

[Home](#) / [Spring Boot](#) / Spring Boot Actuator

Spring Boot Actuator

Last Modified: December 26, 2020

In this **Spring boot actuator tutorial**, learn about in-built HTTP endpoints available for any boot application for different **monitoring and management purposes**. Before the spring framework, if we had to introduce this type of monitoring functionality in our applications then we had to manually develop all those components and that too was very specific to our need. But with spring boot we have **Actuator** module which makes it very easy.

We just need to configure a few things and we are done – all the management and monitoring related information is easily available. Let's learn to configure **Spring boot 2 actuator endpoints**.

Table of Contents

1. [Spring Boot Actuator Module](#)
 - 1.1. [Maven](#)
 - 1.2. [Endpoint URLs](#)
 - 1.3. [Security](#)
 - 1.4. [Enabling Endpoints](#)
 - 1.5. [CORS](#)
 - 1.6. [Caching](#)
2. [Actuator Example](#)
3. [Actuator Endpoints Demo](#)
4. [Advance Configuration Options](#)
5. [Summary](#)

1. Spring Boot Actuator Module

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** like endpoint URLs.

1.1. Actuator Maven Dependency

Maven

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

Gradle

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-actuator'  
}
```

1.2. Important Actuator Endpoints

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.

application.properties

```
management.endpoints.web.exposure.include=*  
  
# To expose only selected endpoints  
#management.endpoints.jmx.exposure.include=health,info,env,beans
```

Some of important and widely used actuator endpoints are given below:

ENDPOINT	USAGE
<code>/auditevents</code>	Returns all auto-configuration candidates and the reason why they 'were' or 'were not' applied.
<code>/beans</code>	Returns a complete list of all the Spring beans in your application.
<code>/mappings</code>	Displays a collated list of all <code>@RequestMapping</code> paths..
<code>/env</code>	Returns list of properties in current environment
<code>/health</code>	Returns application health information.
<code>/caches</code>	It exposes available caches.
<code>/conditions</code>	Shows the conditions that were evaluated on configuration and auto-configuration.
<code>/configprops</code>	It displays a collated list of all <code>@ConfigurationProperties</code> .

<code>/integrationgraph</code>	It shows the Spring Integration graph. Requires a dependency on <code>spring-integration-core</code> .
<code>/loggers</code>	The configuration of loggers in the application..
<code>/scheduledtasks</code>	Displays the scheduled tasks in the application.
<code>/sessions</code>	Returns trace logs (by default the last 100 HTTP requests). Requires an <code>HttpTraceRepository</code> bean.
<code>/httptrace</code>	It allows retrieval and deletion of user sessions from a Spring Session-backed session store. Requires a Servlet-based web application using Spring Session.
<code>/shutdown</code>	Lets the application be gracefully shutdown. Disabled by default.
<code>/threaddump</code>	It performs a thread dump.
<code>/metrics</code>	It shows several useful metrics information like JVM memory used, system CPU usage, open files, and much more.

The Spring web application (Spring MVC, Spring WebFlux, or Jersey) provide the following additional endpoints:

ENDPOINT	USAGE
<code>/heapdump</code>	Returns an <code>hprof</code> heap dump file.
<code>/logfile</code>	Returns the contents of the logfile if <code>logging.file.name</code> or <code>logging.file.path</code> properties have been set.

1.3. Securing Endpoints

By default, `spring security` is enabled for all actuator endpoints if it available in the classpath.

If you wish to configure custom security for HTTP endpoints, for example, only allow users with a certain role to access then configure `WebSecurityConfigurerAdapter` in following manner:

```
@Configuration(proxyBeanMethods = false)
public class ActuatorSecurity extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
```

```
http.requestMatcher(EndpointRequest.toAnyEndpoint()).authorizeRequests((requests,
    requests.anyRequest().hasRole("ENDPOINT_ADMIN")));
http.httpBasic();
}

}
```

The above configuration ensures that only users with role `ENDPOINT_ADMIN` have access to actuation endpoints.

1.4. Enabling Endpoints

By default, all endpoints (except `/shutdown`) are enabled. To disable all endpoints, by default, use property:

Disable all endpoints by default

```
management.endpoints.enabled-by-default=false
```

Then use the only required endpoints which the application need to expose using the pattern `management.endpoint.<id>.enabled`.

Enable only needed endpoints

```
management.endpoint.health.enabled=true
management.endpoint.loggers.enabled=true
```

1.5. CORS support

`CORS` support is *disabled by default* and is only enabled once the `endpoints.cors.allowed-origins` property has been set.

