UDC 004.41

APPLICATION MONITORING WITH SPRING BOOT ACTUATOR

E. BOBROVICH, A. OSKIN
Polotsk State University, Belarus

The article introduces a way to monitor and controls Spring Boot applications using Actuator. It describes the functionality of this tool, settings and integration with external application monitoring systems.

After the application is developed and deployed in a working environment, it is very important to monitor its performance. Enterprise applications are changing and becoming multi-level, distributed between different servers or even continents, moving to the clouds. Therefore, such complex distributed applications require control, as in some companies they are the basis of business. Application performance monitoring solves the tasks of monitoring, managing the availability and directly application performance.

Spring Boot Actuator is a subproject in the Spring Boot project. The Spring Boot Actuator module helps you monitor and manage your Spring Boot application by providing ready-to-use features such as health checks, audits, metrics collection, HTTP tracing, etc.

Actuator also integrates with external application monitoring systems such as Prometheus, Graphite, DataDog, Influx, Wavefront, New Relic and many others. These systems provide excellent dashboards, graphs, analytics and alarms to help you monitor and manage your application from a single interface. Actuator uses Micrometer, a facade of application metrics, to integrate with these external application monitoring systems. This makes it very easy to connect any application monitoring system with minimal settings.

To enable Actuator, you just need to add the dependency to the Spring Boot application package manager. Examples of dependencies for Maven and Gradle are shown in Listings 1 and 2, respectively.

Listing 1 – Add a dependency to a Maven project

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

Listing 2 – Add a dependency to a Gradle project

```
dependencies {
  compile("org.springframework.boot:spring-boot-starter-actuator")
}
```

Actuator creates several so-called endpoints that can be accessed via HTTP or JMX so that you can track and interact with your application. For example, there is a /health endpoint that provides basic application health information. The /metrics endpoint shows some useful metric data, such as used JVM memory, system CPU usage, open files, and more.

By default, all endpoints open through JMX, but only /health and /info are provided through HTTP. To access all endpoints via HTTP, add the management.endpoints.web.exposure.include = * property in the application.properties file. If you need to add access only to some endpoints, then you should list the id of these endpoints in the property value.

A list of available endpoints available via HTTP can be obtained from the base endpoint /actuator. The list of end points is presented in the table.

The /health endpoint shows only a simple UP or DOWN state. For complete information, including the status of each health indicator that was checked during the health check, add the management.endpoint.health.show-details = always property to the application.properties file. After that, /health displays more information, including details of the DiskSpaceHealthIndicator, which starts as part of the application health check process. If a database is connected to your application, then /health will show the current state of connection to it. You can also create your own health indicator by implementing the HealthIndicator interface or by extending the AbstractHealthIndicator class.

Table. –Spring Boot Actuator endpoints and description

ID	Description
auditevents	Exposes audit events information for the current application
beans	Displays a complete list of all the Spring beans in your application
caches	Exposes available caches
conditions	Shows the conditions that were evaluated on configuration and auto-configuration classes and the reasons why they did or did not match
configprops	Displays a collated list of all @ConfigurationProperties
env	Exposes properties from Spring's ConfigurableEnvironment
flyway	Shows any Flyway database migrations that have been applied
health	Shows application health information
httptrace	Displays HTTP trace information (by default, the last 100 HTTP request-response exchanges)
info	Displays arbitrary application info
integrationgraph	Shows the Spring Integration graph
loggers	Shows and modifies the configuration of loggers in the application
liquibase	Shows any Liquibase database migrations that have been applied
metrics	Shows 'metrics' information for the current application
mappings	Displays a collated list of all @RequestMapping paths
scheduledtasks	Displays the scheduled tasks in your application
sessions	Allows retrieval and deletion of user sessions from a Spring Session-backed session store
shutdown	Lets the application be gracefully shutdown. Disabled by default
threaddump	Performs a thread dump

The /metrics endpoint lists all the metrics you can track. To get detailed information about a particular metric, you need to pass the metric name to a URL, like this http://localhost:8080/actuator/metrics/{MetricName}. In response, the details of the metric in JSON format will be received.

The endpoint /loggers displays a list of all loggers configured in the application with the appropriate logging levels. You can also view information about an individual logger by passing its name in the URL, for example: http://localhost:8080/actuator/loggers/{name}. The endpoint /loggers also allows you to change the logging level of the specified logger during application execution. This functionality will really be useful in cases where your application encounters problems with work, and you want to enable DEBUG logging for a while to get more detailed information about the problem.

The endpoint /info displays arbitrary information about your application. Actuator obtains assembly information from the META-INF/build-info.properties file, Git information from the git.properties file. It also displays any information available in the environment properties with the info key. You can add properties with the info key in the application.properties file as in Listing 3.

Listing 3 – Add custom information to Spring Boot Info Actuator

info.app.name=@project.name@
info.app.description=@project.description@
info.app.version=@project.version@
info.app.encoding=@project.build.sourceEncoding@
info.app.java.version=@java.version@

Endpoints Spring Boot Actuator is recommended to protect against unauthorized access. If Spring Security is present in your application, then these endpoints will be protected by default. You can also set your own security settings for them.

Spring Boot Actuator allows you to implement your custom breakpoints. To get the endpoint, you need a bean that can be created using the @Component annotation. To access the endpoint, you must specify the id parameter in the @Endpoint annotation, which will be responsible for the path /actuator/{id}. The annotations @ReadOperation, @WriteOperation, @DeleteOperation process HTTP GET, POST and DELETE, respectively, and execute the methods that apply.

It provided a description of a powerful tool for controlling, monitoring and managing applications - Spring Boot Actuator. This Spring Boot subproject has the ability to flexibly configure, customize, and create custom

endpoints. Actuator can be used with external monitoring systems, which allows the user to work with all metrics and parameters through a single interface

REFERENCES

1. Spring Boot Actuator Web API Documentation [Electronic resource] / Spring Docs – 2020 – Mode of access : https://docs.spring.io/spring-boot/docs/current/actuator-api/html/ – Date of access : 25.02.2020.
2. Spring Boot Actuator: Production-ready features [Electronic resource] / Spring Docs – 2020 – Mode of access : https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html – Date of access : 25.02.2020.