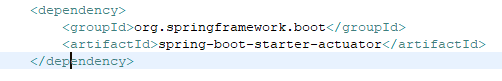
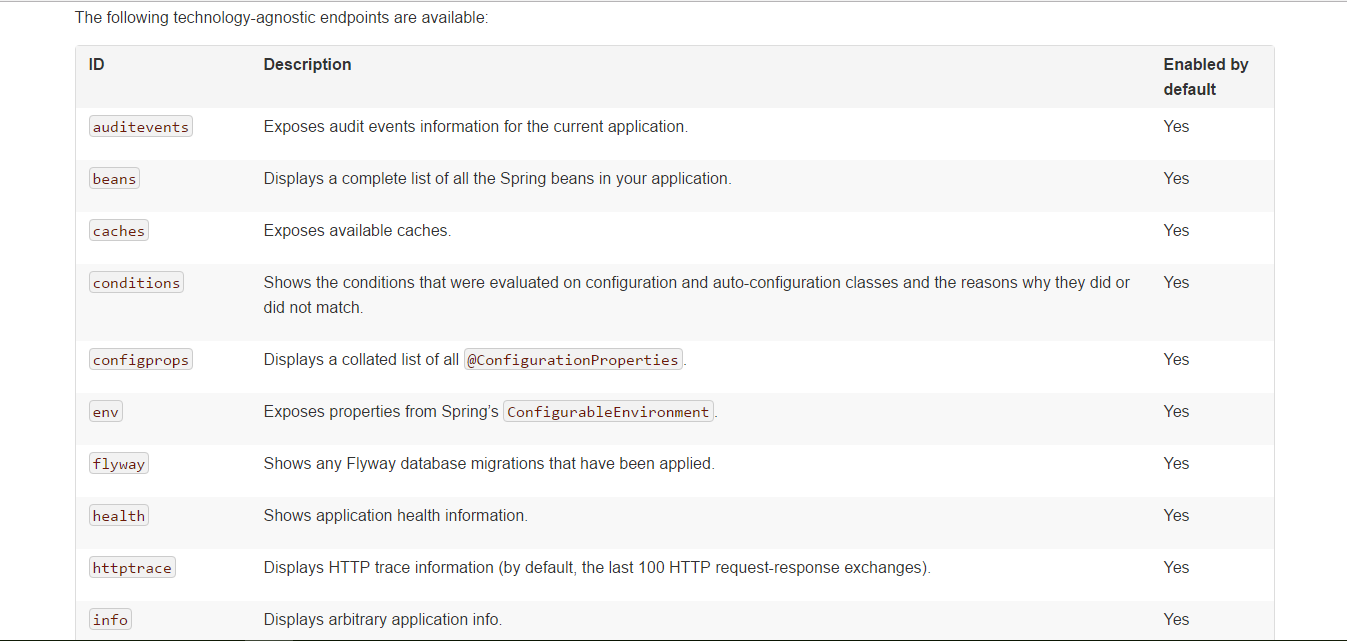
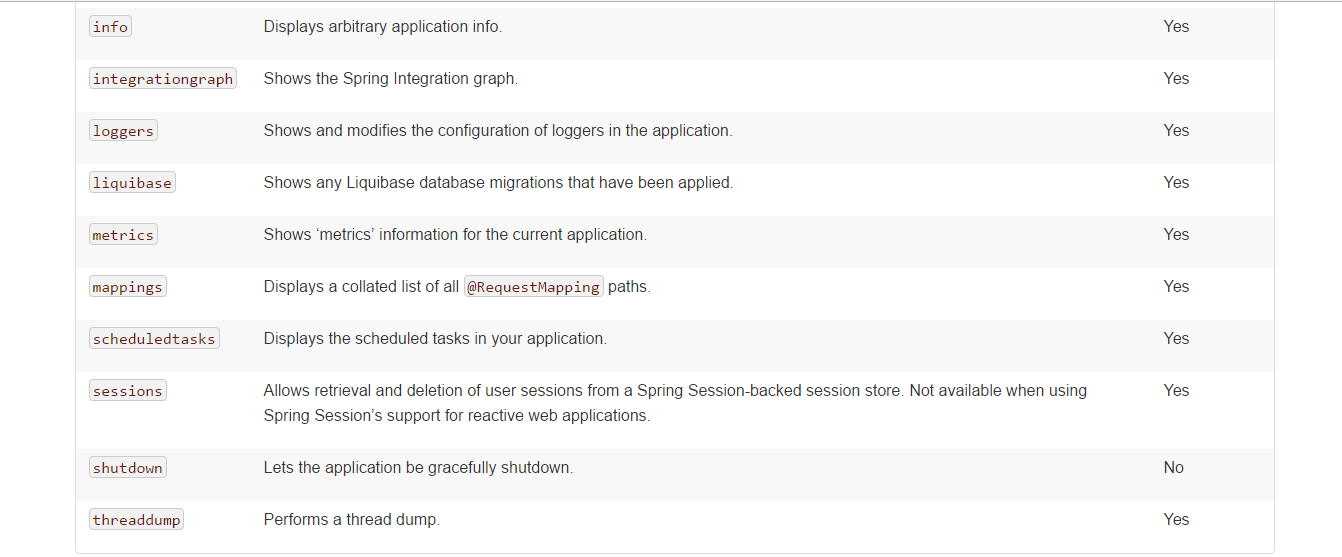
Actuator endpoints let you monitor and interact with your application. Spring Boot includes a number of built-in endpoints and lets you add your own. For example, the health endpoint provides basic application health information.

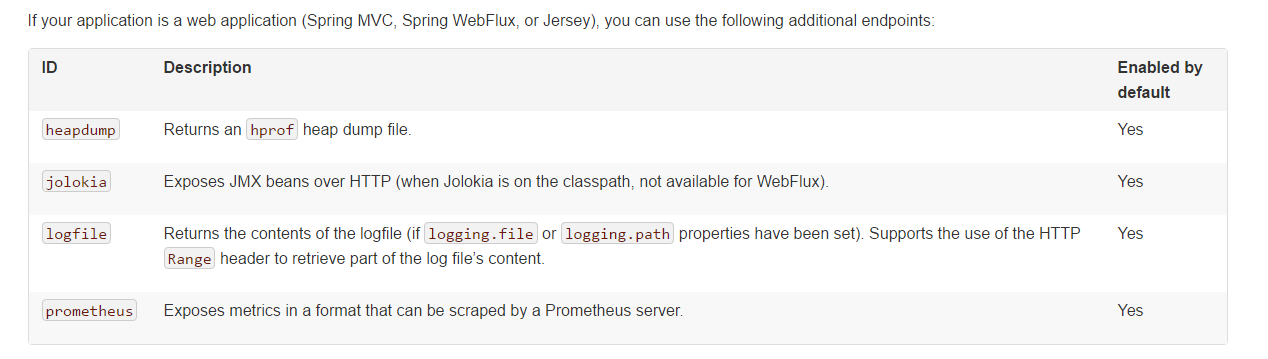
Most applications choose HTTP, where the ID of the endpoint along with a prefix of /actuator is mapped to a URL. For example, by default, the health endpoint is mapped to /actuator/health.

