DevOps Vietnam

Data Online

Dockerfile Contest

Trịnh Thế Long – Dockerfile Java Spring boot

Dockerfile Contest - Java

01 Lựa chọn Base Image

02 Kiểm tra thư viện, Vulnerabilities

03 Tối ưu layers, caching

04 Native Java Healthcheck

05 Entrypoint

Dockerfile mẫu

spring-boot-template > Dockerfile_sample > ...

```dockerfile
# Stage 1: build bằng Gradle
FROM gradle:8.10-jdk21 AS build (last pushed 1 year ago)
WORKDIR /home/gradle/src
COPY . .
RUN ./gradlew --no-daemon clean bootJar

# Stage 2: runtime
FROM hmctspublic.azurecr.io/base/java:21-distroless
COPY lib/applicationinsights.json /opt/app/
# copy JAR đã build từ stage trước
COPY --from=build /home/gradle/src/build/libs/*.jar /opt/app/spring-boot-template.jar

EXPOSE 8080
CMD ["spring-boot-template.jar"]

```

1. Lựa chọn Base Image

https://hub.docker.com/_/gradle

jdk-lts-and-current-graal-noble , 9.2-jdk-lts-and-current-graal-noble , 9-jdk-lts-and-current-graal-noble , jdk-lts-and-current-graal-noble , 9.2.0-jdk-25-and-25-graal , 9.2-jdk-25-and-25-graal , 9-jdk-25-and-25-graal , jdk-25-and-25-graal , 9.2.0-jdk-25-and-25-graal-noble , 9.2-jdk-25-and-25-graal-noble , 9-jdk-25-and-25-graal-noble , jdk-25-and-25-graal-noble

- 8.14.3-jdk21 , 8.14-jdk21 , 8-jdk21 , 8.14.3-jdk21-noble , 8.14-jdk21-noble , 8-jdk21-noble , 8.14.3-jdk , 8.14-jdk , 8-jdk , 8.14.3 , 8.14 , 8 , 8.14.3-jdk-noble , 8.14-jdk-noble , 8-jdk-noble , 8.14.3-noble , 8.14-noble , 8-noble
- 8.14.3-jdk21-jammy , 8.14-jdk21-jammy , 8-jdk21-jammy , 8.14.3-jdk-jammy , 8.14-jdk-jammy , 8-jdk-jammy , 8.14.3-jammy , 8.14-jammy , 8-jammy
- 8.14.3-jdk21-alpine , 8.14-jdk21-alpine , 8-jdk21-alpine , 8.14.3-jdk-alpine , 8.14-jdk-alpine , 8-jdk-alpine , 8.14.3-alpine , 8.14-alpine , 8-alpine
- 8.14.3-jdk21-corretto , 8.14-jdk21-corretto , 8-jdk21-corretto , 8.14.3-jdk21-corretto-al2023 , 8.14-jdk21-corretto-al2023 , 8-jdk21-corretto-al2023
- 8.14.3-jdk21-ubi , 8.14-jdk21-ubi , 8-jdk21-ubi , 8.14.3-jdk21-ubi-minimal , 8.14-jdk21-ubi-minimal , 8-jdk21-ubi-minimal
- 8.14.3-jdk21-graal , 8.14-jdk21-graal , 8-jdk21-graal , 8.14.3-jdk-graal , 8.14-jdk-graal , 8-jdk-graal , 8.14.3-graal , 8.14-graal , 8-graal , 8.14.3-jdk21-graal-noble , 8.14-jdk21-graal-noble , 8-jdk21-graal-noble , 8.14.3-jdk-graal-noble , 8.14-jdk-graal-noble , 8-jdk-graal-noble , 8.14.3-graal-noble , 8.14-graal-noble , 8-graal-noble
- 8.14.3-jdk21-graal-jammy , 8.14-jdk21-graal-jammy , 8-jdk21-graal-jammy , 8.14.3-jdk-graal-jammy , 8.14-jdk-graal-jammy , 8-jdk-graal-jammy , 8.14.3-graal-jammy , 8.14-graal-jammy , 8-graal-jammy

1. Lựa chọn Base Image

Base	OS	Size	Use case
Default jdk	Debian/Ubuntu	Medium	All - General
jdk-jammy	Ubuntu 22.04 (Jammy Jellyfish)	Medium	Legacy/enterprise
jdk-corretto	Amazon Linux	Medium	AWS focused
jdk-ubi	Red Hat Universal base Image	Large	Large Enterprise
jdk-graal	Oracle/GraalVM	Large	Native image
*** jdk-alpine**	**Alpine – musl**	**Small**	**Lightweight apps**

