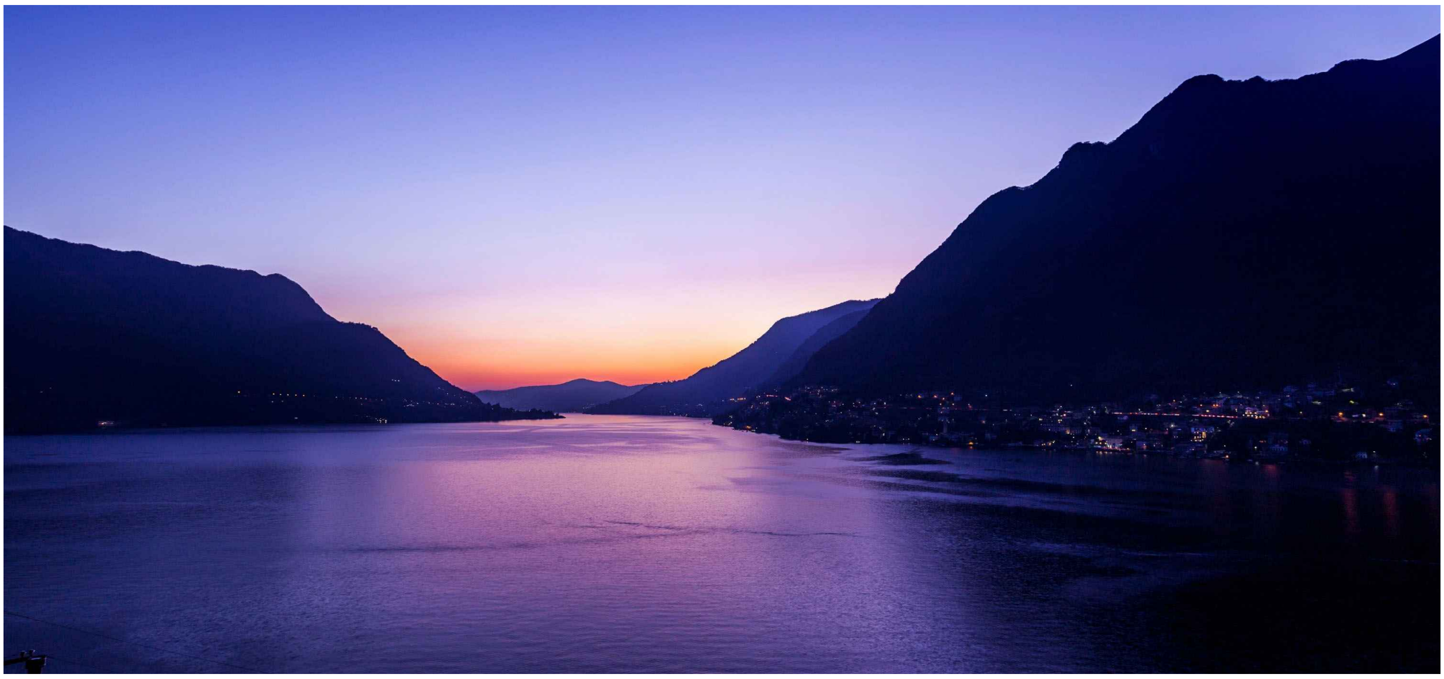


22-Spring Boot Actuator

Author: Vincent Lau

Learning Objectives

> Actuator brings production-ready features to our application

> Monitoring our app, gathering metrics, understanding traffic, or the state of our database become trivial with this dependency

> The main benefit of this library is that we can get production-grade tools without having to actually implement these features ourselves

> Mainly used to expose operational information about the running application — health, metrics, info, dump, env, etc