```
management.endpoints.web.cors.allowed-origins=https://example.com
management.endpoints.web.cors.allowed-methods=GET,POST
```

Here the management context path is `/management`.

1.6. Caching the Response

Actuator endpoints automatically cache the responses to read operations that do not take any parameters. Use `cache.time-to-live property` to configure the amount of time for which an endpoint will cache the response.

Caching the response for 20 seconds

```
management.endpoint.beans.cache.time-to-live=20s
```

2. Spring Boot Actuator Endpoint Example

In this example, we will create a simple spring boot application and access the actuator endpoints to know more about them.

2.1. Development environment

- JDK 1.8, Eclipse, Maven – Development environment
- Spring-boot – Underlying application framework
- Spring-boot Actuator – Management endpoints

2.2. Create Maven Project

Start with creating one spring boot project from [Spring Initializer](#) site with **Web**, **Rest Repositories** and **Actuator** dependencies. Download project in zipped format. Unzip and then import project in eclipse as maven project.

Project
☒ Maven Project
☐ Gradle Project

Language
☒ Java ☐ Kotlin
☐ Groovy

Spring Boot
☐ 2.4.0 (SNAPSHOT) ☐ 2.4.0 (M1) ☐ 2.3.3 (SNAPSHOT)
☒ 2.3.2 ☐ 2.2.10 (SNAPSHOT) ☐ 2.2.9
☐ 2.1.17 (SNAPSHOT) ☐ 2.1.16

Project Metadata

Group	com.howtodoinjava
Artifact	spring-boot-actuator-example
Name	spring-boot-actuator-example
Description	Spring Boot Actuator Endpoint Example
Package name	com.howtodoinjava.spring-boot-actuator-example

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Boot Actuator OPS
Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

Generate Spring Boot Project

2.3. Add simple Rest endpoint

Now add one simple Rest endpoint **/example** to the application.

```
package com.example.springbootmanagementexample;  
  