1. Lựa chọn Base Image

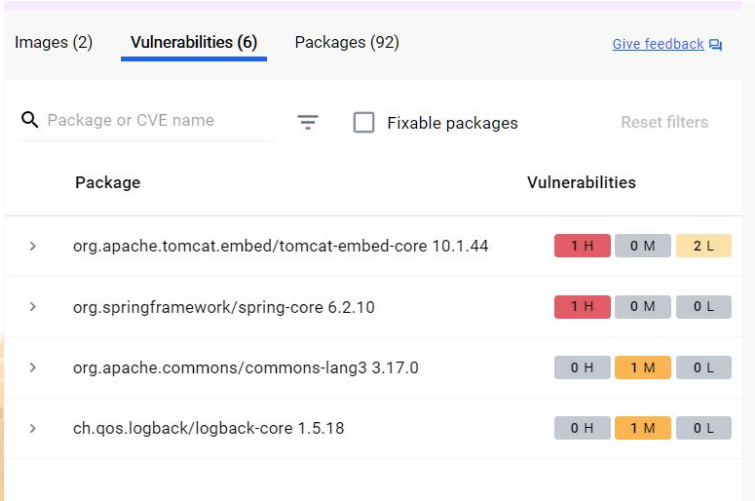
- Build stage: Gradle version 8.14.3
- Runtime stage: Distroless
  - Nonroot default
  - No shell: sh, bash
  - No package manager: apt, yum
  - No OS tool: sed, curl, ps…
  - No compiler tool

2. Kiểm tra thư viện, vulnerabilities

* Điều kiện tới giai đoạn này là ta đã build thành công docker image từ step 1 bằng cách lựa chọn image hợp lý để loại bỏ tối đa Vulnerabilities từ OS/Base image

- Tại đây, ta scan docker image lần đầu tiên, sẽ thấy có nhiều lỗ hổng bảo mật
  => Mình sẽ tìm hướng tiếp cận Fix các lỗ hổng bảo mật từ giai đoạn này

Ví dụ:



2. Kiểm tra thư viện, vulnerabilities

Hướng tiếp cận vấn đề mình đã làm:

1. Kiểm tra trong build.gradle xem có thư viện nào auto check hoặc update thư viện không?

2. Nếu không có thì mình sẽ phải tự viết tay script check lỗ hổng bảo mật và tự vá chính nó.

=> Tuy nhiên ở đề này đã có sẵn thư viện kiểm tra, nên ta sẽ tận dụng luôn

```
plugins {
  id 'application'
  id 'jacoco'
  id 'io.spring.dependency-management' version '1.1.7'
  id 'org.springframework.boot' version '3.5.5'
  id 'com.github.ben-manes.versions' version '0.52.0'
  id 'org.sonarqube' version '6.3.1.5724'
  /*
    Applies analysis tools including checkstyle and OWASP Dependency checker.
    See https://github.com/hmcts/gradle-java-plugin
  */
  id 'uk.gov.hmcts.java' version '0.12.67'
}
```

2. Kiểm tra thư viện, vulnerabilities

Tìm hiểu được thư viện này hoạt động thế nào thông qua đọc tài liệu trên Github
https://github.com/ben-manes/gradle-versions-plugin

Mình sẽ chạy lệnh:

./gradlew --no-daemon dependencyUpdates -Drevision=release

=> Sẽ sinh ra file report.txt với format dưới đây, và từ đây mình xây dựng script update thư viện như trong Dockerfile

2. Kiểm tra thư viện, vulnerabilities

Lợi ích:

- Với cách tiếp cận DevSecOps này sẽ chủ động update thư viện, không phải chờ Repo update rồi build lại. Có thể update ngay khi có scan phát hiện vulnerability, chủ động tránh rủi ro từ sớm nhất.

- Rút ngắn vòng quay vá lỗi khi có CVE zero-day ở thư viện phổ biến.

- Tạo cơ hội chuẩn hoá dependency theo policy nội bộ mà không đợi chỉnh tay repo.

