Spring Boot Actuator

LAHIRU
LIYANAPATHIRANA

What is Spring Boot Actuator?

Spring Boot Actuator is a powerful sub-project of Spring Boot that provides production-ready features for monitoring and managing Spring Boot applications.

It offers a suite of built-in endpoints that expose valuable operational information about the running Spring Boot application with minimal effort.

It exposes operational information (like health, metrics, environment details, and more) through endpoints via HTTP, JMX, or custom channels.

Key Features

Spring Boot Actuator comes with many built-in endpoints that provide deep insights into the running application:

- **Health Checks:** Verify application and dependency status (e.g., databases, disks).
- **Metrics:** Track JVM memory, HTTP request rates, CPU usage, and custom metrics.
- **Environment Details:** Inspect configuration properties, profiles, and environment variables.

Key Features

- **Environment Details:** Inspect configuration properties, profiles, and environment variables.
- **Bean Listing:** List all Spring beans in the Application Context.
- **Endpoint Mappings:** View registered REST API routes.
- **Logging Control:** View logging configurations and dynamically adjust log levels at runtime.

Important Actuator Endpoints

Health (**/actuator/health**):
Provides a simple or detailed health status (e.g., UP or DOWN) of the application, configurable to show details like database connectivity and disk space.

Info (**/actuator/info**):
Exposes arbitrary application details such as version, build information, or any custom info defined via configuration properties.

Metrics (**/actuator/metrics**):
Offers a collection of application metrics (e.g., JVM memory, HTTP requests) that can be queried as a whole or by individual metric names for real-time performance monitoring.

LAHIRU
LIYANAPATHIRANA

Important Actuator Endpoints

Env (**/actuator/env**):
Exposes environment variables and configuration properties from the Spring Environment; marked as sensitive due to the risk of revealing critical system details.

Mappings (**/actuator/mappings**):
Lists all registered REST endpoint mappings, detailing paths and associated controllers to aid in debugging API configurations.

Loggers (**/actuator/loggers**):
Allows you to view and dynamically modify logging levels at runtime, facilitating on-the-fly adjustments for troubleshooting purposes.

LAHIRU
LIYANAPATHIRANA

Important Actuator Endpoints

Beans (**/actuator/beans**):
Lists all Spring beans in the application context along with their dependencies and scopes, providing insight into the internal wiring (sensitive).

Config Props (**/actuator/configprops**):
Displays a consolidated view of all @ConfigurationProperties beans to show how externalized configurations are bound; useful for debugging but sensitive.

Thread Dump (**/actuator/threaddump**):
Captures a snapshot of JVM threads to diagnose performance issues or deadlocks, exposing internal runtime data (sensitive).

Important Actuator Endpoints

Heap Dump (/actuator/heapdump): Enables downloading a JVM heap dump (in HPROF format) for memory analysis, highly sensitive due to the potential exposure of internal state.

Prometheus (/actuator/prometheus): Formats and exposes metrics in a Prometheus-friendly format, enabling seamless integration with external monitoring tools without being considered sensitive.

Setting Up Spring Boot Actuator

Add the following starter dependency to add the actuator to the Spring Boot Application:

For Maven-based project:

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

For Gradle-based project:

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


Configuring/Customizing Endpoints

Spring Boot Actuator is highly customizable.

It is possible to:

- Enable or disable specific endpoints
- Configure security for endpoints
- Create custom health indicators
- Define custom metrics
- Build entirely new actuator endpoints

Configuring/Customizing Endpoints

Enable/Disable Specific Endpoints

Only the /health endpoint is exposed over HTTP by default, for security concerns. Exposed endpoints can be configured as follows:

```
# Expose all endpoints (use with caution in production)
management.endpoints.web.exposure.include=*

# Expose only health and info endpoints
management.endpoints.web.exposure.include=health,info

# Disable exposing env, beans, loggers endpoints
management.endpoints.web.exposure.exclude=env,beans,loggers
```

Change the base path for actuator endpoints (default is /actuator):

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


Configuring/Customizing Endpoints

Create Custom Health Indicators

Custom health indicators can be created by implementing the HealthIndicator interface or by extending the AbstractHealthIndicator class:

```java
@Component
public class DatabaseHealthIndicator extends AbstractHealthIndicator {
    @Autowired
    private DataSource dataSource;

    @Override
    protected void doHealthCheck(Health.Builder builder) {
        try (Connection conn = dataSource.getConnection()) {
            builder.up().withDetail("database", "PostgreSQL");
        } catch (Exception e) {
            builder.down(e);
        }
    }
}
```

Configuring/Customizing Endpoints

Create Custom Health Indicators

Multiple health indicators can be aggregated together with the CompositeHealthContributor class:

```java
@Component
public class CompositeHealthIndicator extends CompositeHealthContributor {
    public CompositeHealthIndicator(
        ApiHealthIndicator apiHealth,
        DatabaseHealthIndicator dbHealth
    ) {
        addContributor("api", apiHealth);
        addContributor("db", dbHealth);
    }
}
```

Configuring/Customizing Endpoints

Create Custom Endpoints

Custom read/write endpoints can be created with @Endpoint annotation:

```java
@Endpoint(id = "features")
@Component
public class FeatureToggleEndpoint {
    private final List<String> activeFeatures = new ArrayList<>();

    // This is exposed as a write endpoint
    @WriteOperation
    public void toggleFeature(String feature, boolean enable) {
        if (enable) activeFeatures.add(feature);
        else activeFeatures.remove(feature);
    }

    // This is exposed as a read endpoint
    @ReadOperation
    public List<String> features() { return activeFeatures; }
}
```


Configuring/Customizing Endpoints

Configure Security

Endpoints can be secured using the Spring Security library.

```java
@Endpoint(id = "admin")
@Component
public class AdminEndpoint {
    @ReadOperation
    @PreAuthorize("hasRole('ADMIN')")
    public String adminData() { return "Sensitive Info"; }
}
```


Integration with Monitoring Tools

Actuator endpoints are designed to integrate seamlessly with various monitoring and observability tools:

Prometheus & Grafana:
With Micrometer, Spring Boot Actuator can expose metrics in a Prometheus-friendly format. Grafana can then visualize these metrics for real-time dashboards.

Third-Party Tools:
Other systems like New Relic or Apidog can also integrate with actuator endpoints to provide advanced analytics, alerting, and automated responses to issues.

Actuator Do's And Don'ts

Do:

✅Restrict sensitive endpoints (e.g., /env, /heapdump) to authorized roles using Spring Security.

✅Expose only critical endpoints (e.g., health, metrics) and disable unused ones.

✅Encrypt Actuator traffic with HTTPS in production.

✅Use custom health checks for dependencies (e.g., databases, APIs).

✅Export metrics to monitoring tools (e.g., Prometheus, Grafana).

Actuator Do’s And Don’ts

Don't:

❌Never expose sensitive endpoints (/env, /heapdump) without authentication.

❌Avoid enabling all endpoints (management.endpoints.web.exposure.include=*) in production.

❌Don’t use default endpoint paths (e.g., /actuator); customize them.

❌Never ignore role-based access control (RBAC) for endpoints.

❌Avoid HTTP for Actuator communication; enforce HTTPS strictly.

Summary

Spring Boot Actuator is a powerful tool for monitoring and managing Spring Boot applications.

It offers built-in endpoints for health, metrics, and more, and allows customizations.

By following best practices and adhering to the dos and don'ts, the application's observability can be enhanced while keeping it secure.

Did You Find This Post Useful?

Stay Tuned for More Spring Related Posts Like This

LAHIRU
LIYANAPATHIRANA