import java.util.Date;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
public class SimpleRestController {
    @GetMapping("/example")
    public String example() {
        return "Hello User !! " + new Date();
    }
}
```

3. Spring Boot Actuator Endpoints Demo

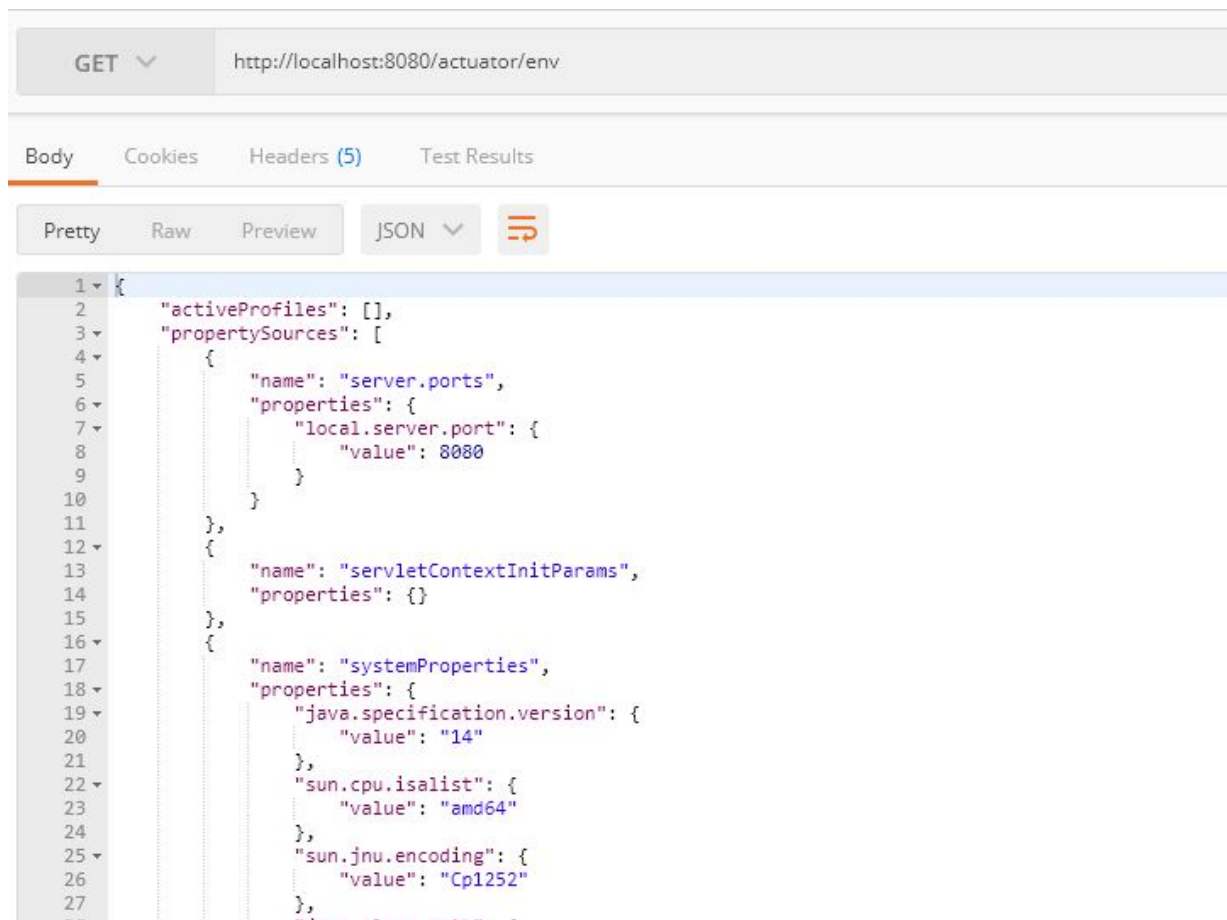
I have added `management.security.enabled=false` entry to the `application.properties` file to disable actuator security. Here I am more interested in actuator endpoints responses.