Configuration

Maven Dependencies

Add the following dependency in pom.xml

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

application.yml

- You must define the connection attributes in the `application.yml` file.

```yaml
#這樣寫的話，原本內建的/actuator/xxx路徑，都會變成/data/xxx，可以用來防止被其他人猜到
management.endpoints.web.base-path: /data

# 可以這樣寫，就會開啟所有endpoints(不包含shutdown)
management.endpoints.web.exposure.include: "*"

# 如果要開啟/actuator/shutdown，要額外再加這一行
management.endpoint.shutdown.enabled: true

# exclude可以用來關閉某些endpoints
# exclude通常會跟include一起用，就是先include了全部，然後再exclude /actuator/beans這個endpoint
management.endpoints.web.exposure.exclude: beans

# 也可以這樣寫，就只會開啟指定的endpoint，因此此處只會再額外開啟/actuator/beans和/actuator/mappings
web.exposure.include: beans,mappings

# now we can access more details from endpoint /metrics/health/readiness
management.endpoint.health.group.readiness.show-details: always
```

Actuator build-in Endpoints

1. **auditevents**: Exposes information about audited events in the application, typically related to security auditing.
2. **beans**: Provides a list of all spring beans in the application context.
3. **caches**: Displays information about the configured cache manager, including cache statistics.
4. **conditions**: Shows the conditions that were evaluated when the application context was created.
5. **configprops**: Lists all configuration properties and their current values.

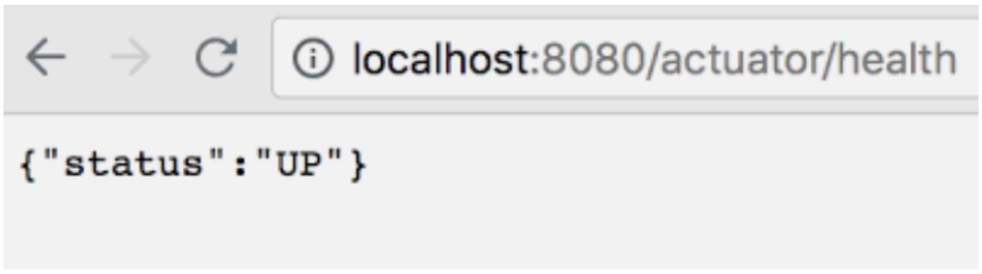
6. **env**: Displays the application's environmental properties, including configuration properties and system information.
7. **flyway**: Shows details about database migrations managed by Flyway.
8. **health**: Provides health information about the application and its dependencies, such as databases and message brokers.
9. **httpexchanges**: Exposes information about HTTP exchanges handled by the application.
10. **info**: Displays custom application information, such as name, description, version, and user-defined details.
11. **integrationgraph**: Shows the integration graph of Spring Integration applications.
12. **loggers**: Allows you to view and modify the application's logging levels for different loggers.
13. **liquibase**: Shows details about database changesets managed by Liquibase.
14. **metrics**: Exposes application metrics, including system metrics and custom application-specific metrics.
15. **mappings**: Lists all HTTP request mappings and their details, including the associated controller methods.
16. **quartz**: Provides information about Quartz Scheduler jobs and triggers.
17. **scheduledtasks**: Displays details about scheduled tasks and their execution status.
18. **sessions**: Allows you to inspect and manage user sessions, if the application uses Spring Session.
19. **shutdown**: Enables you to trigger a graceful application shutdown.
20. **startup**: Provides information about the application's startup, including the startup date and uptime.
21. **threaddump**: Generates a thread dump, showing the status of all threads in the application.
22. **heapdump**: Generates a binary heap dump of the application's Java virtual machine (JVM) memory. A heap dump is a snapshot of the JVM's memory, including the objects and their relationships. It's useful for diagnosing memory-related issues, such as memory leaks. You can use tools like VisualVM or Eclipse MAT to analyze the heap dump.
23. **logfile**: Provides access to the application's log file. You can use this endpoint to view and download log files. It's particularly useful for troubleshooting and analyzing application logs to identify issues or monitor application behavior.

24. **prometheus**: Exposes metrics in a format that can be scraped by Prometheus, a popular monitoring and alerting toolkit. It allows you to collect and store application metrics over time and set up alerting rules based on these metrics. When integrated with Prometheus and Grafana, you can gain deep insights into the performance and behavior of your Spring Boot application.

