**ACTUATORS**

Spring boot’s module Actuator allows you to monitor and manage application usages in production environment, without coding and configuration for any of them. These monitoring and management information is exposed via [REST](https://restfulapi.net/) like endpoint URLs.

Add dependency

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

</dependencies>

Most applications exposes endpoints via HTTP, where the ID of the endpoint along with a prefix of /actuator is mapped to a URL. For example, by default, the health endpoint is mapped to /actuator/health.

By default, only /health and /info are exposed via Web APIs. Rest are exposed via JMX. Use **management.endpoints.web.exposure.include=\***to expose all endpoints through the Web APIs.

**Custom spring boot actuators**

Spring Boot 2 provides an easy way to create custom endpoints. Spring Boot 2.x introduced **@Endpoint** annotation. Spring Boot automatically expose endpoints with **@Endpoint, @WebEndpoint, or @WebEndpointExtension** over HTTP using Jersey, Spring MVC, or Spring WebFlux.

Spring Boot 2.x Actuator support CURD model, it supports read, writes and delete operation with the endpoints. The @Endpoint annotation can be used in combination with @ReadOperation, @WriteOperation and @DeleteOperation to develop endpoints.

**2.1 Creating Custom Endpoint**

We are creating a custom health endpoint, this endpoint will provide a custom information to the client.

Data Model

@JsonInclude(JsonInclude.Include.NON\_EMPTY)

public class CustomHealth {

private Map<String, Object> healthDetails;

@JsonAnyGetter

public Map<String, Object> getHealthDetails() {

return this.healthDetails;

}

}

**Custom Health endpoint.**

@Component

@Endpoint(id="custom-health")

public class CustomHealthEndPoint {

@ReadOperation

public CustomHealth health() {

Map<String, Object> details = new LinkedHashMap<>();

details.put("CustomHealthStatus", "Everything looks good");

CustomHealth health = new CustomHealth();

health.setHealthDetails(details);

return health;

}

@ReadOperation

public String customEndPointByName(@Selector String name) {

return "custom-end-point";

}

@WriteOperation

public void writeOperation(@Selector String name) {

//perform write operation

}

@DeleteOperation

public void deleteOperation(@Selector String name){

//delete operation

}

}

[pullquote align=”normal”]Keep an eye on [#11107](https://github.com/spring-projects/spring-boot/issues/11107) while naming your endpoint. There is an interesting issue if we name our endpoint in camel case [/pullquote]

The Id property of the @Endpoint annotation determines the mapping of our endpoint (in our example it is / custom-health).

**@ReadOperation – HTTP Get method.**

**@WriteOperation – POST method.**

**@DeleteOperation – HTTP DELETE operation.**

To access our custom endpoint, use http://host:port/actuator<em>/custom-health</em></em> to check the output.

{

"CustomHealthStatus":"Everything looks good"

}