Do maven build using `mvn clean install` and start the application using `java -jar target\spring-boot-actuator-example-0.0.1-SNAPSHOT.jar` command. This will bring up one tomcat server in default port `8080` and application will be deployed in it.

Access `/example` API in browser to generate few monitoring information on server.

- <http://localhost:8080/actuator/env>

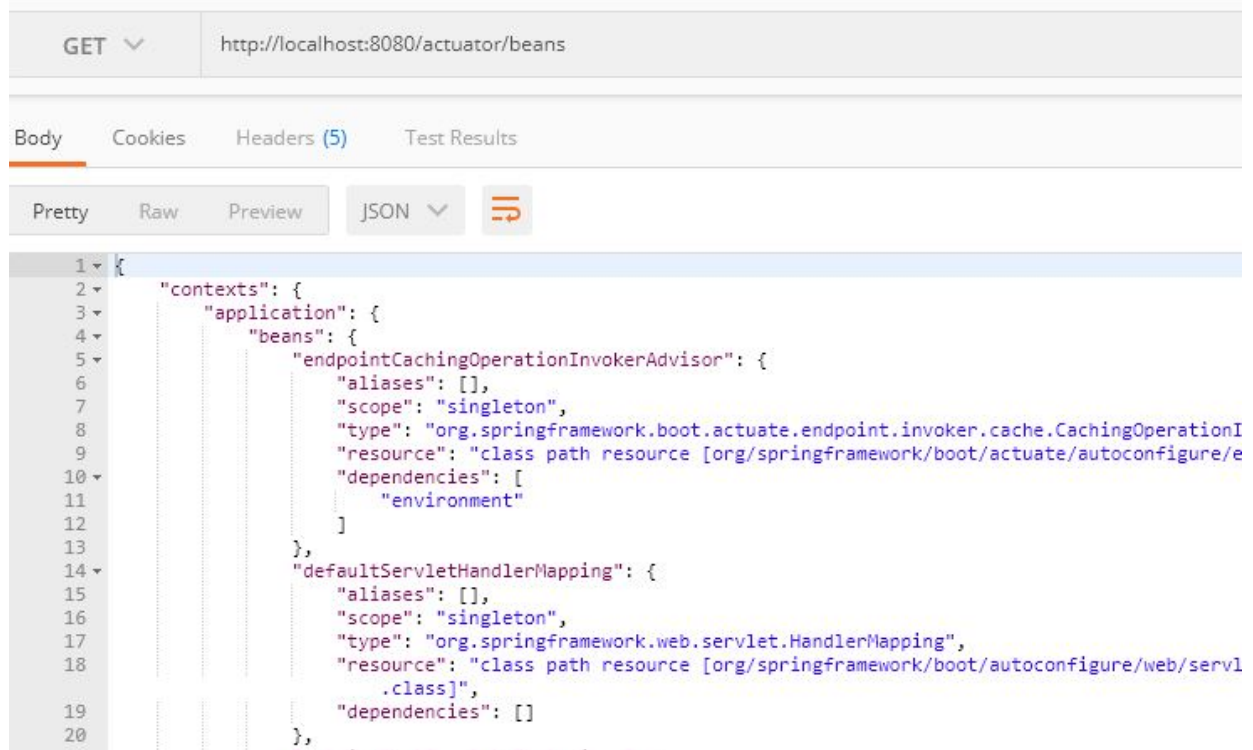
This will give all the environmental configuration about the server.



Endpoint env output

- <http://localhost:8080/actuator/beans>

This will give all the spring beans loaded in the context.

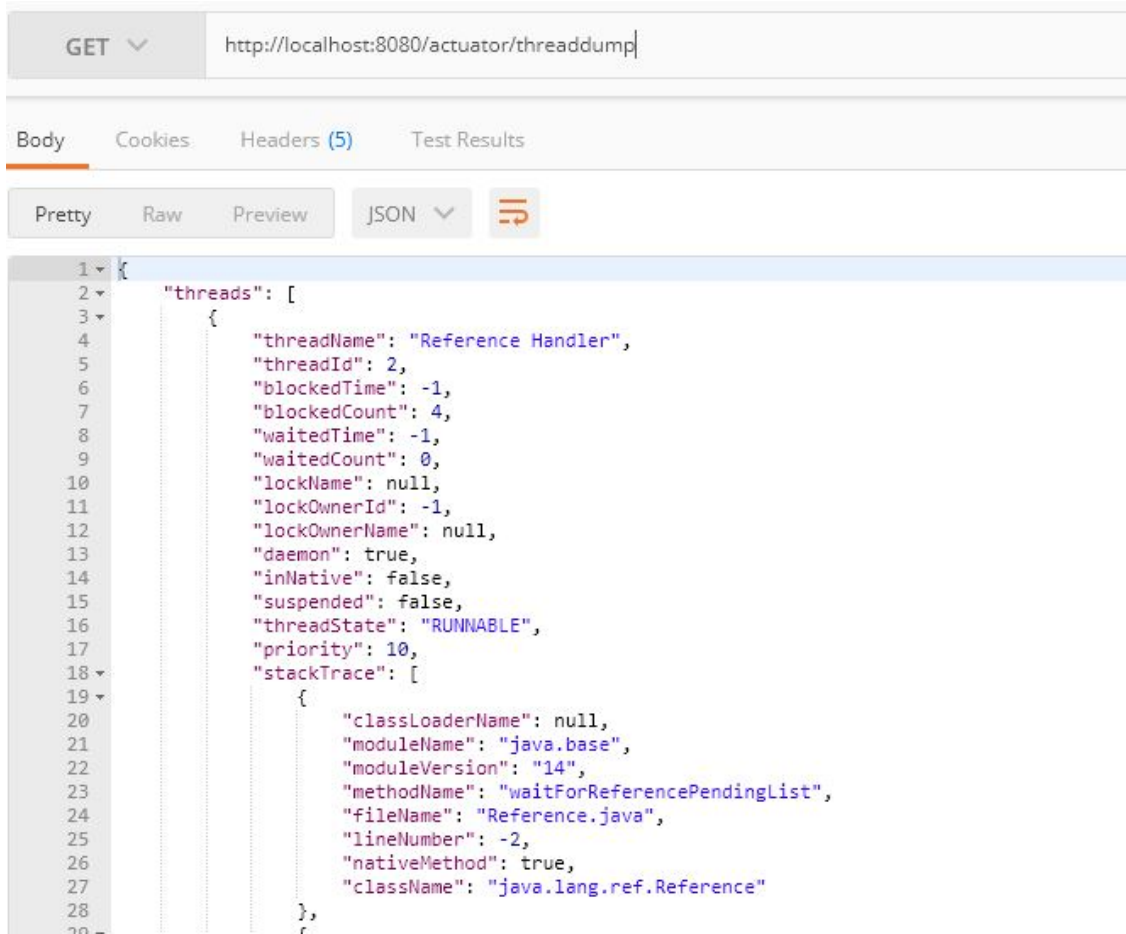