These Actuator endpoints provide valuable insights into your Spring Boot application's health, configuration, and runtime behavior, making them essential for monitoring and managing your applications in a production environment.

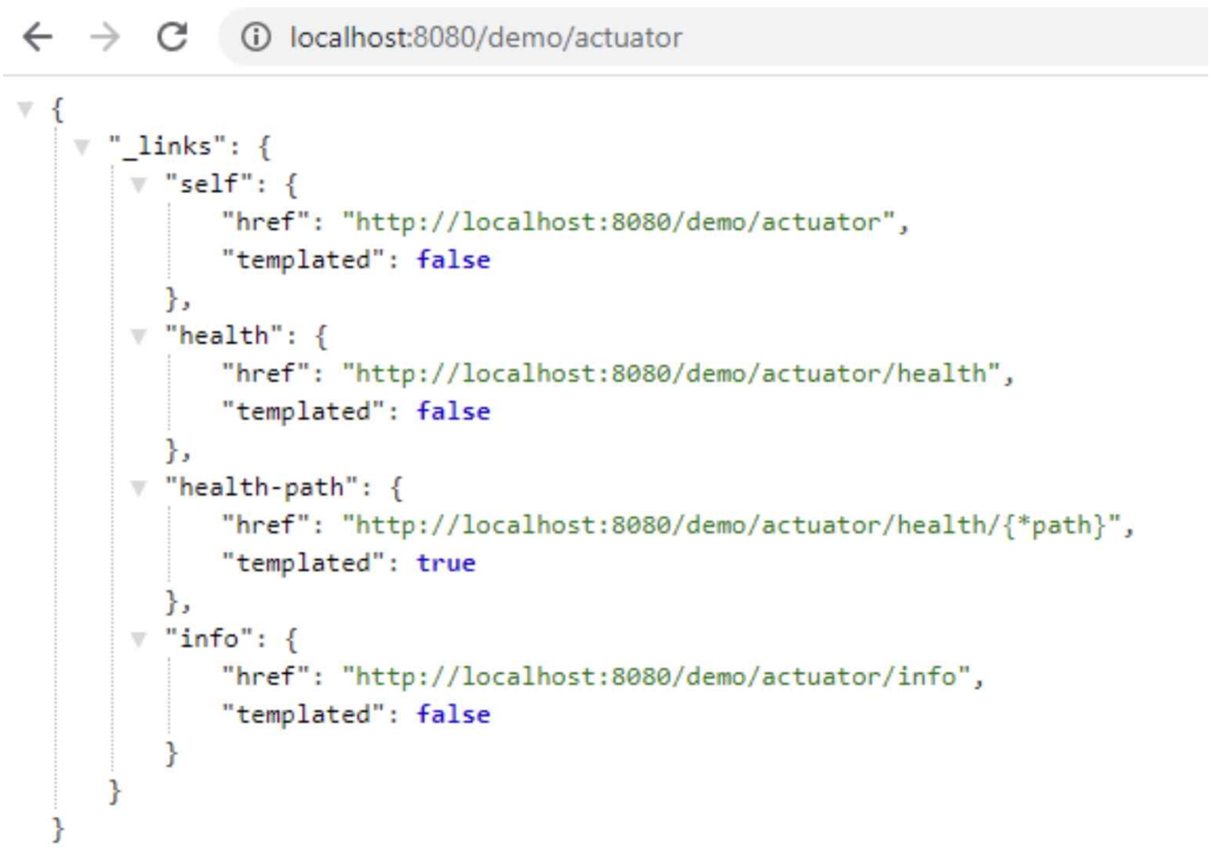
Endpoints Result Examples

1. /actuator/health



```
{"status":"UP"}
```

2. /actuator



```
{
  "_links": {
    "self": {
      "href": "http://localhost:8080/demo/actuator",
      "templated": false
    },
    "health": {
      "href": "http://localhost:8080/demo/actuator/health",
      "templated": false
    },
    "health-path": {
      "href": "http://localhost:8080/demo/actuator/health/{*path}",
      "templated": true
    },
    "info": {
      "href": "http://localhost:8080/demo/actuator/info",
      "templated": false
    }
  }
}
```