Dependency

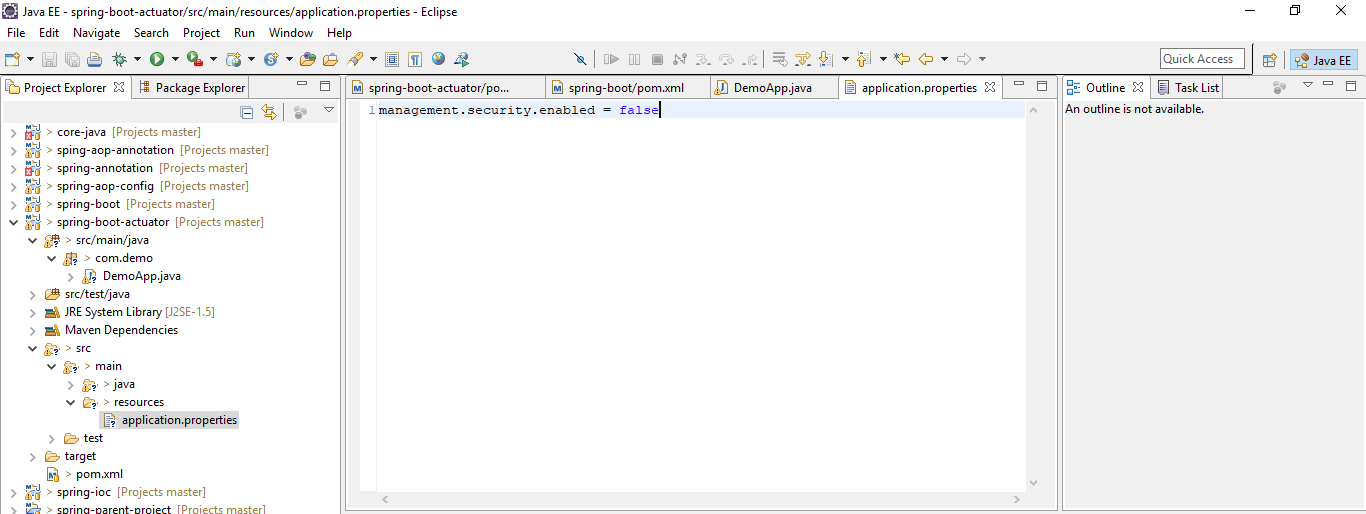








By default spring security is enabled for all endpoints. To disable use



Enabling Endpoints

By default, all endpoints except for shutdown are enabled. To configure the enablement of an endpoint, use its management.endpoint.<id>.enabled property. The following example enables the shutdown endpoint:

management.endpoint.shutdown.enabled=true

If you prefer endpoint enablement to be opt-in rather than opt-out, set the management.endpoints.enabled-by-default property to false and use individual endpoint enabled properties to opt back in. The following example enables the info endpoint and disables all other endpoints:

management.endpoints.enabled-by-default=false

management.endpoint.info.enabled=true

\* can be used to select all endpoints. For example, to expose everything over HTTP except the env and beans endpoints, use the following properties:

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

management.endpoints.web.exposure.exclude=env,beans

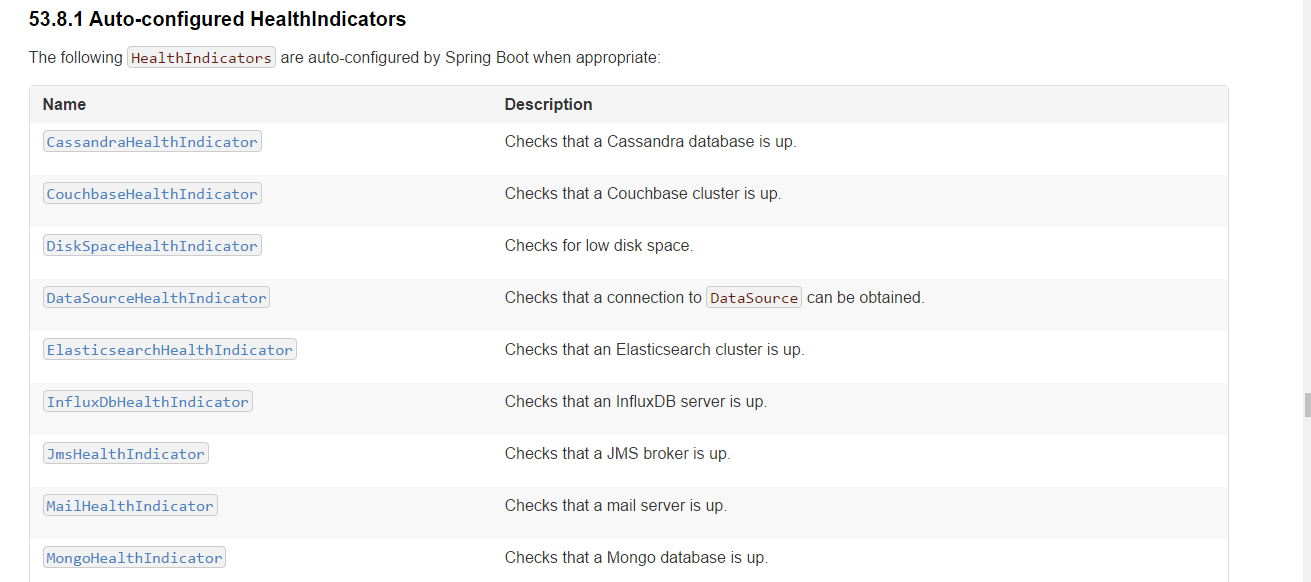
Custom endpoints

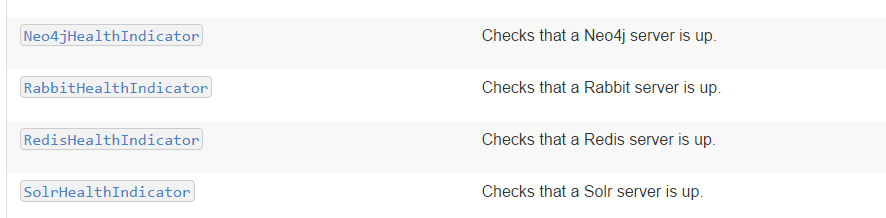
You can also write technology-specific endpoints by using @JmxEndpoint or @WebEndpoint. These endpoints are restricted to their respective technologies. For example, @WebEndpoint is exposed only over HTTP and not over JMX.

Controller endpoints

@ControllerEndpoint and @RestControllerEndpoint can be used to implement an endpoint that is only exposed by Spring MVC or Spring WebFlux. Methods are mapped using the standard annotations for Spring MVC and Spring WebFlux such as @RequestMapping and @GetMapping, with the endpoint’s ID being used as a prefix for the path. Controller endpoints provide deeper integration with Spring’s web frameworks but at the expense of portability. The @Endpoint and @WebEndpoint annotations should be preferred whenever possible.

