(/)

Metodología de doce factores en un microservicio Spring Boot

Última modificación: 19 de septiembre de 2019

por Kumar Chandrakant (https://www.baeldung.com/author/kumar-chandrakant/)
(https://www.baeldung.com/author/kumar-chandrakant/)

Arquitectura (https://www.baeldung.com/category/architecture/)
DESCANSO (https://www.baeldung.com/category/rest/)
Spring Boot (https://www.baeldung.com/category/spring/spring-boot/)

1. Información general

En este tutorial, entenderemos la metodología de la aplicación de doce factores (https://12factor.net/) .

También entenderemos cómo desarrollar un microservicio con la ayuda de Spring Boot. En el proceso, veremos cómo aplicar la metodología de doce factores para desarrollar dicho microservicio.

2. ¿Cuál es la metodología de doce factores?

La metodología de doce factores es un conjunto de doce mejores prácticas para desarrollar aplicaciones desarrolladas para ejecutarse como un servicio . Originalmente, esto fue redactado por Heroku para aplicaciones implementadas como servicios en su plataforma en la nube, en 2011. Con el tiempo, esto ha demostrado ser lo suficientemente genérico para cualquier desarrollo de software como servicio (https://en.wikipedia.org/wiki/Software_as_a_service) (SaaS).

Entonces, ¿qué entendemos por software como servicio? Tradicionalmente, diseñamos, desarrollamos, implementamos y mantenemos soluciones de software para obtener valor comercial de él. Pero, no tenemos que participar en este proceso para lograr el mismo resultado necesariamente. Por ejemplo, calcular el impuesto aplicable es una función genérica en muchos dominios.

Ahora, podemos decidir construir y administrar este servicio nosotros mismos o **suscribirnos a una oferta de servicio comercial** . Dichas **ofertas de servicios son lo que conocemos como software como servicio** .

Si bien el software como servicio no impone ninguna restricción en la arquitectura en la que se desarrolla; es bastante útil adoptar algunas mejores prácticas.

Si diseñamos nuestro software para que sea modular, portátil y escalable en las plataformas modernas de la nube, es bastante adecuado para nuestras ofertas de servicios. Aquí es donde ayuda la metodología de los doce factores. Los veremos en acción más adelante en el tutorial.

3. Microservicio con Spring Boot

El microservicio (https://www.baeldung.com/spring-microservices-guide) es un estilo arquitectónico para desarrollar software como servicios acoplados libremente. El requisito clave aquí es que los **servicios deben organizarse en torno a los límites del dominio empresarial** . Esta es a menudo la parte más difícil de identificar.

Además, un servicio aquí tiene la autoridad exclusiva sobre sus datos y expone las operaciones a otros servicios. La comunicación entre servicios generalmente se realiza a través de protocolos ligeros como HTTP. Esto da como resultado servicios desplegables y escalables de forma independiente.

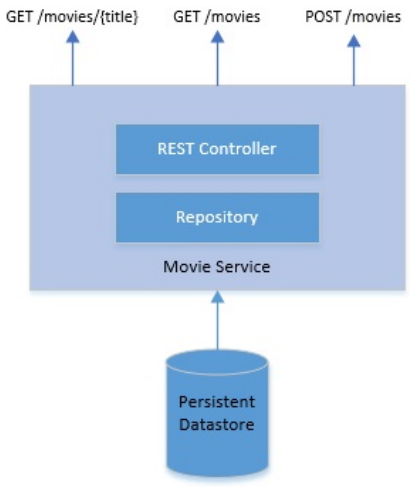
Ahora, la arquitectura de microservicios y el software como servicio no dependen unos de otros. Pero no es difícil entender que, al **desarrollar software como servicio, aprovechar la arquitectura del microservicio es bastante beneficioso** . Ayuda a lograr muchos objetivos que discutimos anteriormente, como la modularidad y la escalabilidad.