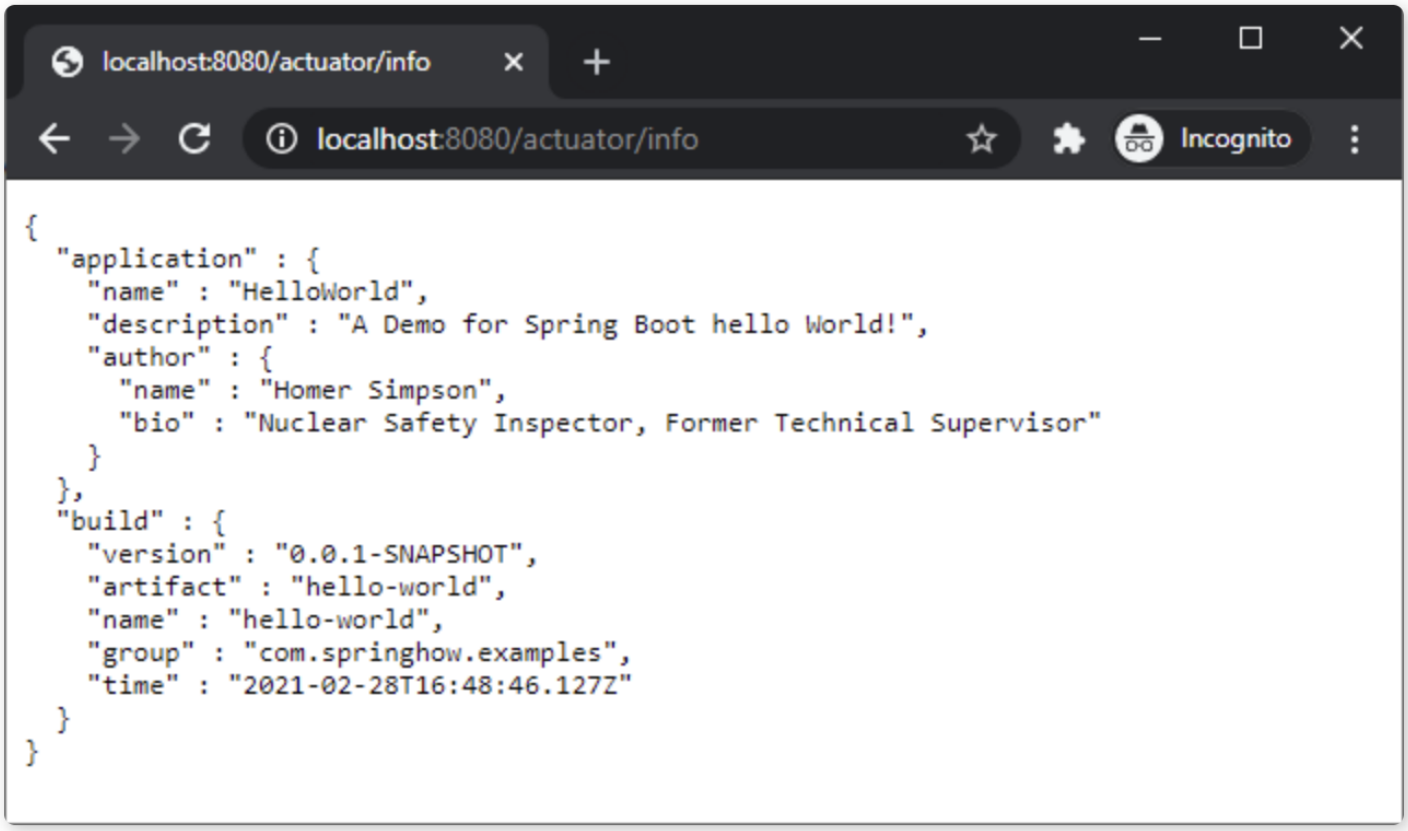
Build info

If you want to add the build version to the info endpoint, then the application.properties won’ t cut it. Also, changing the application.properties on each build takes effort. Spring boot handles this by loading data from **META-INF/build-info.properties** file if it is present in the JAR file.