```
1 {
2   "contexts": {
3     "application": {
4       "beans": {
5         "endpointCachingOperationInvokerAdvisor": {
6           "aliases": [],
7           "scope": "singleton",
8           "type": "org.springframework.boot.actuate.endpoint.invoker.cache.CachingOperationI
9           "resource": "class path resource [org/springframework/boot/actuate/autoconfigure/e
10          "dependencies": [
11            "environment"
12          ]
13        },
14        "defaultServletHandlerMapping": {
15          "aliases": [],
16          "scope": "singleton",
17          "type": "org.springframework.web.servlet.HandlerMapping",
18          "resource": "class path resource [org/springframework/boot/autoconfigure/web/servl
19          .class]",
20          "dependencies": []
21        },
22        ...
23      }
24    }
25  }
```

Endpoint beans output

- <http://localhost:8080/actuator/threaddump>

This will give the current server thread dump.



```

1 {
2   "threads": [
3     {
4       "threadName": "Reference Handler",
5       "threadId": 2,
6       "blockedTime": -1,
7       "blockedCount": 4,
8       "waitedTime": -1,
9       "waitedCount": 0,
10      "lockName": null,
11      "lockOwnerId": -1,
12      "lockOwnerName": null,
13      "daemon": true,
14      "inNative": false,
15      "suspended": false,
16      "threadState": "RUNNABLE",
17      "priority": 10,
18      "stackTrace": [
19        {
20          "className": "java.lang.ref.Reference",
21          "moduleName": "java.base",
22          "moduleVersion": "14",
23          "methodName": "waitForReferencePendingList",
24          "fileName": "Reference.java",
25          "lineNumber": -2,
26          "nativeMethod": true,
27          "className": "java.lang.ref.Reference"
28        }
29      ]
30    }
31  ]
32 }

```

Endpoint threaddump output

Those endpoints will give standard information in the browser. These are the basic important endpoints we generally refer, but spring boot provides many more endpoints as mentioned in this [link](#)

4. Actuator Advance Configuration Options

4.1. Change the Management endpoint context path

By default all endpoints comes in default context path of the application, suffixed with `/actuator`. If for some reason, we have existing endpoints in application starting with `/actuator` then we can customize the base path to something else.

All we need to specify the new base path in the `application.properties`.

```
management.endpoints.web.base-path=/manage
```

Now you will be able to access all actuator endpoints under a new URL. e.g.

- `/manage/health`
- `/manage/dump`

- /manage/env
- /manage/beans

4.2. Customize the management server port

To customize the management endpoint port, we need to add this entry in the `application.properties` file.

```
management.server.port=8081
```

5. Summary

In this **spring boot actuator example**, we learned to configure management and monitoring endpoints with few easy configurations. So next time, you need to add application health checks or add monitoring support, you should consider adding the Spring actuator project and use these endpoints.

Feel free to drop your questions in the comments section.

Happy Learning !!

[Download Source Code](#)

Share this:



Subscribe to Blog

Enter your email address to subscribe and receive notifications of new posts by email.

Join 3,816 other subscribers

Feedback, Discussion and Comments

Ron

March 11, 2020

Yet another guide that is very old and no longer relevant to Spring Boot 2.x, yet keeps coming up in search results

[Reply](#)

Lokesh Gupta

March 11, 2020

Thanks for the feedback. I will cross-check and update it with latest information.

[Reply](#)

Ankita

February 3, 2020

Hi,

This is not working when spring boot version 2.2.4 is being used.

Any extra changes required for 2.2.4 ?

[Reply](#)

Suryanarayan Jena

October 4, 2019

How to configure same port for application and manage.

[Reply](#)

Leave a Reply

Enter your comment here...

Search Tutorials

Type and Press ENTER...



Meta Links

About Me
Contact Us
Privacy policy
Advertise
Guest and Sponsored Posts

Recommended Reading

10 Life Lessons
Secure Hash Algorithms
How Web Servers work?
How Java I/O Works Internally
Best Way to Learn Java
Java Best Practices Guide
Microservices Tutorial
REST API Tutorial
How to Start New Blog

Copyright © 2021 · Hosted on [Bluehost](#) · [Sitemap](#)