**Microservice Architecture Style Guide and Best Practices**

Version1 : 24 enero 2025

Revision: rodrigo aaron martinez tellez

Buenas pueden usar estos dos enlaces como referencias pagina oficial de spring.io

Para basarse en la construcción de apis con enfoque microservicio  
  
<https://spring.io/blog/2015/07/14/microservices-with-spring>  
<https://medium.com/ms-club-of-sliit/lets-build-a-microservice-with-spring-boot-faf39b968857>

<https://spring.io/guides/tutorials/rest>

comparto enlaces interesantes para lectura :

<https://github.com/katopz/best-practices/blob/master/best-practices-for-building-a-microservice-architecture.md>

<https://piotrminkowski.com/2019/12/06/spring-boot-best-practices-for-microservices/>  
<https://martinfowler.com/microservices/>  
<https://martinfowler.com/bliki/CircuitBreaker.html>  
<https://medium.com/ms-club-of-sliit/spring-boot-microservices-best-practices-and-coding-style-guidelines-d48aa371b75e>  
<https://github.com/devalexandre/microservice-style-guide>  
<https://amplication.com/blog/the-complete-microservices-guide>

key points :

Collect metrics

Don’t forget about logging

Make your API usable

Don’t afraid of using circuit breaker

Make your application transparent

Write contract tests

Be up-to-date

Decentralized Data Management

Design for failure

Componentization via Services

Organized around Business Capabilities

Infrastructure Automation

[**https://www.devskillbuilder.com/18-essential-microservice-best-practices-655fd4d20ee6**](https://www.devskillbuilder.com/18-essential-microservice-best-practices-655fd4d20ee6)

**Difference between docker-compose and Dockerfile**

The key difference between the Dockerfile and docker-compose is that the Dockerfile describes how to *build* Docker images, while docker-compose is used to *run* Docker containers.

<https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Dockerfile-vs-docker-compose-Whats-the-difference>

si requerimos enlace para event-driven architecture para notificar y demás:

<https://spring.io/blog/2019/10/15/simple-event-driven-microservices-with-spring-cloud-stream>

**Deploy**

When using Microservices, you need to be aware that all services have a separate job, so you have to make a container for each service. This allows you to, scale one service at a time.

When you use a service separately, you may use different servers. Example: to have a service on AWS and another service on Heruko and API gateway on GCP. Remember that all communications are done through messaging.

**Sugerencias en spring boot :**

Usar application.properties file in src/main/resources e intentar que la api sea simple internamente con única responsabilidad, intentemos declarar lo que no forma parte de la api como una entidad externa

spring.datasource.url=jdbc:mysql://localhost:3306/orderservice

spring.datasource.username=root

spring.datasource.password=yourpassword

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

server.port=8081

dentro de cada directorio del proyecto del microservicio crear su propio archivo del tipo Dockerfile.yml como el ejemplo a continuación:

#stage 1  
#Start with a base image containing Java runtime (Java 17)  
FROM openjdk:17-slim as *build*# Add Maintainer Info  
LABEL maintainer="name\_dev <email\_dev@gmail.com>"  
# The application's jar file  
ARG *JAR\_FILE*# Add the application's jar to the container  
COPY ${*JAR\_FILE*} app.jar  
#unpackage jar file  
RUN mkdir -p target/dependency && (cd target/dependency; jar -xf /app.jar)  
#stage 2  
#Same Java runtime (Java 17)  
FROM openjdk:17-slim  
#Add volume pointing to /tmp  
VOLUME /tmp  
#Copy unpackaged application to new container  
ARG *DEPENDENCY*=/target/dependency  
COPY --from=*build* ${*DEPENDENCY*}/BOOT-INF/lib /app/lib  
COPY --from=*build* ${*DEPENDENCY*}/META-INF /app/META-INF  
COPY --from=*build* ${*DEPENDENCY*}/BOOT-INF/classes /app  
#execute the application  
ENTRYPOINT ["java","-cp","app:app/lib/\*","com.example.books.BooksApplication"]

Con los dias revisaremos si cada microservicio requerirá en un futuro en cloud su propio Docker compose ….

**Instalación y Configuración Entorno Desarrollo localmente**

usar H2 como lo recomienda el capability pero si se sienten limitados pueden usar la base de datos que levantaremos en local una mysql

dentro del directiorio Docker están dos ambientes local y prod

en este archivo esta declarado como se levantan los servicios

local.docker-compose.yml

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Para levantar esto requerimos

Docker desktop en Windows

Git bash

:$ export ENVIRONMENT=local

Abrir build.sh en jetbrains IDE habilitar que pueda correr scripts y darle run.

Sobre este repositorio pueden crear sus propios servicios con springboot y tomando las guias anteriores paso a paso veremos lo de conectar eureka para verlo en acción.

Verán estos sitios localmente al final de levantarse cada uno de los contenedores

Sonarqube   
http://localhost:9000/

A screenshot of a login page

Description automatically generated  
prometheus  
<http://localhost:9090/query>  
A screenshot of a computer

Description automatically generated

Grafana

http://localhost:3000/login

A screenshot of a computer

Description automatically generated

Kibana

<http://localhost:5601>

A screenshot of a computer

Description automatically generated