To create the build-info.properties file, you should first change the **spring-boot-maven-plugin** settings as shown below.

```xml
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <executions>
        <execution>
            <goals>
                <goal>build-info</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Here, the **build-info** goal will generate the **META-INF/build-info.properties**.



```
{
  "application" : {
    "name" : "HelloWorld",
    "description" : "A Demo for Spring Boot hello World!",
    "author" : {
      "name" : "Homer Simpson",
      "bio" : "Nuclear Safety Inspector, Former Technical Supervisor"
    }
  },
  "build" : {
    "version" : "0.0.1-SNAPSHOT",
    "artifact" : "hello-world",
    "name" : "hello-world",
    "group" : "com.springhow.examples",
    "time" : "2021-02-28T16:48:46.127Z"
  }
}
```

build information in actuator endpoint

Application Info

Also known as Environment info, you can pass info to this endpoint directly from application configuration files. And here is a straightforward example.

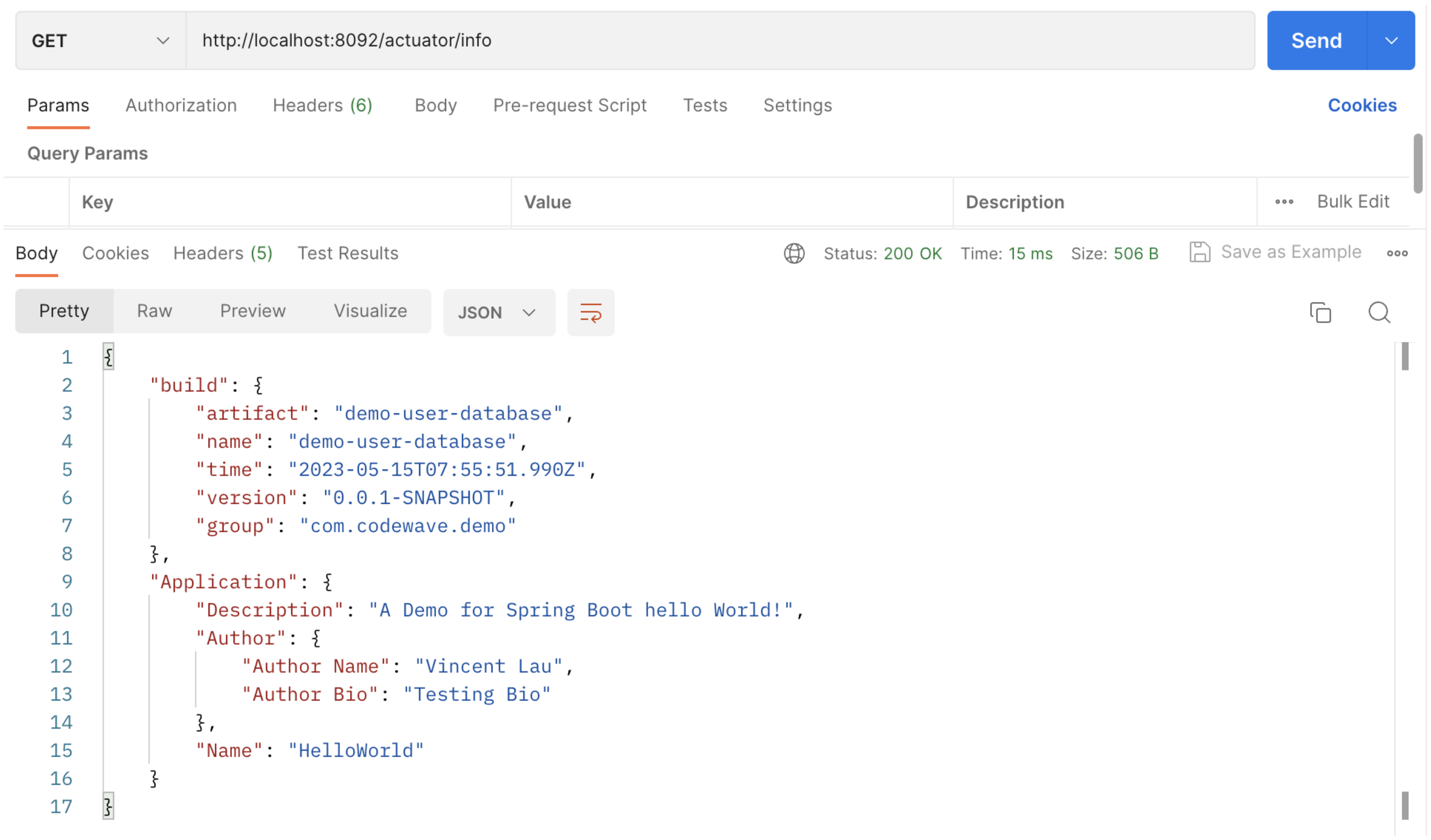
```yaml
info.application:
  name: HelloWorld
  description: A Demo for Spring Boot hello World!
  author:
    name: Vincent Lau
    bio: Testing Bio
```