2. Kiểm tra thư viện, vulnerabilities

Report.txt từ thư viện com.github.ben-manes.versions

build > dependencyUpdates > report.txt

```
------------------------------------------------------------
: Project Dependency Updates (report to plain text file)
------------------------------------------------------------

The following dependencies are using the latest milestone version:
 - com.github.hmcts.java-logging:logging:6.1.9
 - com.puppycrawl.tools:checkstyle:9.3
 - io.rest-assured:rest-assured:5.5.6
 - io.spring.dependency-management:io.spring.dependency-management.gradle.plugin:1.1.7
 - org.jacoco:org.jacoco.agent:0.8.13
 - org.jacoco:org.jacoco.ant:0.8.13
 - uk.gov.hmcts.java:uk.gov.hmcts.java.gradle.plugin:0.12.67

The following dependencies have later milestone versions:
 - ch.qos.logback:logback-classic [1.5.18 -> 1.5.20]
     http://logback.qos.ch
 - ch.qos.logback:logback-core [1.5.18 -> 1.5.20]
     http://logback.qos.ch
 - com.github.ben-manes.versions:com.github.ben-manes.versions.gradle.plugin [0.52.0 -> 0.53.0]
 - org.apache.logging.log4j:log4j-api [2.25.1 -> 2.25.2]
     https://logging.apache.org/log4j/2.x/
 - org.apache.logging.log4j:log4j-to-slf4j [2.25.1 -> 2.25.2]
     https://logging.apache.org/log4j/2.x/
 - org.sonarqube:org.sonarqube.gradle.plugin [6.3.1.5724 -> 7.0.1.6134]
 - org.springdoc:springdoc-openapi-starter-webmvc-ui [2.8.13 -> 2.8.14]
     https://springdoc.org/
 - org.springframework.boot:org.springframework.boot.gradle.plugin [3.5.5 -> 3.5.7]
     https://spring.io/projects/spring-boot
 - org.springframework.boot:spring-boot-starter-actuator [3.5.5 -> 3.5.7]
     https://spring.io/projects/spring-boot
 - org.springframework.boot:spring-boot-starter-aop [3.5.5 -> 3.5.7]
     https://spring.io/projects/spring-boot
 - org.springframework.boot:spring-boot-starter-json [3.5.5 -> 3.5.7]
     https://spring.io/projects/spring-boot
 - org.springframework.boot:spring-boot-starter-test [3.5.5 -> 3.5.7]
```

3. Tối ưu layers, caching

Chia 2 stages: Builder & Runtime:

- Toàn bộ build tool, jdk,… tất cả nằm trong builder
- Runtime chỉ nhận các artifacts cần thiết để run

3. Tối ưu layers, caching

Build stage:

- Ưu tiên khởi tạo gradle daemon trước để chủ động cache gradle do gradle version sẽ gần như không update xuyên suốt thời gian development

```
FROM gradle:8.14.3-jdk21-alpine AS builder (last pushed 6 days ago)

WORKDIR /app

# Caching wrapper and build configuration before build
COPY gradlew ./
COPY gradle gradle
COPY build.gradle build.gradle

# Caching gradle/download
RUN ./gradlew --no-daemon help
```

3. Tối ưu layers, caching

Build stage:

- Sau đó sẽ chủ đích copy source code (chỉ copy source, không copy test vì không cần thiết)

```
# Copy src code
COPY src/main src/main

# Check dependency need to update
RUN ./gradlew --no-daemon dependencyUpdates -Drevision=release

# Auto update for auto vulnerability fixing
RUN REPORT_FILE="build/dependencyUpdates/report.txt" && \...

# Build the application JAR after dependency check
RUN ./gradlew --no-daemon bootJar
```

3. Tối ưu layers, caching

Build stage:

- Cuối cùng sẽ generate 1 Class Healthcheck

```
# Generate java healthcheck class
RUN mkdir -p /app/health && cat > /app/health/HealthCheck.java <<'EOF' ···

# Compile HealthCheck.java file
RUN javac /app/health/HealthCheck.java
```

3. Tối ưu layers, caching

Runtime stage:

* Runtime chỉ nhận các artifacts cần thiết để run, không cài thêm bất kỳ thứ gì. Đúng nghĩa distroless

```
FROM hmctspublic.azurecr.io/base/java:21-distroless

WORKDIR /app

# Copy compiled app
COPY --from=builder /app/build/libs/*.jar app.jar

# Copy complied healthcheck class
COPY --from=builder /app/health/HealthCheck.class /app/HealthCheck.class

EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]

HEALTHCHECK --interval=15s --timeout=5s --start-period=10s --retries=3 \
    CMD ["java", "HealthCheck"]
```