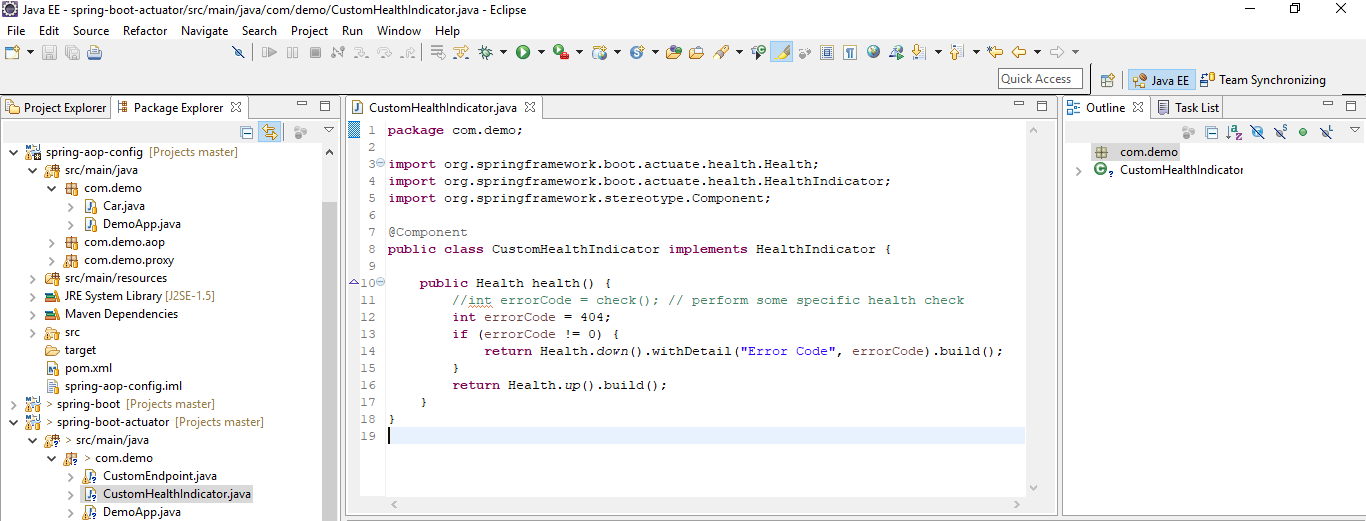
Health Indicator



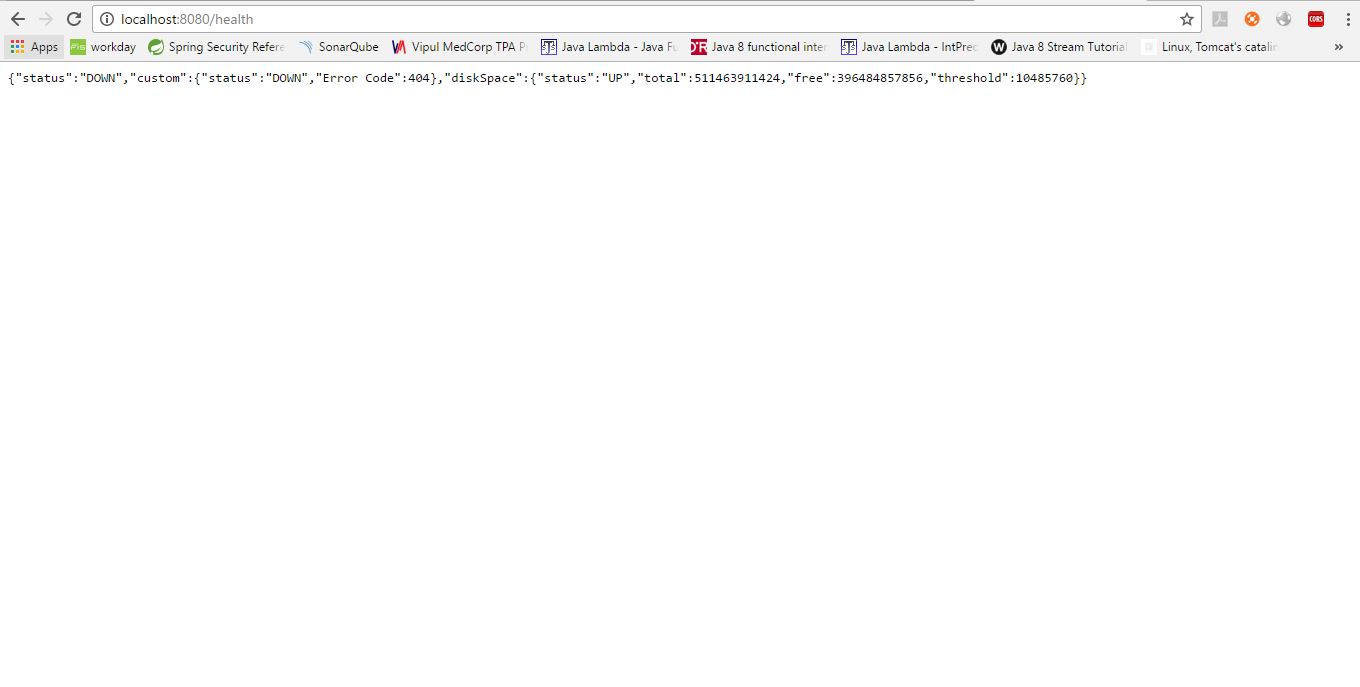


Custom Health Indicator

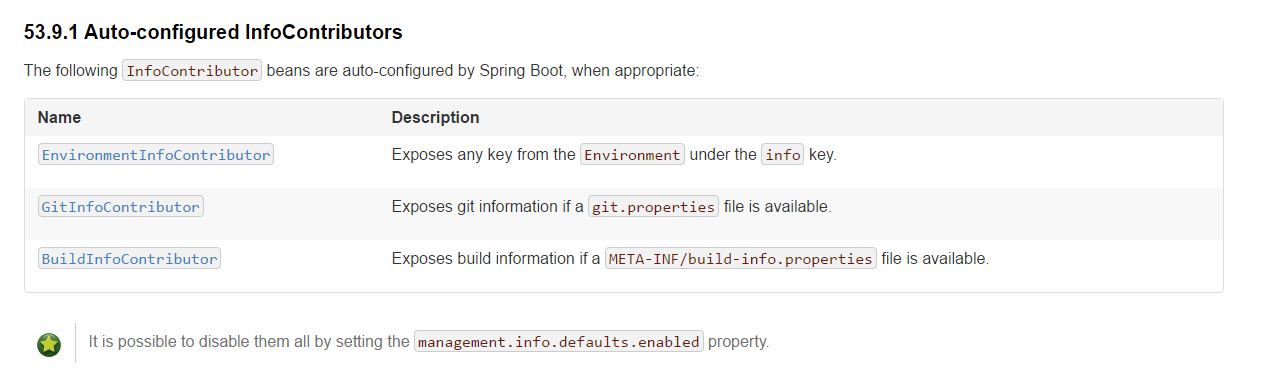
Override /health endpoint to provide custom info