```java
@Component
public class CustomInfoContributor implements InfoContributor {

    @Autowired
    private Environment env;

    private static final Map<String, Object> application = new HashMap<>();
    private static final Map<String, Object> author = new HashMap<>();

    @Override
    public void contribute(Info.Builder builder) {
        application.put("Name", env.getProperty("info.application.name"));
        application.put("Description", env.getProperty("info.application.description"));
        author.put("Author Name", env.getProperty("info.application.author.name"));
        author.put("Author Bio", env.getProperty("info.application.author.bio"));
        application.put("Author", author);
        builder.withDetail("Application", application);
    }
}
```



http://localhost:8092/actuator/info

Status: 200 OK Time: 15 ms Size: 506 B

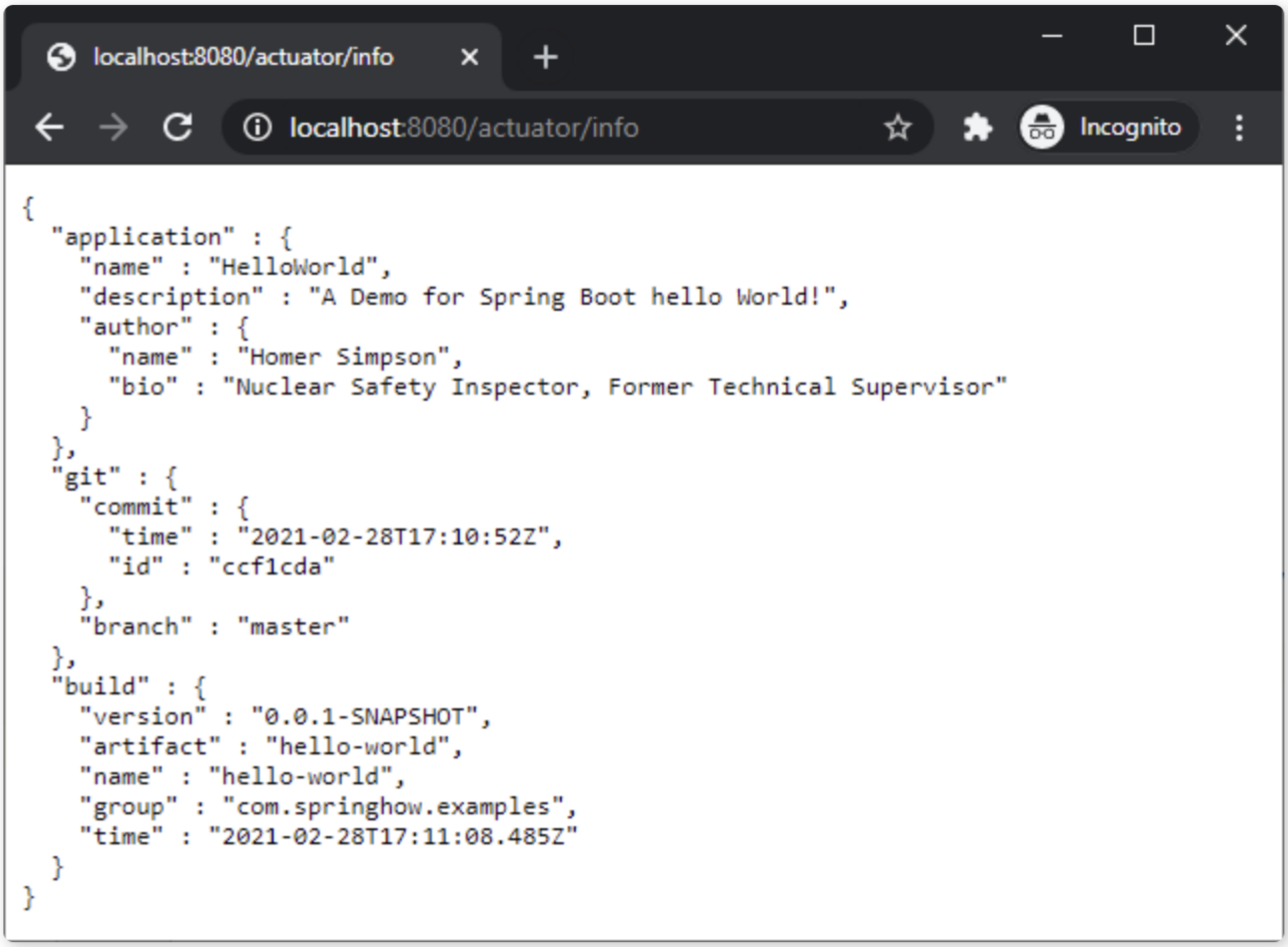
```
{
    "build": {
        "artifact": "demo-user-database",
        "name": "demo-user-database",
        "time": "2023-05-15T07:55:51.990Z",
        "version": "0.0.1-SNAPSHOT",
        "group": "com.codewave.demo"
    },
    "Application": {
        "Description": "A Demo for Spring Boot hello World!",
        "Author": {
            "Author Name": "Vincent Lau",
