Zapis: MySQL

Użytkownik

```
@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String firstName;
    private String secondName;
    @Temporal(TemporalType.TIMESTAMP)
    private Date birthDate;

    public Integer getId() { return id; }

    public String getFirstName() { return firstName; }

    public String getSecondName() { return secondName; }

    public Date getBirthDate() { return birthDate; }
}
```

Kontroler

```
@RestController
@RequestMapping("/api")
public class UsersController {
    @Autowired
    UserRepository userRepository;

    @GetMapping("/users")
    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    @PostMapping("/users")
    public User createUser(@Valid @RequestBody User user) {
        return userRepository.save(user);
    }

    @GetMapping("/users/{id}")
    public User getUserById(@PathVariable(value = "id") Integer userId) {
        return userRepository.findById(userId)
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", userId));
    }
}
```

Demo usługi

- Budowanie

```
$ mvn package
```

- Uruchomienie

```
$ java -jar target/users-app-1.0.jar
```

- Otwarcie

http://localhost:8080/api/users

Zdokeryzujmy Aplikacje

Google

docker image ubuntu with java

Wszystko | Filmy | Wiadomości | Grafika | Zakupy | Więcej | Ustawienia | Narzędzia

Około 10 500 000 wyników (0,49 s)

ubuntu-java - Docker Hub

https://hub.docker.com › parrotstream › ubuntu-java ▾ Tłumaczenie strony

This **image** adds Oracle **Java** Development Kit to a base **Ubuntu** Linux distribution. The latest tag of this **image** is build using the latest available **Ubuntu Docker** ...

parrotstream/ubuntu-java

By parrotstream • Updated 2 years ago

Docker image with a Ubuntu Linux base distribution and Oracle JDK

Container

ubuntu-java

Description

This image adds Oracle Java Development Kit to a base **Ubuntu** Linux distribution.

The *latest* tag of this image is build using the latest available Ubuntu Docker image and the latest available Oracle JDK. You can pull it with:

```
docker pull parrotstream/ubuntu-java
```

Nasz pierwszy Dockerfile

```
FROM parrotstream/ubuntu-java

RUN mkdir -p /var/users-app

COPY target/users-app-1.0.jar /var/users-app/users-app-1.0.jar
COPY run.sh /var/users-app/run.sh

ENTRYPOINT "/var/users-app/run.sh"
```

Oraz run.sh

```
#!/usr/bin/env sh

echo "Running users App"
/usr/lib/jvm/java-8-oracle/bin/java -jar /var/users-app/users-app-1.0.jar
```

Przygotujmy się na łomot

Sygnały - Entrypoint

Mamy 2 opcje kiedy definiujemy ENTRYPOINT

```
ENTRYPOINT "/usr/bin/java -jar /var/users-app/app-1.0.jar"
```

Lub inny, gdzie linia komend jest jak tablica JSON’a

```
ENTRYPOINT ["java”, „-jar”, ”/var/users-app/app-1.0.jar”]
```

Długa historia w wersji skróconej: zawsze używaj wersji tablicowej. Wersja powłokowa uruchamia entrypoint jako pod polecenie /bin/sh -c co wprowadza wiele problemów, jednym z nich wartym zapamiętania jest że nigdy nie zobaczymy sygnału w aplikacji

Sygnały - skrypt jako entrypoint

- Nie które aplikacje nie są gotowe do uruchomienia w ENTRYPOINT i wymagają pewnej logiki przed uruchomieniem
- Używaj exec - zamiast tworzyć nowy proces z istniejącej powłoki to ją zastąpi

Użyj:

```
exec /app/bin/your-app
```

Zamiast:

```
/app/bin/your-app
```

Sygnały - podsumowanie

- Używaj ENTRYPOINT w formie exec/tablica JSON.
- Używaj exec w skryptach wejściowych.
- Nie używaj strumieni w wyjściu aplikacji.
- Unikaj używania PID 1.
- Nasłuchuj SIGTERM lub ustaw STOPSIGNAL w Dockerfile.