Run and verify

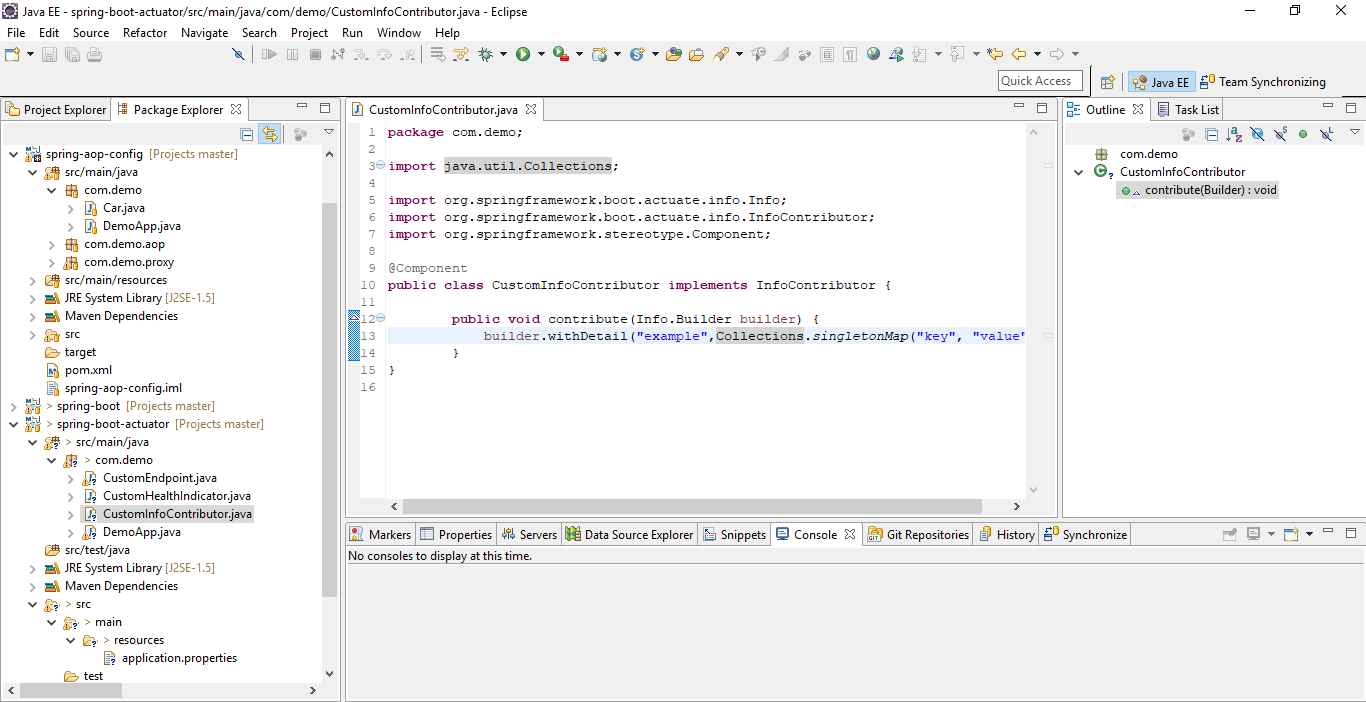


Info Contributors

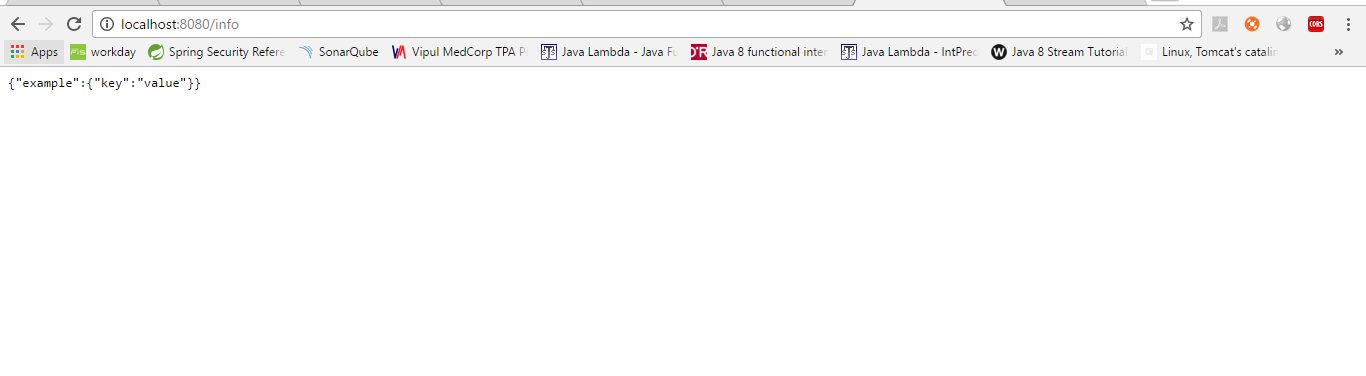


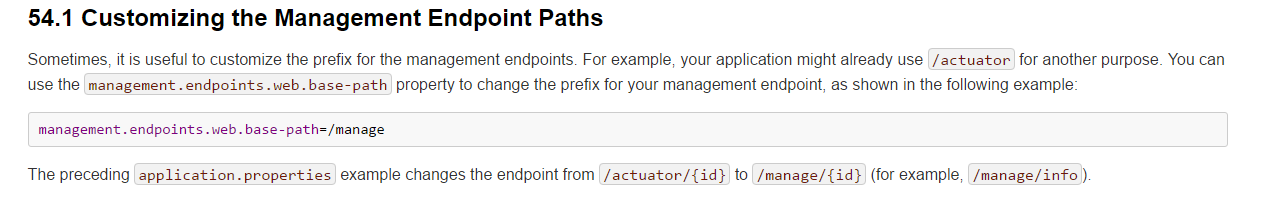
Custom Info Contributor

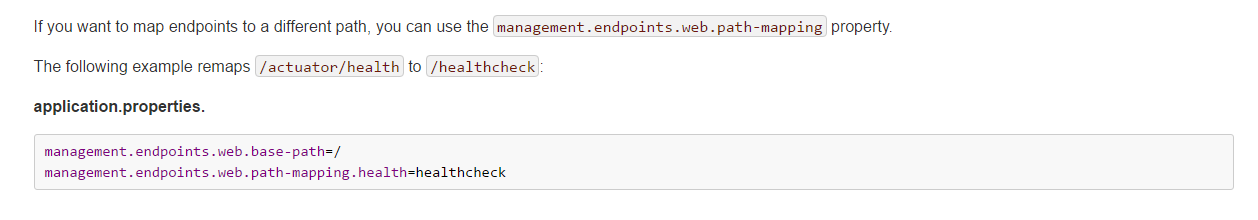
Override /info endpoint to provide custom info

