Lab 4 - Introspection, Monitoring, and Metrics using Spring Boot Actuator

Set up the Actuator

1. Change to the lab directory (or build off of lab_01, lab_02, or lab_03):

   ```
   $ cd $COURSE_HOME/day_01/session_01/lab_04/initial/hello-spring-boot
   ```

2. Import the project's `pom.xml` into your editor/IDE of choice if you haven't already done so in a previous lab.

3. To `pom.xml` add the following dependency to include the starter for Spring Boot Actuator:

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

Introspection Endpoints

1. Build the application:

   ```
   $ mvn package
   ```

2. Run the application:

```
$ java -jar target/hello-spring-boot-0.0.1-SNAPSHOT.jar
```

3. Try out the following endpoints. The output is omitted here because it can be quite large:

   http://localhost:8080/beans

   Dumps all of the beans in the Spring context.

   http://localhost:8080/autoconfig

   Dumps all of the auto-configuration performed as part of application bootstrapping.

   http://localhost:8080/configprops

   Displays a collated list of all `@ConfigurationProperties`.

   http://localhost:8080/env

   Dumps the application's shell environment as well as all Java system properties.

   http://localhost:8080/mappings

   Dumps all URI request mappings and the controller methods to which they are mapped.

   http://localhost:8080/dump

   Performs a thread dump.

   http://localhost:8080/trace

   Displays trace information (by default the last few HTTP requests).

Build and Version Control Info

Spring Boot provides an endpoint (http://localhost:8080/info) that allows the exposure of arbitrary metadata.

One thing that it does well is expose information about the specific build and version control coordinates for a given deployment.

1. Add the following plugin to your Maven build. It will add Git branch and commit coordinates to the `/info` endpoint:

   ```
   <plugin>
           <groupId>pl.project13.maven</groupId>
           <artifactId>git-commit-id-plugin</artifactId>
           <configuration>
                   <dotGitDirectory>../../../../../.git</dotGitDirectory>
           </configuration>
   </plugin>
   ```

   NOTE The path `../../../../../.git` refers to the `.git` directory at the root of the course materials.

2. Add the following properties to `src/main/resources/application.yml`:

   ```
   info:
     build:
       artifact: @project.artifactId@
       name: @project.name@
       description: @project.description@
       version: @project.version@
   ```

   These will add the project's Maven coordinates to the `/info` endpoint. The Spring Boot Maven plugin will cause them to automatically be replaced in the assembled JAR.

3. Build the application:

   ```
   $ mvn package
   ```

4. Run the application:

   ```
   $ java -jar target/hello-spring-boot-0.0.1-SNAPSHOT.jar
   ```

5. Visit the application in the browser (http://localhost:8080/info), and verify that the output is similar to the following:

   ```
   {
     build: {
       artifact: "hello-spring-boot",
       name: "hello-spring-boot",
       description: "Hello Spring Boot",
       version: "0.0.1-SNAPSHOT"
     },
     git: {
       branch: "master",
       commit: {
         id: "a15f771",
         time: "2015-05-03T16:51:31-0400"
       }
     }
   }
   ```

Health Indicators

Spring Boot provides an endpoint (http://localhost:8080/health) that allows for the notion of various health indicators.

1. Normally, when Spring Security is not enabled, the `/health` endpoint will only expose an `UP` or `DOWN` value. To simplify working with the endpoint for this lab, we will turn off its sensitivity. Add the following to `src/main/resources/application.yml`:

```
endpoints:
  health:
    sensitive: false
```

2. Create the class `io.pivotal.spring.hello.FlappingHealthIndicator` and into it paste the following code:

```
@Component
public class FlappingHealthIndicator implements HealthIndicator{

    private Random random = new Random(System.currentTimeMillis());

    @Override
    public Health health() {
        int result = random.nextInt(100);
        if (result < 50) {
            return Health.down().withDetail("flapper", "failure").withDetail("random", result).build();
        } else {
            return Health.up().withDetail("flapper", "ok").withDetail("random", result).build();
        }
    }
}
```

This demo health indicator will randomize the health check.

3. Build the application:

```
$ mvn package
```

4. Run the application:

```
$ java -jar target/hello-spring-boot-0.0.1-SNAPSHOT.jar
```

5. Visit the application in the browser (http://localhost:8080/health), and verify that the output is similar to the following (and changes randomly!):

   ```
   {
     status: "UP",
     flapping: {
       status: "UP",
       flapper: "ok",
       random: 69
     },
     diskSpace: {
       status: "UP",
       free: 113632186368,
       threshold: 10485760
     }
   }
   ```

Metrics

Spring Boot provides an endpoint (http://localhost:8080/metrics) that exposes several automatically collected metrics for your application. It also allows for the creation of custom metrics.

1. Create the class `io.pivotal.spring.hello.GreetingService` and into it paste the following code:

```
@Component
public class GreetingService {

    @Autowired
    CounterService counterService;

    @Value("${greeting}")
    String greeting;

    public String getGreeting() {
        counterService.increment("counter.services.greeting.invoked");
        return greeting;
    }
}
```

This class is using the `@Autowired` `CounterService` to count the number of times that the `getGreeting()` method has been invoked.

2. Refactor the contents of the class `io.spring.hello.HelloSpringBootApplication`:

```
@Autowired
private GreetingService greetingService;

@RequestMapping("/")
public String hello() {
    return String.format("%s World!", greetingService.getGreeting());
}

public static void main(String[] args) {
    SpringApplication.run(HelloSpringBootApplication.class, args);
}
```

`hello()` is now delegating the source of the greeting to our newly created `GreetingService`.

3. Build the application:

   ```
   $ mvn package
   ```

4. Run the application:

   ```
   $ java -jar target/hello-spring-boot-0.0.1-SNAPSHOT.jar
   ```

5. Visit the application in the browser (http://localhost:8080) and refresh the page several times.

6. Now visit the `/metrics` endpoint (http://localhost:8080/metrics). Among the autogenerated metrics you should see a `counter` for the `GreetingService` invocations:

   ```
   counter.services.greeting.invoked: 16,
   ```

   To learn more about the autogenerated metrics, visit http://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-metrics.html.

Last updated 2015-06-13 19:24:51 EDT