Spring Boot (https://spring.io/projects/spring-boot) es un marco de aplicación basado en Spring que elimina muchas repeticiones asociadas con el desarrollo de una aplicación empresarial. Nos da una plataforma altamente obstinada pero flexible para desarrollar microservicios. Para este tutorial, aprovecharemos Spring Boot para entregar un microservicio utilizando la metodología de doce factores.

4. Aplicación de la metodología de doce factores

Ahora definamos una aplicación simple que intentaremos desarrollar con las herramientas y prácticas que acabamos de discutir. A todos nos encanta ver películas, pero es difícil hacer un seguimiento de las películas que ya hemos visto.

Ahora, ¿a quién le gustaría comenzar una película y luego abandonarla más tarde? Lo que necesitamos es un servicio simple para grabar y consultar películas que hemos visto:



(http://inprogress.baeldung.com/wp-content/uploads/2019/08/12-factpr-app.jpg)
Este es un microservicio bastante simple y estándar con un almacén de datos y puntos finales REST.
Necesitamos definir un modelo que también se mapee a la persistencia:

```
@Entity
public class Movie {
    @Id
    private Long id;
    private String title;
    private String year;
    private String rating;
    // getters and setters
}
```

Hemos definido una entidad JPA con una identificación y algunos otros atributos. Veamos ahora cómo se ve el controlador REST:

```
@RestController
public class MovieController {

    @Autowired
    private MovieRepository movieRepository;
    @GetMapping("/movies")
    public List<Movie> retrieveAllStudents() {
        return movieRepository.findAll();
    }

    @GetMapping("/movies/{id}")
    public Movie retrieveStudent(@PathVariable Long id) {
        return movieRepository.findById(id).get();
    }

    @PostMapping("/movies")
    public Long createStudent(@RequestBody Movie movie) {
        return movieRepository.save(movie).getId();
    }
}
```

Esto cubre la base de nuestro servicio simple. Revisaremos el resto de la aplicación mientras discutimos cómo implementamos la metodología de doce factores en las siguientes subsecciones.

4.1. Codebase

La primera mejor práctica de las aplicaciones de doce factores es rastrearla en un sistema de control de versiones. Git (https://git-scm.com/) es el sistema de control de versiones más popular en uso hoy en día y es casi omnipresente. El principio establece que **una aplicación debe rastrearse en un único repositorio de código y no debe compartir ese repositorio con ninguna otra aplicación** .

Spring Boot ofrece muchas maneras convenientes de iniciar una aplicación, incluida una herramienta de línea de comandos y una interfaz web (https://start.spring.io/) . Una vez que generamos la aplicación bootstrap, podemos convertir esto en un repositorio git:

```
git init
```

Este comando debe ejecutarse desde la raíz de la aplicación. La aplicación en esta etapa ya contiene un archivo .gitignore que efectivamente restringe el control de la versión de los archivos generados. Entonces podemos crear de inmediato una confirmación inicial:

```
git add .
git commit -m "Adding the bootstrap of the application."
```

Finalmente, podemos agregar un control remoto y enviar nuestras confirmaciones al control remoto si lo deseamos (esto no es un requisito estricto):

```
git remote add origin https://github.com/<username (https://github.com/<username)>/12-factor-app.git
git push -u origin master
```

4.2. Dependencias

A continuación, la **aplicación de doce factores siempre debe declarar explícitamente todas sus dependencias** . Deberíamos hacer esto usando un manifiesto de declaración de dependencia. Java tiene múltiples herramientas de administración de dependencias como Maven y Gradle. Podemos usar uno de ellos para lograr este objetivo.

Por lo tanto, nuestra aplicación simple depende de algunas bibliotecas externas, como una biblioteca para facilitar las API REST y para conectarse a una base de datos. Veamos cómo podemos definirlos declarativamente usando Maven.

