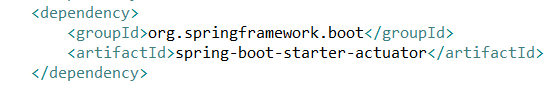
The Spring Boot Actuator module provides production-ready features such as monitoring, metrics, health checks, etc. The Spring Boot Actuator enables us to monitor the application using HTTP endpoints and JMX.

Spring Boot provides spring-boot-starter-actuator to autoconfigure Actuator. we can take advantage of Actuator’s features in order to monitor a Spring Boot application

To enable the actuator, simply add the Actuator starter POM to the project.

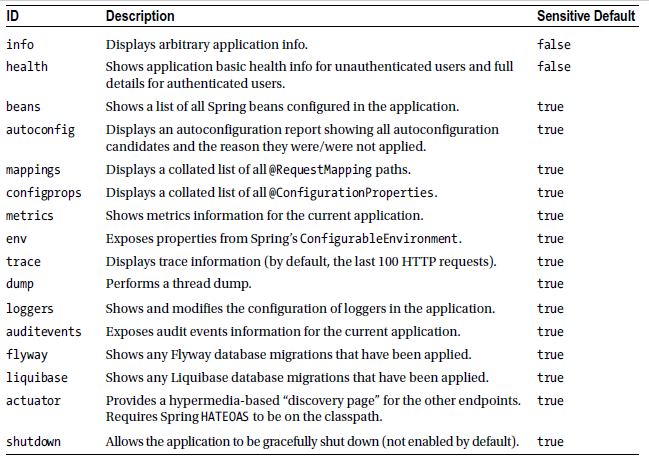


Now when we run the application, we could see the following logs in our console

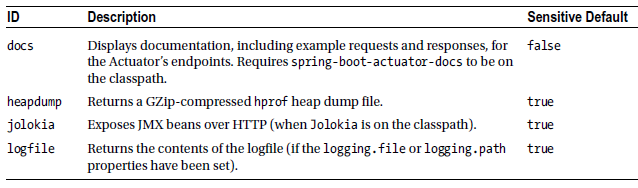


**Exploring Actuator’s Endpoints**

**Spring Boot Actuator Endpoints**



**Additional Spring Boot Actuator Endpoints for SpringMVC Applications**



The sensitive actuator endpoints can be accessed by authenticated users only. so for now we can disable security for actuator endpoints by setting the following property.

management.endpoints.web.exposure.include=\*

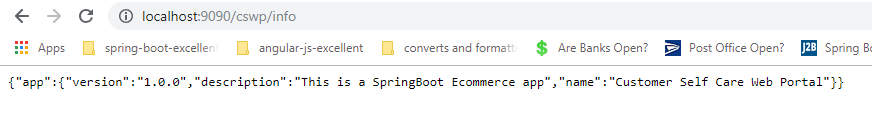
By default actuator endpoints run on the same HTTP port (server.port) with /application as the base path prefix.

**The /info Endpoint**

If we added any information about the application in the application.properties file using the

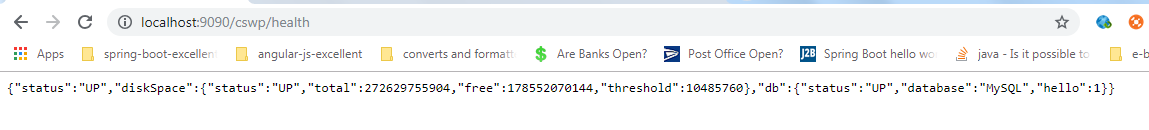
info.app.\* properties, as shown below, then we can view it at the [http://localhost:8080/<application](http://localhost:8080/%3capplication) name>/info endpoint.





**The /health Endpoint**

The /health endpoint shows the health of the application, including the disk space, databases, etc. Go to http://localhost:8080/cswp/health to check the health of the application