Nasz Dockerfile v2

```
FROM parrotstream/ubuntu-java

RUN mkdir -p /var/users-app

COPY target/users-app-1.0.jar /var/users-app/users-app-1.0.jar

ENTRYPOINT ["/usr/lib/jvm/java-8-oracle/bin/java", "-XshowSettings:VM",
"-XX:NativeMemoryTracking=summary", "-jar", „/var/users-app/users-
app-1.0.jar"]
```

```
agst-19000769:users-app krzypo$ docker build -f v2.Dockerfile -t docker.io/krzpob/users-app:latest .
Sending build context to Docker daemon  76.99MB
Step 1/5 : FROM parrotstream/ubuntu-java
latest: Pulling from parrotstream/ubuntu-java
e8a74323e913: Downloading [==================================>                ]  25.79MB/38.05MB
5fe91835aea8: Download complete
4b2aac3e93a5: Download complete
faefbf4d7e6d: Download complete
071e113a30b5: Download complete
cc307588bf07: Downloading [=================>                                 ]  24.71MB/68.49MB
29ccd6f464c8: Download complete
b5a069e6b905: Download complete
17ec46d2f5bf: Downloading [==>                                                ]  20.88MB/386MB
7c3bbfd92dc7: Waiting
bdcc34a17571: Waiting
```

Sprawdzajmy rozmiar obrazu

```
latest: digest: sha256:ef8ad5ffd338f72afa87414033d7806e2f60ffbd7bf41fc7c0aab21bf8598c9f size: 3244
agst-19000769:users-app krzypo$ docker image ls | grep krzpob
krzpob/users-app                                    latest             4c9af042ecd1        4 minutes ago       882MB
agst-19000769:users-app krzypo$
```

To jest 882MB !!!!!!!!!!!!!!!!!!!!!!