Maven nos exige que describamos las dependencias de un proyecto en un archivo XML, generalmente conocido como Modelo de objetos de proyecto (https://maven.apache.org/guides/introduction/introduction-to-the-pom.html) (POM):

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Aunque esto parece simple y simple, estas dependencias generalmente tienen otras dependencias transitivas. Esto lo complica en cierta medida, pero nos ayuda a lograr nuestro objetivo. Ahora, nuestra aplicación no tiene una dependencia directa que no se describe explícitamente.

4.3. Configuraciones

Una aplicación generalmente tiene mucha configuración, algunas de las cuales pueden variar entre implementaciones, mientras que otras siguen siendo las mismas.

En nuestro ejemplo, tenemos una base de datos persistente. Necesitaremos la dirección y las credenciales de la base de datos para conectarnos. Es más probable que esto cambie entre implementaciones.

Una aplicación de doce factores debería externalizar todas esas configuraciones que varían entre implementaciones . La recomendación aquí es usar variables de entorno para tales configuraciones. Esto conduce a una separación limpia de configuración y código.

Spring proporciona un archivo de configuración donde podemos declarar tales configuraciones y adjuntarlo a las variables de entorno:

```
spring.datasource.url=jdbc:mysql:// (mysql://)${MYSQL_HOST}:${MYSQL_PORT}/movies
spring.datasource.username=${MYSQL_USER}
spring.datasource.password=${MYSQL_PASSWORD}
```

Aquí, hemos definido la URL y las credenciales de la base de datos como configuraciones y hemos mapeado los valores reales que se elegirán de la variable de entorno.

En Windows, podemos establecer la variable de entorno antes de iniciar la aplicación:

```
set MYSQL_HOST=localhost
set MYSQL_PORT=3306
set MYSQL_USER=movies
set MYSQL_PASSWORD=password
```

Podemos utilizar una herramienta de gestión de configuración como Ansible (https://www.ansible.com/) o Chef (https://www.chef.io/) para automatizar este proceso.

4.4. Servicios de respaldo

Los servicios de respaldo son servicios de los que depende la aplicación para su funcionamiento. Por ejemplo, una base de datos o un agente de mensajes. **Una aplicación de doce factores debe tratar todos los servicios de respaldo como recursos adjuntos.** Lo que esto significa efectivamente es que no debería requerir ningún cambio de código para intercambiar un servicio de respaldo compatible. El único cambio debe estar en las configuraciones.

En nuestra aplicación, hemos utilizado MySQL (https://www.mysql.com/) como servicio de respaldo para proporcionar persistencia.

Spring JPA (https://www.baeldung.com/the-persistence-layer-with-spring-and-jpa) hace que el código sea bastante independiente del proveedor de la base de datos real. Solo necesitamos definir un repositorio que proporcione todas las operaciones estándar:

```
@Repository
public interface MovieRepository extends JpaRepository<Movie, Long> {
}
```

Como podemos ver, esto no depende de MySQL directamente. Spring detecta el controlador MySQL en el classpath y proporciona una implementación específica de MySQL de esta interfaz de forma dinámica. Además, extrae otros detalles de las configuraciones directamente.

Entonces, si tenemos que cambiar de MySQL a Oracle, todo lo que tenemos que hacer es reemplazar el controlador en nuestras dependencias y reemplazar las configuraciones.

4.5. Construir, liberar y ejecutar

La metodología de doce factores **separa estrictamente el proceso de convertir la base de código en una aplicación en ejecución** en tres etapas distintas:

- Etapa de compilación: aquí es donde tomamos la base de código, realizamos comprobaciones estáticas y dinámicas, y luego generamos un paquete ejecutable como un JAR. Usando una herramienta como **Maven (https://maven.apache.org/)** , esto es bastante trivial:

```
mvn clean compile test package
```

- Etapa de lanzamiento: esta es la etapa en la que tomamos el paquete ejecutable y lo combinamos con las configuraciones correctas. Aquí, podemos usar **Packer (https://www.packer.io/)** con un aprovisionador como **Ansible (https://www.ansible.com/)** para crear imágenes de Docker:

```
packer build application.json
```