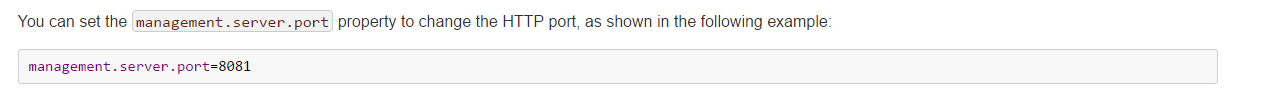


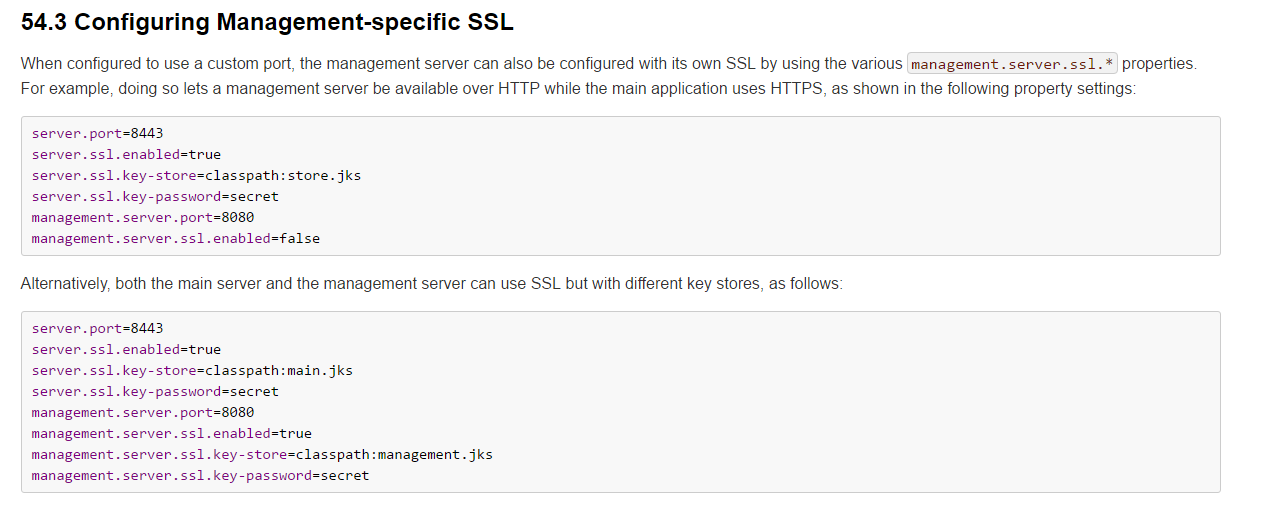
Run and verify

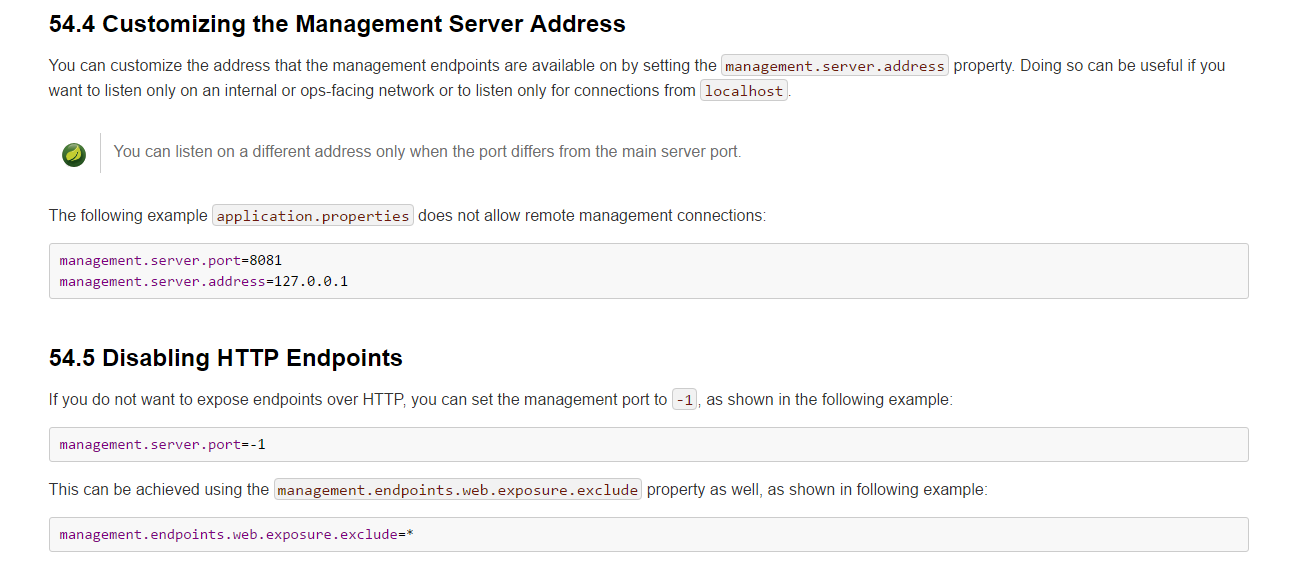










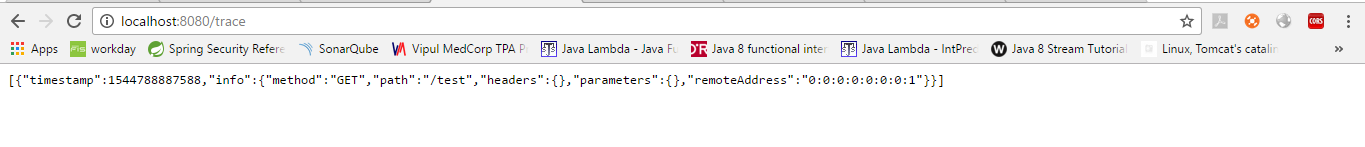




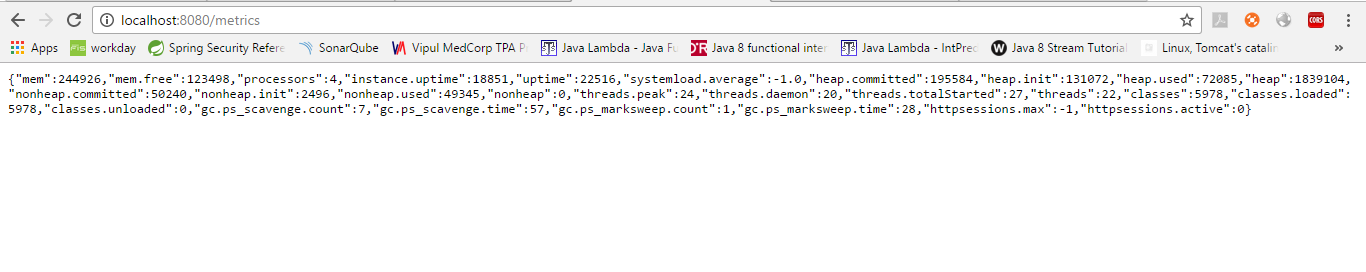
Customizing



Now run and verify



Metrics



The following system metrics are exposed by Spring Boot:

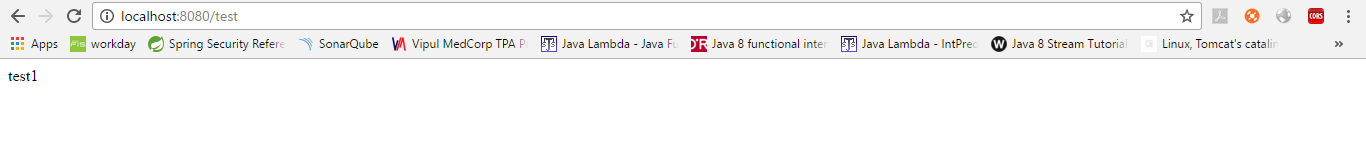
* The total system memory in KB (mem)
* The amount of free memory in KB (mem.free)
* The number of processors (processors)
* The system uptime in milliseconds (uptime)
* The application context uptime in milliseconds (instance.uptime)
* The average system load (systemload.average)
* Heap information in KB (heap, heap.committed, heap.init, heap.used)
* Thread information (threads, thread.peak, thread.daemon)
* Class load information (classes, classes.loaded, classes.unloaded)
* Garbage collection information (gc.xxx.count, gc.xxx.time)

Tomcat session metrics

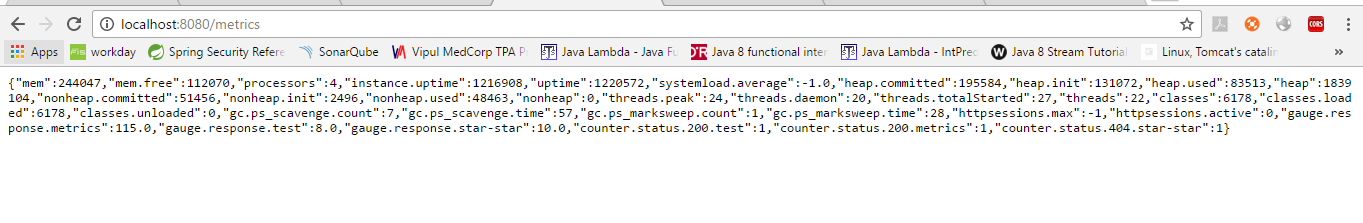
If you are using Tomcat as your embedded servlet container, session metrics will automatically be exposed. The httpsessions.active and httpsessions.max keys provide the number of active and maximum sessions.