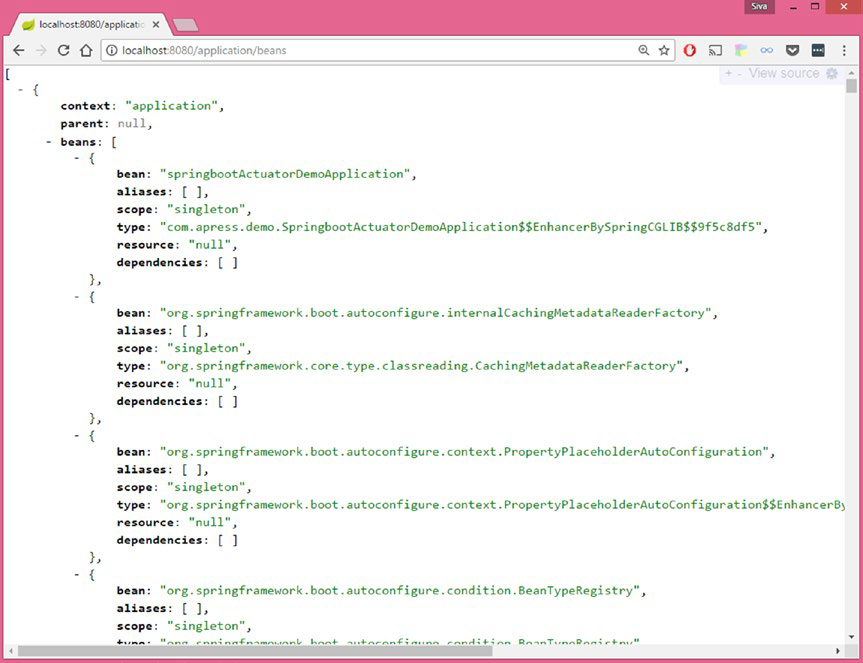


**Note:**

By default, the /health endpoint shows only whether the application is UP or DOWN for unauthorized users. If the user is authorized or management security is disabled, the /health endpoint shows additional information, such as disk space, databases, health, etc.

**The /beans Endpoint**

The /beans endpoint shows all the beans registered in our application, including the beans we explicitly configured and the beans autoconfigured by Spring Boot. Point our browser to <http://localhost:8080/cswp/beans,> we could see the following registered beans list

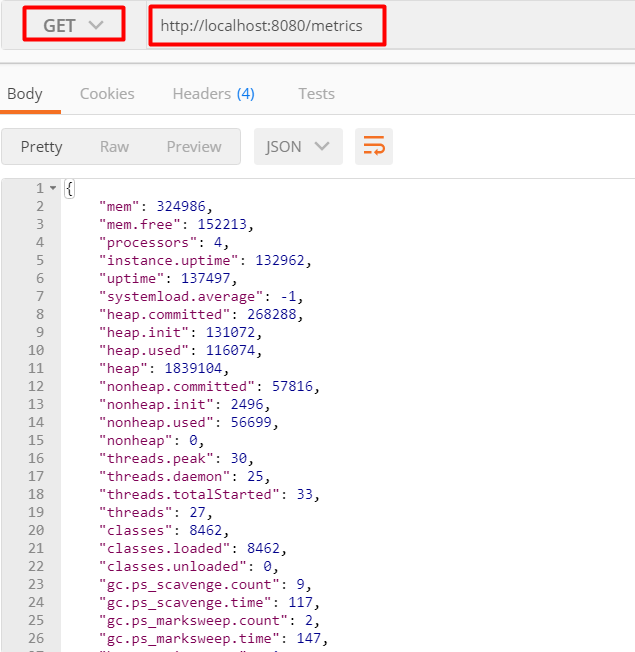


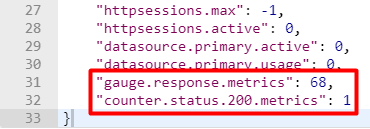
## Metrics Endpoint

The *metrics* endpoint exposes a range of application metrics that are extremely useful when it comes to **monitoring and trouble shooting** poorly performing applications. Data exposed by the *metrics*endpoint includes

* System Memory - Total and free system memory in KB
* System Uptime - Server uptime and application context uptime
* JVM Memory Usage - Initial, used and max available heap and non heap size
* Thread Statistics - total thread count, number of threads started and number of daemon threads
* Classloader - total classes loaded, number currently loaded and number unloaded
* Garbage Collector - GC algorithm run, number of times it has run and time taken for last GC
* HTTP Sessions - Current and peak number of HTTP sessions
* Data Source - total number of connections and total active connections
* Endpoints - number of times each endpoint has been called and the last response time for each endpoint