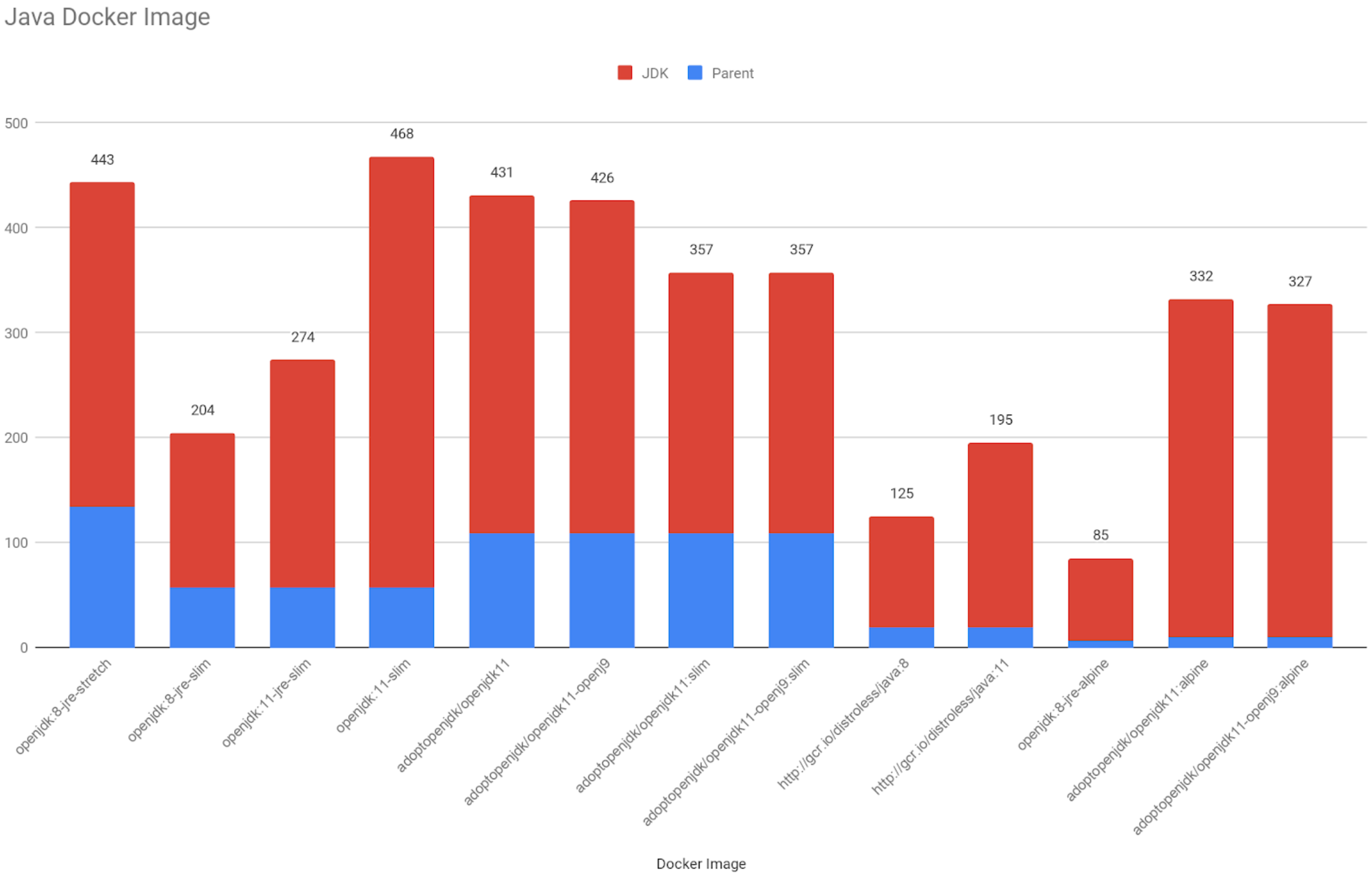
Wybór bazowego obrazu dla kontenera

Pierwszy problem: wybór Java Runtime

- Oracle JDK
- OpenJDK builds by Oracle
- AdoptOpenJDK builds
- AdoptOpenJDK OpenJ9 builds
- Linux (Red Hat, Debian, Fedora, Arch, Ubuntu) OpenJDK builds
- Azul Zulu
- Amazon Corretto
- SAP SapMachine
- Liberica from Bellsoft

Drugi problem: wybór bazowego obrazu linuxa

- Decyzja zależy od wyboru Java Runtime
- Na prawdę potrzebujemy mieć zewnętrzne narzędzia np. wget, curl i inne ?
- Na prawdę troszczymy się o wielkość obrazu?
- Bezpieczeństwo!
  - Powinniśmy wybierać obrazy wolne od CVE
  - Mniej komponentów to mniejsza dziura i mniej potencjalnych miejsc ataku


Java Docker Image
JDK
Parent
500
400
300
200
100
0
443
204
274
468
431
426
357
357
125
195
85
332
327
openjdk:8-jre-stretch
openjdk:8-jre-slim
openjdk:11-jre-slim
openjdk:11-slim
adoptopenjdk/openjdk11
adoptopenjdk/openjdk11-openj9
adoptopenjdk/openjdk11:slim
adoptopenjdk/openjdk11-openj9:slim
http://gcr.io/distroless/java:8
http://gcr.io/distroless/java:11
openjdk:8-jre-alpine
adoptopenjdk/openjdk11:alpine
adoptopenjdk/openjdk11-openj9:alpine
Docker Image

Wybierzmy Alpine

Alpine

- Lekka dystrybucja linuxa bazująca na **musl libc** i **busybox**
- **musl** jest „libc”, implementacją standardowej biblioteki opisanej w standardach ISO C i POSIX
- musl jest nową biblioteką C ogólnego przeznaczenia. Jest lekka, szybka, prosta, darmowa i wydaje się być odpowiednią ze względu na spełnianie standardów i bezpieczeństwo.
- Bazowy ustandaryzwoany interfejs między jądrem a przestrzenią użytkownika

Projekt Partola

- Repozytorium OpenJDK zawiera wiele kodu C/C++ (około 19%)
- JVM został przportowany na glibc
- Ale w celu zapewnienia przejrzystego działania JVM (i jego kodu C/C ++) na musl, należy podjąć wysiłek, aby przenieść do niego JVM.
- To jest celem projektu Partola i jest dostępny jako element java 13

Nasz Dockerfile 3

```
FROM openjdk:8-jre-alpine

RUN mkdir -p /var/users-app

COPY target/users-app-1.0.jar /var/users-app/users-app-1.0.jar

ENTRYPOINT ["java", "-XshowSettings:VM", \\
"-XX:NativeMemoryTracking=summary", "-jar", \\
"/var/users-app/users-app-1.0.jar"]
```

Sprawdźmy rozmiar obrazu

```
agst-19000769:users-app krzypo$ docker image ls | grep krzpob | grep v3
krzpob/users-app                                    v3                  ed2c9460181e        8 seconds ago       125MB
agst-19000769:users-app krzypo$
```

To jest 125m !!!!!!!!!!!!!