- Etapa de ejecución: Finalmente, esta es la etapa en la que ejecutamos la aplicación en un entorno de ejecución de destino. Si usamos **Docker (https://www.docker.com/)** como contenedor para lanzar nuestra aplicación, ejecutar la aplicación puede ser bastante simple:


```
docker run --name <container_id> -it <image_id>
```

Finalmente, no necesariamente tenemos que realizar estas etapas manualmente. Aquí es donde Jenkins (https://jenkins.io/) resulta bastante útil con su canal declarativo.

4.6. Procesos

Se espera que una aplicación de doce factores se ejecute en un entorno de ejecución como procesos sin estado. En otras palabras, no pueden almacenar el estado persistente localmente entre solicitudes. Pueden generar datos persistentes que deben almacenarse en uno o más servicios de respaldo con estado.

En el caso de nuestro ejemplo, tenemos múltiples puntos finales expuestos. Una solicitud en cualquiera de estos puntos finales es completamente independiente de cualquier solicitud realizada antes. Por ejemplo, si hacemos un seguimiento de las solicitudes de los usuarios en la memoria y usamos esa información para atender solicitudes futuras, viola una aplicación de doce factores.

Por lo tanto, una aplicación de doce factores no impone tal restricción como las sesiones fijas. Esto hace que dicha aplicación sea altamente portátil y escalable. En un entorno de ejecución en la nube que ofrece escalado automatizado, es un comportamiento bastante deseable de las aplicaciones.

4.7. Enlace de puerto

Una aplicación web tradicional en Java se desarrolla como WAR o archivo web. Esta es típicamente una colección de Servlets con dependencias, y espera un tiempo de ejecución de contenedor conforme como Tomcat. **Una aplicación de doce factores, por el contrario, no espera tal dependencia del tiempo de ejecución.** Es completamente autónomo y solo requiere un tiempo de ejecución de ejecución como Java.

En nuestro caso, hemos desarrollado una aplicación usando Spring Boot. Spring Boot, además de muchos otros beneficios, nos proporciona un servidor de aplicaciones integrado predeterminado. Por lo tanto, el JAR que generamos anteriormente usando Maven es totalmente capaz de ejecutarse en cualquier entorno simplemente con un tiempo de ejecución Java compatible:

```
1 java -jar application.jar
```

Aquí, nuestra aplicación simple expone sus puntos finales sobre un enlace HTTP a un puerto específico como 8080. Al iniciar la aplicación como lo hicimos anteriormente, debería ser posible acceder a los servicios exportados como HTTP.

Una aplicación puede exportar múltiples servicios como FTP o WebSocket (https://www.baeldung.com/websockets-spring) mediante la vinculación a múltiples puertos.

4.8. Concurrencia

Java ofrece *Thread* como un modelo clásico para manejar la concurrencia en una aplicación. Los subprocesos son como procesos ligeros y representan múltiples rutas de ejecución en un programa. Los subprocesos son potentes pero tienen limitaciones en cuanto a cuánto puede ayudar a una aplicación a escalar.

La metodología de doce factores sugiere que las aplicaciones se basen en procesos para escalar. Lo que esto significa efectivamente es que las aplicaciones deben estar diseñadas para distribuir la carga de trabajo en múltiples procesos. Sin embargo, los procesos individuales son libres de aprovechar internamente un modelo de concurrencia como *Thread* .

Una aplicación Java, cuando se inicia, obtiene un único proceso que está vinculado a la JVM subyacente. Lo que realmente necesitamos es una forma de lanzar múltiples instancias de la aplicación con distribución inteligente de carga entre ellas. Como ya hemos empaquetado nuestra aplicación como un contenedor Docker (https://www.baeldung.com/docker-java-api) , Kubernetes (https://www.baeldung.com/kubernetes) es una opción natural para dicha orquestación.