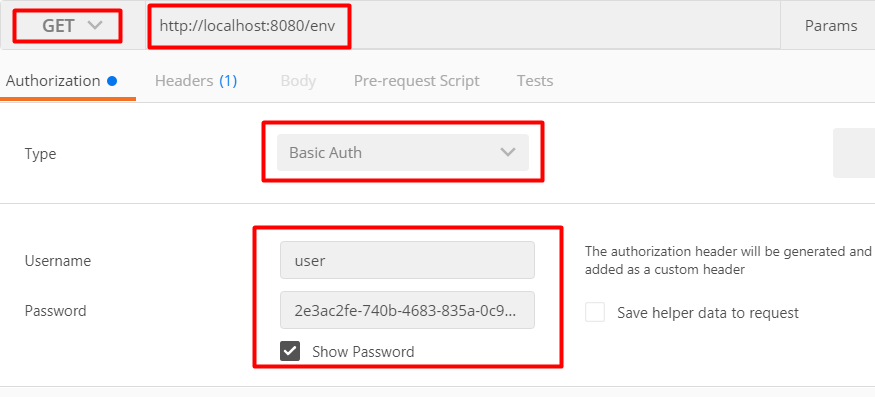
Using *Postman* we can access the *metrics* endpoint as below:





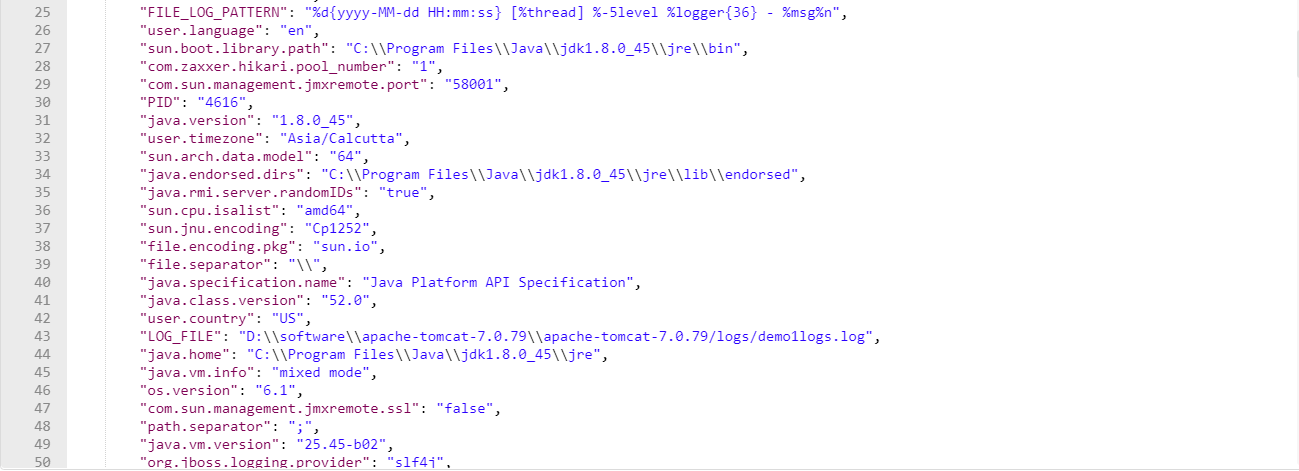
Most of the data in the sample response is fairly intuitive. One section worth describing further is the use of gauge and counter for endpoint metrics. A gauge is used to record a single value, for example on line **31 *gauge.response.metrics: 68.0***shows the last response time in milliseconds for the *metrics* endpoint. A *counter* records the increment or decrement of a value, for example on line **32 *counter.status.200.metrics: 1*** shows the number of times the *metrics* endpoint has been called since the app started.

**Env Endpoint**  
The *env* endpoint exposes data from the application *Environment*, the object that encapsulates all configuration available to the running application. Using *postman* we can access the *env* endpoint as:



Below is a sample response that includes **profiles, server ports, Servlet initialising parameters, system properties, environment variables and application properties**.







This information is very useful for checking that an application is using the expected configuration. It's particularly useful when an application is deployed to a test or prod environment and using externalized configuration such as an external properties file or environment variables.

**The /shutdown Endpoint**

The /shutdown endpoint can be used to gracefully shut down the application, which is not enabled by default. We can enable this endpoint by adding the following property to application.properties: **endpoints.shutdown.enabled=true**

After adding this property, you can send the HTTP POST method to http://localhost:8080/ application/shutdown to invoke the /shutdown endpoint.

Once the /shutdown endpoint is invoked successfully, we should see the following message:

**{**

**"message": "Shutting down, bye..."**

**}**