Było 882m

Inne opcje zmniejszenia rozmiaru obrazu

- Java
  - Moduły Javy
  - Refaktor kodu
  - Zależności
- Docker
  - Mniejszy obraz bazowy
  - Warstwy i Cache (dockerfilelint)
  - Dockerignore (szybsze budowanie)
  - Wieloetapowy build
  - Docker-squash

Zmieńmy logikę biznesową

```
agst-19000769:users-app krzypo$ docker push docker.io/krzpob/users-app:v3
The push refers to repository [docker.io/krzpob/users-app]
e5e0bcf1f06d: Pushing [==================>                          ]  16.19MB/40.42MB
db23e0426772: Pushed
edd61588d126: Mounted from library/openjdk
9b9b7f3d56a0: Mounted from library/openjdk
f1b5933fe4b5: Mounted from library/openjdk
```

Rozwiązanie: Rozpakować FAT Jar

- Zbudujmy aplikację

```
$ mvn clean package
```

- Rozpakujmy JAR do stworzenia wenętrzych i zewnętrznych zależności

```
$ mkdir target/dependency

$ (cd target/dependency; tar -zxf ../*.jar)
```

Nasz Dokerfile 4

```
FROM openjdk:8-jre-alpine

RUN mkdir -p /var/users-app

COPY target/dependency/BOOT-INF/lib /var/users-app/lib
COPY target/dependency/META-INF /var/users-app/META-INF
COPY target/dependency/BOOT-INF/classes /var/users-app

ENTRYPOINT [ "java", \
    "-XshowSettings:VM", \
    "-XX:NativeMemoryTracking=summary", \
    "-cp", "/var/users-app:/var/users-app/lib/*", \
    "com.example.users.UsersApp"]
```

Plusy rozpakowanego jara

- Szybki build dockera
- Tylko rzeczywiste zmiany są wysyłane przez sieć do rejestru
- Mniej miejsca zajmujemy na rejestrzy i lokalnie

Inny problem

```
MacBook-Pro-Macbook:users-app krzysztof$ DOCKERFILE_VERSION=v4 make docker-build
docker build \
        -f v4.Dockerfile \
        --tag="krzpob/users-app:v4" \
        .
Sending build context to Docker daemon  81.1MB
Step 1/6 : FROM openjdk:8-jre-alpine
 ---> f7a292bbb70c
Step 2/6 : RUN mkdir -p /var/users-app
 ---> Using cache
 ---> d4da981bb2e3
Step 3/6 : COPY target/dependency/BOOT-INF/lib /var/users-app/lib
 ---> Using cache
 ---> acf6bef6680f
Step 4/6 : COPY target/dependency/META-INF /var/users-app/META-INF
 ---> Using cache
 ---> 971734d5837e
Step 5/6 : COPY target/dependency/BOOT-INF/classes /var/users-app
 ---> Using cache
 ---> e04e968f39a6
Step 6/6 : ENTRYPOINT [ "java",     "-XshowSettings:VM",     "-XX:NativeMemoryTracking=summary",
     "-cp", "/var/users-app:/var/users-app/lib/*",     "com.example.users.UsersApp"]
 ---> Using cache
 ---> 4f88b416c2ca
Successfully built 4f88b416c2ca
Successfully tagged krzpob/users-app:v4
```

Musimy zbudować naszą aplikację lokalnie

Rozwiązanie - wieloetapowy build

```
FROM openjdk:8-jdk-alpine as build
WORKDIR /workspace/users-app

COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
COPY src src

RUN ./mvnw install -DskipTests
RUN mkdir -p target/dependency && (cd target/
dependency; jar -xf ../*.jar)

FROM openjdk:8-jre-alpine

RUN mkdir -p /var/users-app
ARG DEPENDENCY=/workspace/users-app/target/dependency
COPY --from=build ${DEPENDENCY}/BOOT-INF/lib /var/users-
app/lib
COPY --from=build ${DEPENDENCY}/META-INF /var/users-app/
META-INF
COPY --from=build ${DEPENDENCY}/BOOT-INF/classes /var/
users-app

ENTRYPOINT [ "java", \
    "-XshowSettings:VM", \
    "-XX:NativeMemoryTracking=summary", \
    "-cp", "/var/users-app:/var/users-app/lib/*", \
    "com.example.users.UsersApp"]
```

BUILD

Run

Plusy wieloetapowego builda

- Nie trzeba wysyłać wszystkich zależnośc do deamona
- Zależności są cachowane przez Dockera
- Buildy są repodukowalne i nie zależą od lokalnej instalacji javy

Ergonomia Javy

Ergonomia Javy

JVM ma wiele opcji ułatwiających pracę i bazujących na podstawowych elemntach systemu:

- Pamięć
- Ilość CPU
- Model CPU
- Garbage Collector
- Optymalizacja JIT

Twoi przyjaciele to:

- -XX:+PrintFlagsFinal
- -XX:+PrintGCDetails
- -XshowSettings:VM
- -XX:NativeMemoryTracking=summary