Spring MVC Metrics

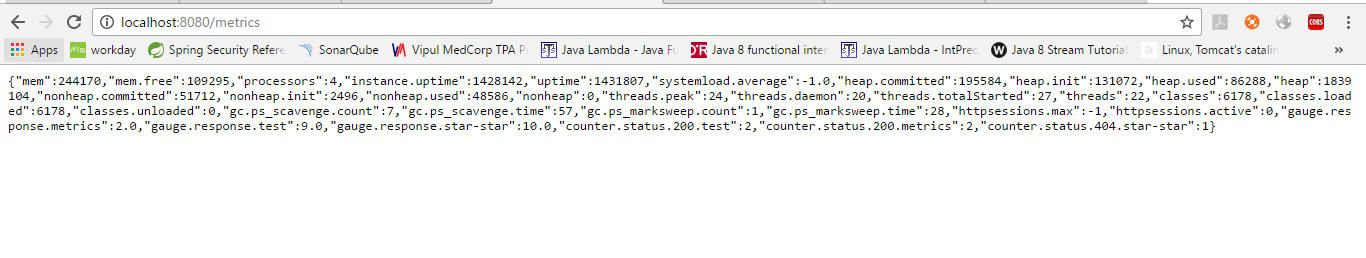
Hit a rest URL



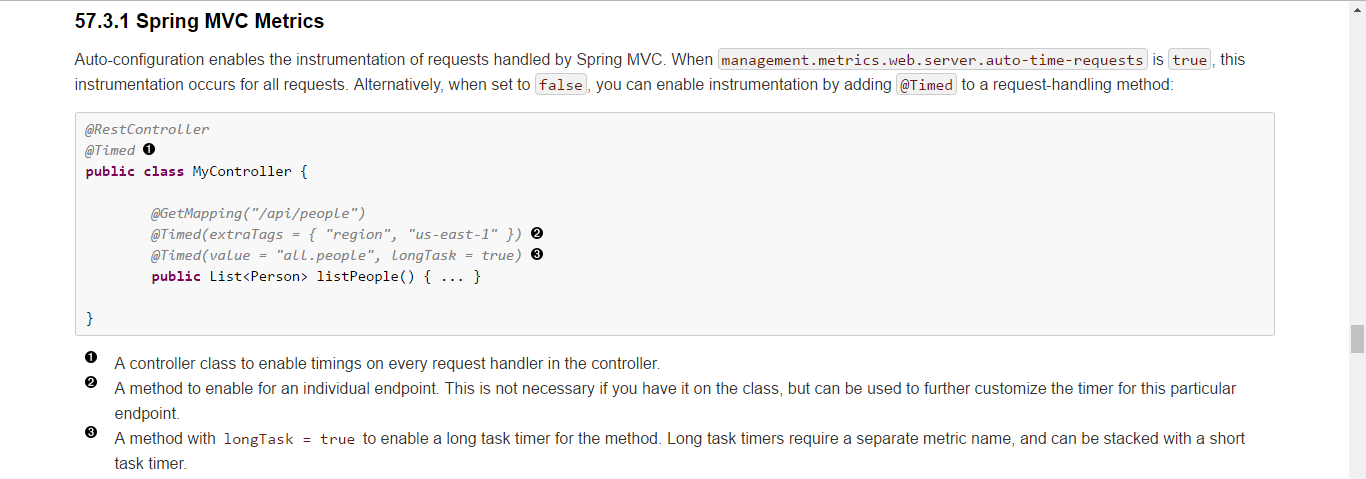
Now run metrics



Try again counter will update

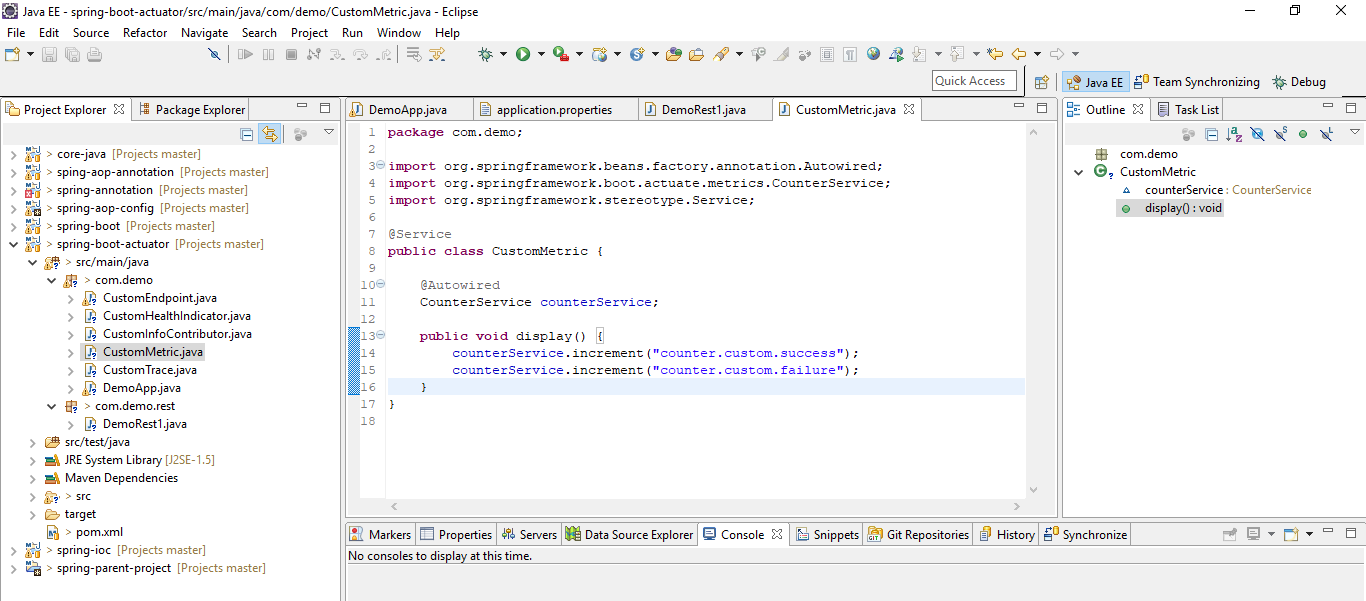


Default available to all requests. To customize this

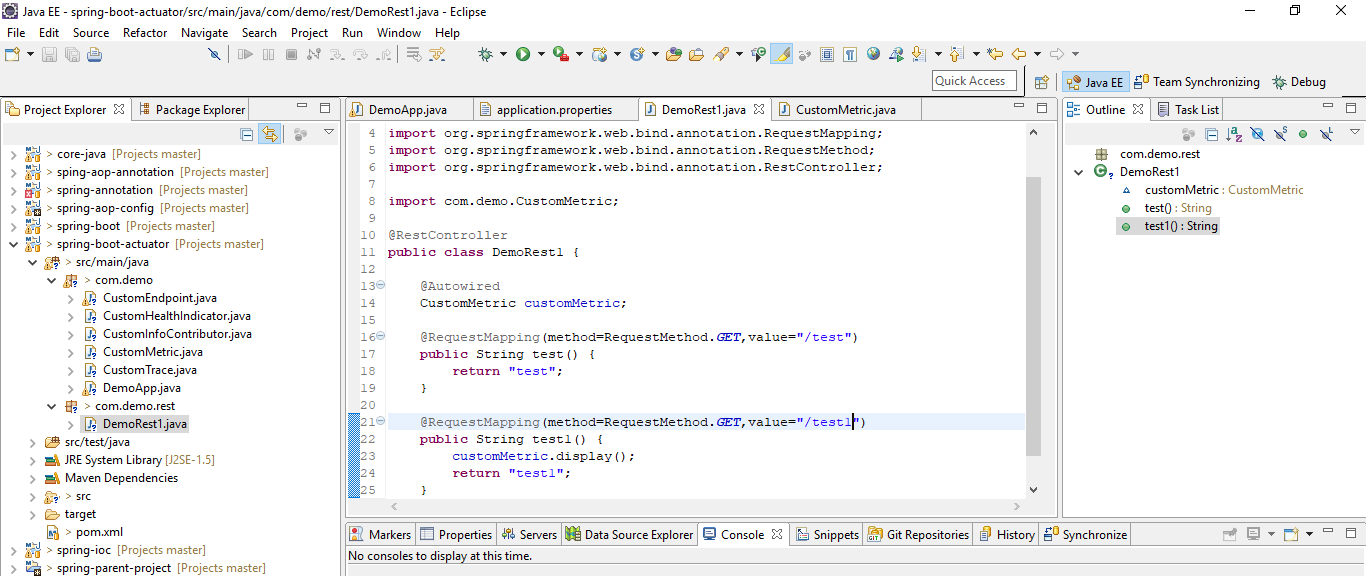


Custom Metric Data

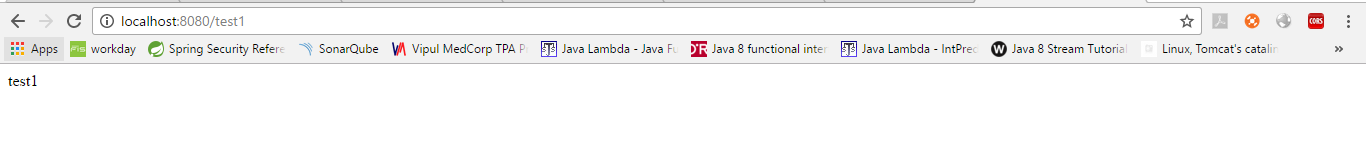
To record your own metrics inject a CounterService and/or GaugeService into your bean. The CounterService exposes increment, decrement and reset methods; the GaugeService provides a submit method.