4.9. Desechabilidad

Los procesos de aplicación se pueden cerrar a propósito o mediante un evento inesperado. En cualquier caso, **se supone que una aplicación de doce factores lo maneja con gracia** . En otras palabras, un proceso de solicitud debe ser completamente desechable sin efectos secundarios no deseados. Además, los procesos deberían comenzar rápidamente

Por ejemplo, en nuestra aplicación, uno de los puntos finales es crear un nuevo registro de base de datos para una película. Ahora, una aplicación que maneja dicha solicitud puede bloquearse inesperadamente. Sin embargo, esto no debería afectar el estado de la aplicación. Cuando un cliente envía la misma solicitud nuevamente, no debería generar registros duplicados.

En resumen, la aplicación debe exponer servicios idempotentes. Este es otro atributo muy deseable de un servicio destinado a implementaciones en la nube. Esto brinda la flexibilidad de detener, mover o girar nuevos servicios en cualquier momento sin ninguna otra consideración.

4.10. Dev / Prod Parity

Es típico que las aplicaciones se desarrollen en máquinas locales, se prueben en otros entornos y finalmente se implementen en producción. A menudo es el caso donde estos entornos son diferentes. Por ejemplo, el equipo de desarrollo trabaja en máquinas con Windows, mientras que la implementación de producción ocurre en máquinas con Linux.

La metodología de doce factores sugiere mantener la brecha entre el entorno de desarrollo y producción lo más mínima posible. Estas brechas pueden ser el resultado de largos ciclos de desarrollo, diferentes equipos involucrados o diferentes tipos de tecnología en uso.

Ahora, la tecnología como Spring Boot y Docker automáticamente cierran esta brecha en gran medida. Se espera que una aplicación en contenedor se comporte igual, sin importar dónde la ejecutemos. Debemos usar los mismos servicios de respaldo, como la base de datos, también.

Además, deberíamos tener los procesos correctos, como la integración continua y la entrega para facilitar aún más esta brecha.

4.11. Registros

Los registros son datos esenciales que genera una aplicación durante su vida útil. Proporcionan información invaluable sobre el funcionamiento de la aplicación. Típicamente, una aplicación puede generar registros en múltiples niveles con diferentes detalles y resultados ii en múltiples formatos diferentes.

Sin embargo, una aplicación de doce factores se separa de la generación de registros y su procesamiento. **Para una aplicación de este tipo, los registros no son más que una secuencia de eventos ordenada por tiempo.** Simplemente escribe estos eventos en la salida estándar del entorno de ejecución. La captura, el almacenamiento, la conservación y el archivo de dicha secuencia deben ser manejados por el entorno de ejecución.

Tenemos bastantes herramientas disponibles para este propósito. Para empezar, podemos usar SLF4J (https://www.baeldung.com/slf4j-with-log4j2-logback) para manejar el registro de forma abstracta dentro de nuestra aplicación. Además, podemos usar una herramienta como Fluentd (https://www.fluentd.org/) para recopilar el flujo de registros de aplicaciones y servicios de respaldo.

Esto lo podemos alimentar a Elasticsearch (https://www.elastic.co/) para almacenamiento e indexación. Finalmente, podemos generar paneles significativos para la visualización en Kibana (https://www.elastic.co/products/kibana) .

4.12. Procesos administrativos

A menudo necesitamos realizar algunas tareas puntuales o procedimientos de rutina con nuestro estado de aplicación. Por ejemplo, arreglando malos registros. Ahora, hay varias formas en que podemos lograr esto. Como a menudo no lo requerimos, podemos escribir un pequeño script para ejecutarlo por separado de otro entorno.

Ahora, **la metodología de doce factores sugiere encarecidamente mantener dichos scripts de administración junto con la base de código de la aplicación** . Al hacerlo, debe seguir los mismos principios que aplicamos a la base de código de la aplicación principal. También es aconsejable utilizar una herramienta REPL incorporada del entorno de ejecución para ejecutar dichos scripts en servidores de producción.