Java 8u121 i wcześniej

Jest nadzieja …

Fabric8

```
# Based on the cgroup limits, figure out the max number of core we should utilize
core_limit() {
  local cpu_period_file="/sys/fs/cgroup/cpu/cpu.cfs_period_us"
  local cpu_quota_file="/sys/fs/cgroup/cpu/cpu.cfs_quota_us"
  if [ -r "${cpu_period_file}" ]; then
    local cpu_period="$(cat ${cpu_period_file})"

    if [ -r "${cpu_quota_file}" ]; then
      local cpu_quota="$(cat ${cpu_quota_file})"
      # cfs_quota_us == -1 --> no restrictions
      if [ ${cpu_quota:-0} -ne -1 ]; then
        echo $(calc 'ceil($1/$2)' "${cpu_quota}" "${cpu_period}")
      fi
    fi
  fi
}

max_memory() {
  # High number which is the max limit until which memory is supposed to be
  # unbounded.
  local mem_file="/sys/fs/cgroup/memory/memory.limit_in_bytes"
  if [ -r "${mem_file}" ]; then
    local max_mem_cgroup="$(cat ${mem_file})"
    local max_mem_meminfo_kb="$(cat /proc/meminfo | awk '/MemTotal/ {print $2}')"
    local max_mem_meminfo="$(expr $max_mem_meminfo_kb \* 1024)"
    if [ ${max_mem_cgroup:-0} != -1 ] && [ ${max_mem_cgroup:-0} -lt ${max_mem_meminfo:-0} ]
    then
      echo "${max_mem_cgroup}"
    fi
  fi
}
```

Source: github.com/fabric8io-images/run-java-sh

Java 9, 8u121+

Java 9, 8u121+

Konfiguracja CPU

-XX:ParallelGCThreads i -XX:CICompilerCount są zbiorem parametrów bazujących na ograniczeniach CPU dla kontenera (mogą być nadpisane) , i bazuja na fladze dockera

--cpuset-cpus

Konfiguracja pamięci

-XX:+UnlockExperimentalVMOptions

-XX:+UseCGroupMemoryLimitForHeap

-XX:MaxRAMFraction do 2 (domyślnie jest 4)

Java 10+, Java 8u191+

Java 10+, Java8u191+

- Większa świadomość kontenera
- Ulepszona alokacja i zarządzanie stosem [JDK-8196595]
  - -XX:InitialRAMPercentage, -XX:MinRAMPercentage, I -XX:MaxRAMPercentage
  - -XX:InitialRAMFraction, -XX:MinRamFraction, i -XX:MinRAMFraction są przestarzałe
- Całkowita ilość CPU dostępnych dla Javy jest liczona z parametrów dociera: --cpus, --cpu-shares, --cpu-quota [JDK-8146115]
  - By przywrócić stare działanie należy użyć: -XX:-UseContainerSupport
- Dołączanie w linuxe stało się względem /proc/pid/root i jest świadoma przestrzeni nazw (jcmd, jstack itp.)

Java 11+

Java 11+

- Bardziej świadoma kontenerów
- Usunięto -XX:+UnlockExperimentalVMOptions, -XX:+UseCGroupMemoryLimitForHeap [JDK-8194086]
- Polecenia jcmd -l I jps nie pokazują JVM wewnątrz kontenerów
- Metryki kontenera (-XshowSettings:system) [JDK-8204107]
- Zaktualizowany algorytm wyliczający dostępne CPU jak użyto limitu dla liczby CPU i quot [JDK-8197867]
  - -XX:+PreferContainerQuotaForCPUCount

Podsumowanie wygody używania Javy

- Jeśli nie można zmienić wersji na jakąś nowszą w dokerze, to użyć skryptu fabric8, wspomagającego uruchomienie aplikacji
- Jeśli można to:
  - Używać java 11 lub nowsza,
  - Lub przynajmniej 8u191+

Bezpieczeństwo

Bezpieczeństwo w kontenerze

- Wiedz co zawiera kontener
- Skanuj podatność na zagrożenia - np. Clair
- Zaufana zawartość dockera
- Kontener zawiera minimum usług i oprogramowania (minimalny obszar ataku)
- Monitorowanie i kontrola

Bezpieczeństwo w kontenerze

Ograniczamy uprawnienia

- Zmieniamy użytkownika USER na nie-root
- Uruchamiamy kontener z systemem pliku tylko do odczytu
- Ograniczamy zasoby