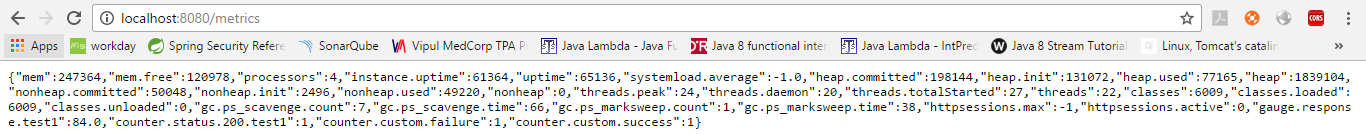


Calling this method in rest controller



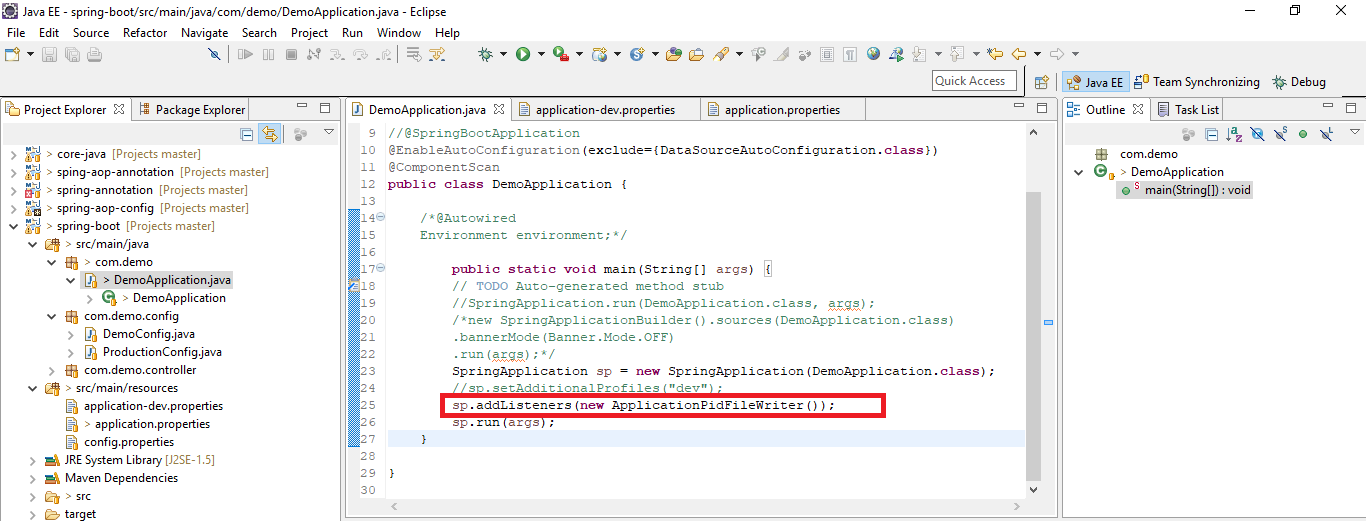
Run and verify



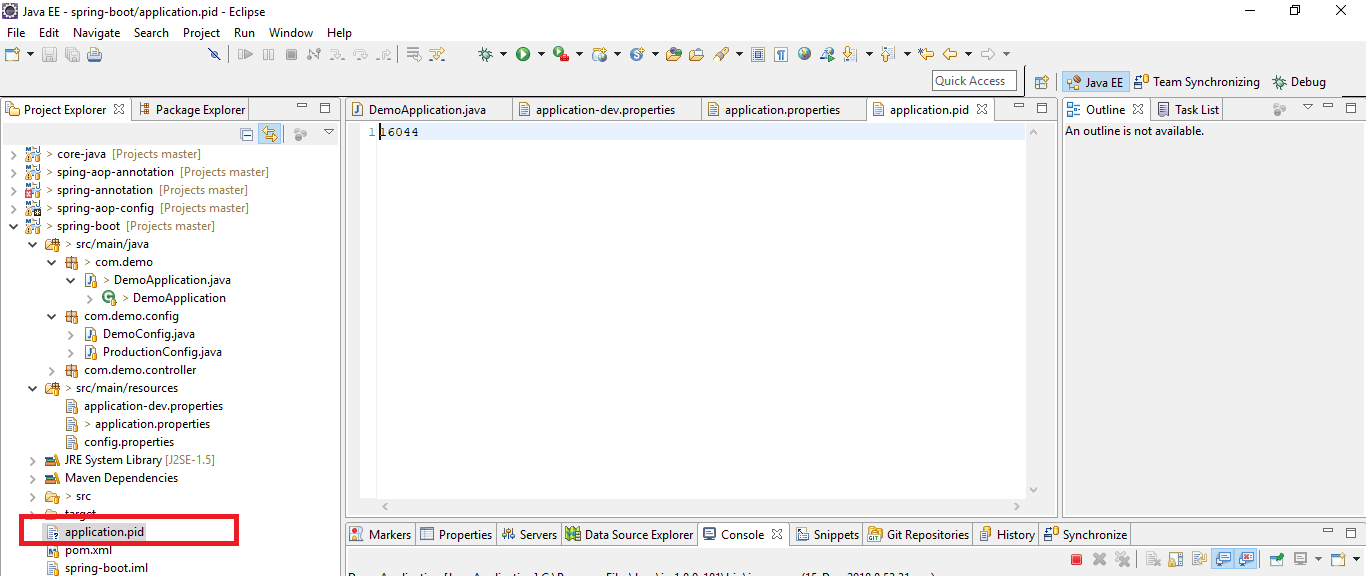


Process Monitoring

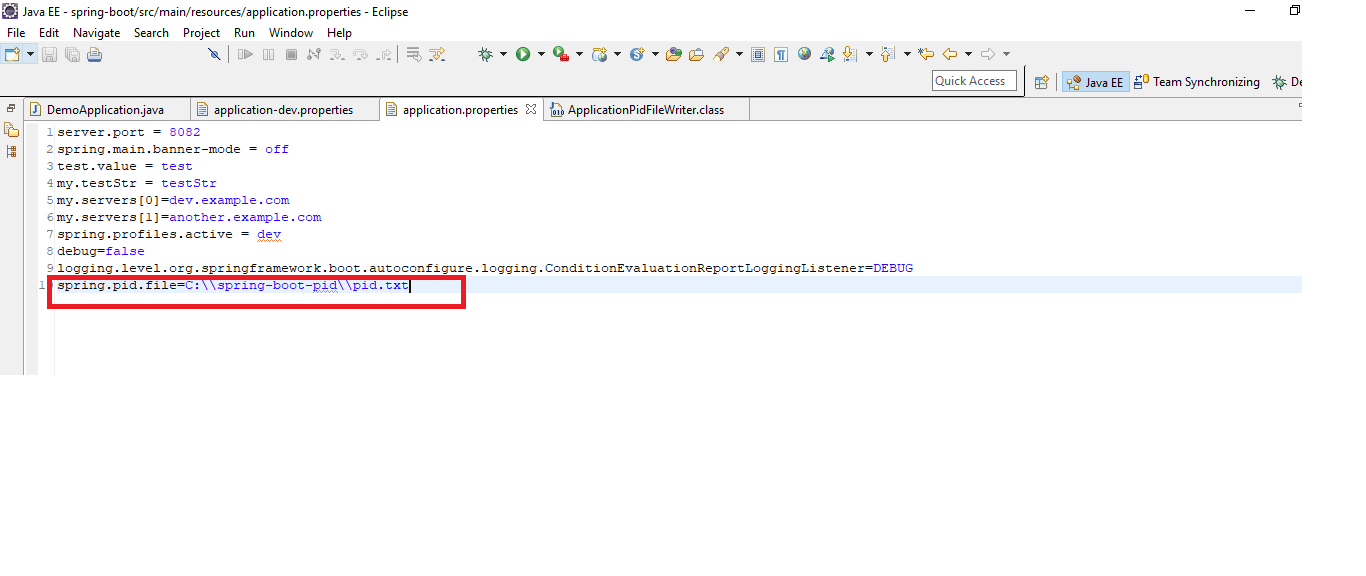
ApplicationPidFileWriter creates a file containing the application PID (by default, in the application directory with a file name of application.pid).