            "Author Bio": "Testing Bio"
        },
        "Name": "HelloWorld"
    }
}
```

GIT Commit info

To generate git related data, we need to use **git-commit-id-plugin**. This is how you should configure the plugin.

```
<plugin>
    <groupId>pl.project13.maven</groupId>
    <artifactId>git-commit-id-plugin</artifactId>
    <configuration>
        <failOnNoGitDirectory>false</failOnNoGitDirectory>
    </configuration>
</plugin>
```

This plugin will create a **git.properties** file under classpath.



```
{
  "application" : {
    "name" : "HelloWorld",
    "description" : "A Demo for Spring Boot hello World!",
    "author" : {
      "name" : "Homer Simpson",
      "bio" : "Nuclear Safety Inspector, Former Technical Supervisor"
    }
  },
  "git" : {
    "commit" : {
      "time" : "2021-02-28T17:10:52Z",
      "id" : "ccf1cda"
    },
    "branch" : "master"
  },
  "build" : {
    "version" : "0.0.1-SNAPSHOT",
    "artifact" : "hello-world",
    "name" : "hello-world",
    "group" : "com.springhow.examples",
    "time" : "2021-02-28T17:11:08.485Z"
  }
}
```

As you see in this screenshot, the info endpoint now includes VCS info.

Advanced Setting: Health Group

Try to self-study for grouping. You can refer to actuator.endpoints.health.groups from doc.spring.io.