En nuestro ejemplo, ¿cómo podemos sembrar nuestra aplicación con las películas ya vistas hasta ahora? Si bien podemos usar nuestro pequeño punto final dulce, pero eso puede parecer poco práctico. Lo que necesitamos es un script para realizar una carga única. Podemos escribir una pequeña función Java para leer una lista de películas de un archivo y guardarlas en lotes en la base de datos.

Además, podemos usar Groovy integrado con Java (https://www.baeldung.com/groovy-java-applications) Runtime para iniciar dichos procesos.

5. Aplicaciones prácticas

Entonces, ahora hemos visto todos los factores sugeridos por la metodología de doce factores. Desarrollar una aplicación para que sea una aplicación de **doce factores ciertamente tiene sus beneficios, especialmente cuando deseamos implementarlos como servicios en la nube** . Pero, como todas las demás pautas, marco, patrones, debemos preguntarnos, ¿es esto una bala de plata?

Honestamente, ninguna metodología única en diseño y desarrollo de software afirma ser una bala de plata. La metodología de doce factores no es una excepción. Si bien **algunos de estos factores son bastante intuitivos** , y lo más probable es que ya los estemos haciendo, **otros pueden no aplicarse a nosotros** . Es esencial evaluar estos factores en el contexto de nuestros objetivos y luego elegir sabiamente.

Es importante tener en cuenta que todos estos factores están **ahí para ayudarnos a desarrollar una aplicación que sea modular, independiente, portátil, escalable y observable** . Dependiendo de la aplicación, podemos lograrlos a través de otros medios mejor. Tampoco es necesario adoptar todos los factores juntos, adoptar incluso algunos de estos puede hacernos mejores de lo que éramos.

Finalmente, estos factores son bastante simples y elegantes. Tienen una mayor importancia en una época en la que exigimos que nuestras aplicaciones tengan un mayor rendimiento y una menor latencia prácticamente sin tiempo de inactividad ni fallas. **Adoptar estos factores nos da el comienzo correcto desde el principio.** Combinados con la arquitectura de microservicios y la contenedorización de aplicaciones, parecen dar en el clavo.

6. Conclusión

En este tutorial, revisamos los conceptos de la metodología de doce factores. Discutimos cómo aprovechar una arquitectura de microservicios con Spring Boot para entregarlos de manera efectiva. Además, exploramos cada factor en detalle y cómo aplicarlos a nuestra aplicación. También exploramos varias herramientas para aplicar estos factores individuales de manera efectiva con éxito.

¡Los comentarios están cerrados en este artículo!

CATEGORÍAS

PRIMAVERA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
DESCANSO (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SEGURIDAD (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCIA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP DEL LADO DEL CLIENTE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIE

TUTORIAL DE SPRING BOOT (/SPRING-BOOT-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)

HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
RESTO CON SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
TUTORIAL SPRING PERSISTENCE (/PERSISTENCE-WITH-SPRING-SERIES)
SEGURIDAD CON PRIMAVERA (/SECURITY-SPRING)

ACERCA DE

SOBRE BAELDUNG (/ABOUT)
LOS CURSOS (HTTPS://COURSES.BAELDUNG.COM)
TRABAJO DE CONSULTORÍA (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
EL ARCHIVO COMPLETO (/FULL_ARCHIVE)
ESCRIBIR PARA BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORES (/EDITORS)
NUESTROS COMPAÑEROS (/PARTNERS)
ANUNCIE EN BAELDUNG (/ADVERTISE)

TÉRMINOS DE SERVICIO (/TERMS-OF-SERVICE)
POLÍTICA DE PRIVACIDAD (/PRIVACY-POLICY)
INFORMACIÓN DE LA COMPAÑÍA (/BAELDUNG-COMPANY-INFO)
CONTACTO (/CONTACT)