Nasz Dockerfile 6

```
FROM openjdk:8-jdk-alpine as build
WORKDIR /workspace/users-app

COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
COPY src src

RUN ./mvnw install -DskipTests
RUN mkdir -p target/dependency && (cd target/dependency; jar -xf ../*.jar)

FROM openjdk:8-jre-alpine
RUN addgroup -S app && adduser -S -G app app

USER app
RUN mkdir -p /home/app/users-app

ARG DEPENDENCY=/workspace/users-app/target/dependency

COPY --from=build ${DEPENDENCY}/BOOT-INF/lib /home/app/users-app/users-app/lib
COPY --from=build ${DEPENDENCY}/META-INF /home/app/users-app/users-app/META-INF
COPY --from=build ${DEPENDENCY}/BOOT-INF/classes /home/app/users-app/users-app

ENTRYPOINT [ "java", \
   "-XshowSettings:VM", \
   "-XX:NativeMemoryTracking=summary", \
   "-cp", "/home/app/users-app/users-app:/home/app/users-app/users-app/lib/*", \
   "com.example.users.UsersApp"]
```

Magic

Clair

- Clair jest opensoure'owym projektem do statycznej analizy podatności na ataki aplikacji w kontenerach
- W regularnych odstępach czasu Clair przyswaja metadane związane z zagrożeniami bezpieczeństwa z konfigurowalnych źródeł i odkłada je w bazie
- Klieci używają Clair API by odpytać bazę danych o zagrożenia bezpieczeństwa dla danego obrazu

clair

Clair-Scanner

- Klient dla Clair: github.com/arminc/clair-scanner
- Użycie:

```
$ clair-scanner alpine:3.5
2017/09/24 11:20:24 [INFO] Start clair-scanner
2017/09/24 11:20:24 [INFO] Server listening on port 9279
2017/09/24 11:20:24 [INFO] Analyzing 693bdf455e7bf0952f8a4539131
2017/09/24 11:20:24 [INFO] Image [alpine:3.5] not vulnerable
```

```
$ clair-scanner alpine:3.4
2019/09/24 11:16:41 [INFO] Start clair-scanner
2019/09/24 11:16:41 [INFO] Server listening on port 9279
2019/09/24 11:16:41 [INFO] Analyzing
693bdf455e7bf0952f8a45131323 2017/09/24 11:16:41 [CRIT] Image
contains unapproved vulnerabilities: [CVE-2016-9840 CVE-2016-9841
CVE-2016-9842 CVE-2016-9843]
```

Jib

Containerize your Java application.

Google JIB

Google JIB

- Wszystkie dobre praktyki tworzenia obrazu dockerowego
- Tworzy obraz dockerowy
- Nie wymaga Dockerfile
- Bazuje na `gcr.io/distroless/java`
- Pluginy dla Gradle i maven
- Opensource: github.com/GoogleContainerTools/jib

Google JIB

Plugin do maven

```
<build>
        <plugins>
            <plugin>
                <groupId>com.google.cloud.tools</groupId>
                <artifactId>jib-maven-plugin</artifactId>
                <version>1.6.1</version>
                <configuration>
                    <to>docker.io/krzpob/users-app:$
{env.DOCKERFILE_VERSION}</to>
                </configuration>
            </plugin>
        </plugins>
</build>
```

Google JIB

Użycie:

Budowanie obrazu:

```
$ mvn compile jib:build
```

Budowanie obrazu pod lokalnego dociera:

```
$ mvn compile jib:dockerBuild
```

Google JIB - magia wewnątrz

```
FROM gcr.io/distroless/java
COPY target/dependencies /app/dependencies
COPY target/resources /app/resources
COPY target/classes /app/classes
ENTRYPOINT java -cp /app/dependencies/*:/app/resources:/app/classes my.app.Main
```

Ekstra

Moduły Javy w kontekście Dockera

- Nieco szybszy start
- Wymagane okrojone JRE
- Mniejszy obraz, bo mniejsze JRE

GraalVM w kontekście Dockerowym

- Znacznie szybsze uruchomienie
- Nie wymagane JRE
- Mniejszy obraz dockerowy

Podsumowanie

Co chcę byście zapamiętali

- Jak tylko można, używamy Javy 11+ (lub przynajmniej jak 8u191+)
- Rozpakować fat JARa, by zmiany między obrazami byle małe
- Nie uruchamiamy aplikacji jako root!
- Można zoptymalizować czas startu.
- Jeśli nie potrzebujemy utrzymywać obrazu docker, użyć Google JIB