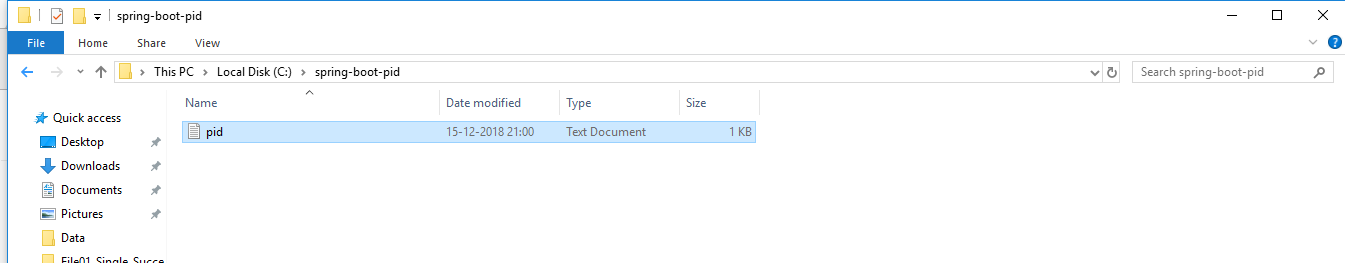
Run now, creates a file called application.pid



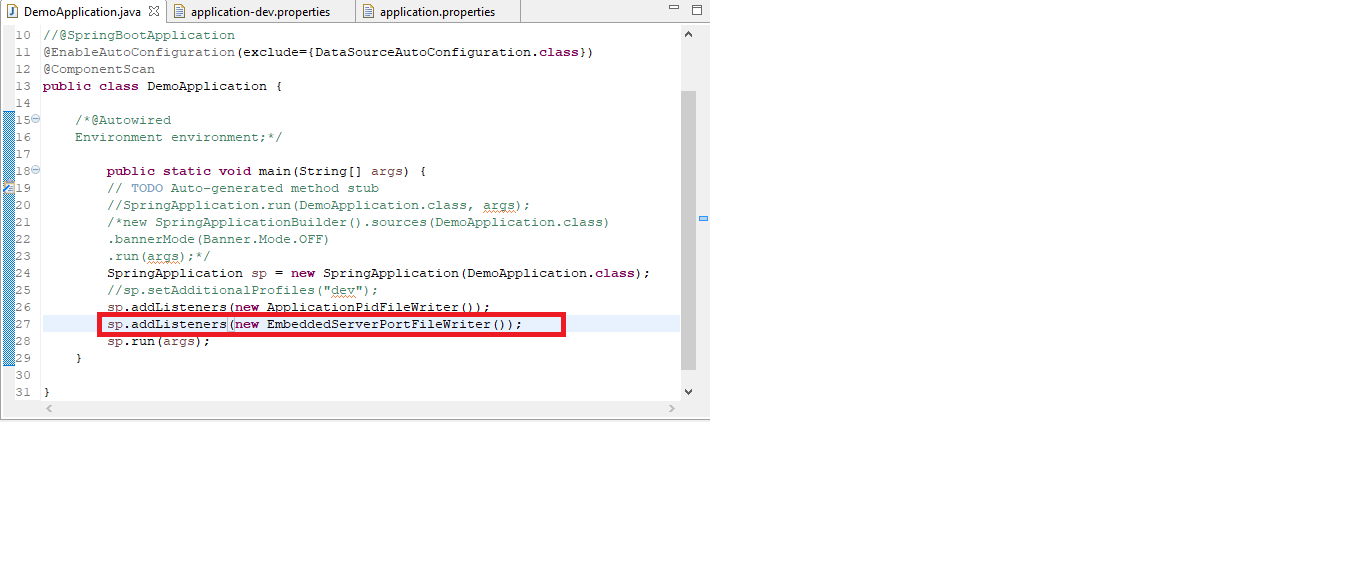
Mention file path in application properties



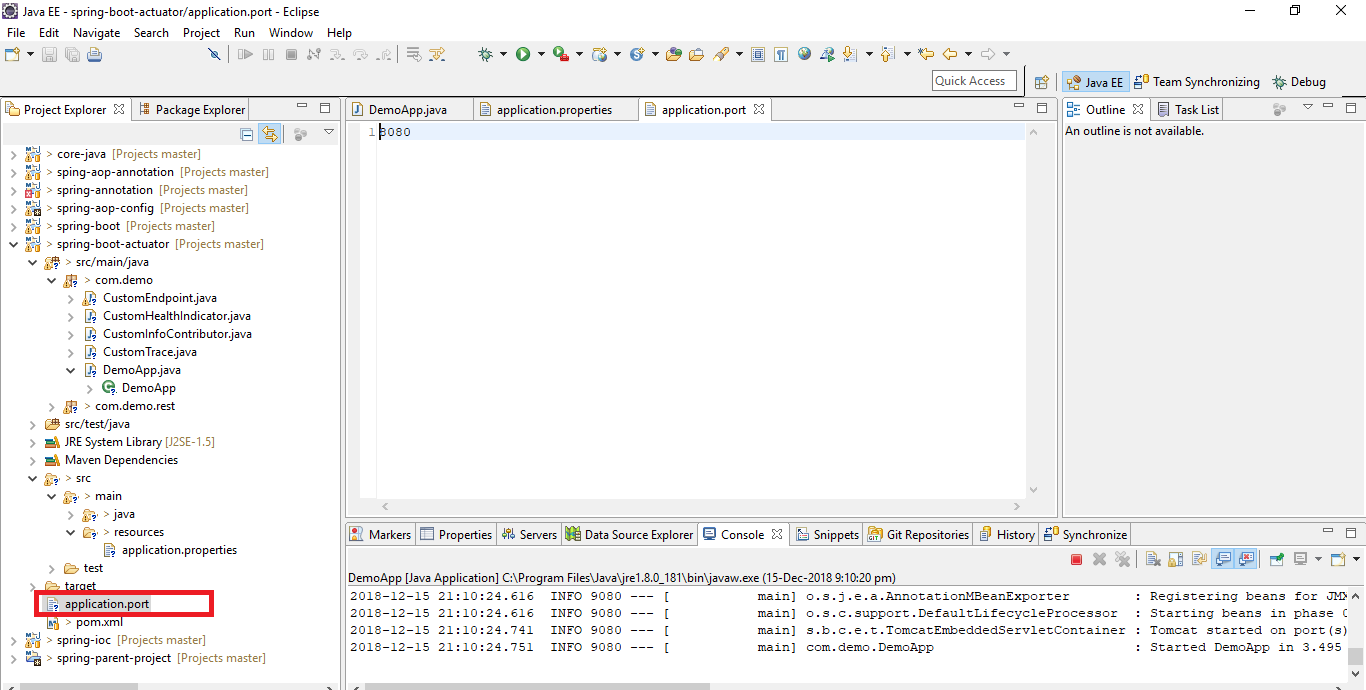
Run and verify



WebServerPortFileWriter creates a file (or files) containing the ports of the running web server (by default, in the application directory with a file name of application.port).



Run



Example projects

https://github.com/spring-projects/spring-boot/tree/v2.1.1.RELEASE/spring-boot-samples