4. Native Java Healthcheck

Để tiếp cận với vấn đề Healthcheck sẽ có 1 số hướng thông dụng:

1. **Thêm package shell, curl, wget vào runtime**
   - Nếu "cố tình" thêm vào thì sẽ được nhưng sẽ trái với bản chất distroless

⇒ *Có thể có tiềm ẩn vulnerability trong các package shell, curl mình thêm vào nếu không được scan kỹ. Sẽ cần build lại distroless image để custom và vá lỗ hổng sau 1 thời gian sử dụng. Không tận dụng được image luông được cập nhật và "sạch" từ Google*

2. Dựng thêm service healthcheck trong docker compose hoặc k8s

⇒ Khả thi và đảm bảo an toàn nhưng phát sinh thêm chi phí maintenance cho service healthcheck đó.

** Tuy nhiên kỹ thuật này có sai số do cần public 1 local port trong docker compose để các services giao tiếp với nhau do không cùng TLS stack với ứng dụng*

4. Native Java Healthcheck

```
# Generate java healthcheck class
RUN mkdir -p /app/health && cat > /app/health/HealthCheck.java <<'EOF'
import java.net.HttpURLConnection;
import java.net.URL;
import java.time.Instant;

public class HealthCheck {
    public static void main(String[] args) {
        String healthUrl = "http://localhost:8080/health";
        try {
            URL url = new URL(healthUrl);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setConnectTimeout(2000);
            conn.setReadTimeout(2000);
            conn.setRequestMethod("GET");

            int code = conn.getResponseCode();
            if (code == 200) {
                System.out.println(Instant.now() + "Healthcheck OK (" + code + ")");
                System.exit(0);
            } else {
                System.err.println(Instant.now() + "Healthcheck failed (" + code + ")");
                System.exit(1);
            }
        } catch (Exception e) {
            System.err.println(Instant.now() + " Healthcheck error: " + e.getMessage());
            System.exit(1);
        }
    }
}
EOF

# Compile HealthCheck.java file
RUN javac /app/health/HealthCheck.java
```

4. Native Java Healthcheck

Với kỹ thuật Generate 1 class nhỏ Java này sẽ chạy trực tiếp local cùng với Container của ứng dụng. Điều này đảm bảo:

- Tính bảo mật và bản chất distroless: Không cài thêm bất cứ thứ gì trong runtime
- Đồng nhất tuyệt đối môi trường giữa ứng dụng và Healtcheck do đều nằm trong cùng 1 container khi run. Giảm thiểu tối đa latency cho heathcheck tại môi trường production. Không có chuyện call thành công nhưng app fail do khác TLS hoặc DNS stack.
- Không phải vận hành thêm bất cứ service healthcheck nào, giảm thiểu chi phí cho production
- Khi có sự cố, log của APP và Healthcheck trong cùng 1 luồng JVM, dễ dàng tích hợp với những hệ thống log tập trung

5. Entrypoint

```
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Trong dạng này:

Docker chạy **trực tiếp** process java trong container.

- Process java trở thành **PID 1** trong container.

5. Entrypoint

```
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Trong Linux, process **PID 1** có trách nhiệm đặc biệt:

- Nhận **tín hiệu** (signals) từ Docker/Kubernetes (SIGTERM, SIGINT…)
- Quản lý lifecycle của ứng dụng
- Nếu JVM là PID 1 → JVM sẽ **nhận trực tiếp** SIGTERM từ orchestrator (Docker, Kubernetes, ECS…).

5. Entrypoint

Lợi ích:

Spring Boot shutdown đúng trình tự

* Khi nhận SIGTERM:
* Spring Boot kích hoạt **graceful shutdown**
* Đóng thread pools
* Tắt web server (Tomcat)
* Hoàn thành request đang xử lý
* Đảm bảo clean shutdown

Nếu SIGTERM không đến đúng nơi → ứng dụng có thể bị kill đột ngột, gây:

* mất request
* lỗi database connection
* corrupted cache
* log không gửi kịp

5. Entrypoint

Lợi ích:
Ngoài ra có thể tránh việc cần init process phụ: dumb-init

Bởi vì:

- Nếu ENTRYPOINT "java -jar app.jar“ **=> PID 1 là** shell, có khả năng không forward tín hiệu đúng
- Nếu ENTRYPOINT ["dumb-init", "--", "java", "-jar", "app.jar"] => Cần dumb-init, image sẽ nặng hơn