Spring Actuator: Features and Best Practices

Table of Contents

Introduction

Spring Boot Actuator is a sub-project of Spring Boot that adds several production-ready features to help monitor and manage your Spring Boot application. It provides HTTP endpoints or JMX beans to interact with your application, offering insights into the application's health, metrics, environment properties, and more.

Key Features

- Health checks
- Metrics
- Auditing
- HTTP tracing
- Environment information
- Loggers configuration
- Thread dump

Enabling Spring Actuator

To enable Spring Actuator in your Spring Boot project, add the following dependency:

For Maven:

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

For Gradle:

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

Important Endpoints

By default, all Actuator endpoints except `/health` are disabled. To enable endpoints, use the following property in `application.properties`:

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

Key endpoints include:

- `/actuator/health`: Application health information
- `/actuator/info`: Application information
- `/actuator/metrics`: Application metrics
- `/actuator/env`: Environment properties
- `/actuator/loggers`: Logger configuration
- `/actuator/httptrace`: HTTP trace information
- `/actuator/threaddump`: Thread dump

Customization

Spring Actuator allows extensive customization:

1. **Custom Endpoints**: Create custom endpoints using `@Endpoint`, `@ReadOperation`, `@WriteOperation`, and `@DeleteOperation` annotations.
2. **Health Indicators**: Implement `HealthIndicator` interface to provide custom health checks.
3. **Info Contributors**: Implement `InfoContributor` interface to add custom information to the `/info` endpoint.
4. **Metrics**: Use `MeterRegistry` to register custom metrics.

Security Considerations

Actuator endpoints can expose sensitive information. Best practices for securing endpoints include:

1. Enable HTTPS
2. Use Spring Security to protect endpoints
3. Expose only necessary endpoints
4. Use environment-specific configurations

Example security configuration:

```
@Configuration
public class ActuatorSecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http.securityMatcher(EndpointRequest.toAnyEndpoint());
```

```
        http.authorizeHttpRequests((requests) ->
requests.anyRequest().hasRole("ENDPOINT_ADMIN"));
        http.httpBasic(withDefaults());
        return http.build();
    }

    @Bean
    public InMemoryUserDetailsManager userDetailsService(PasswordEncoder
passwordEncoder) {
        UserDetails user = User.withUsername("user")
                .password(passwordEncoder.encode("password"))
                .roles("USER")
                .build();

        UserDetails admin = User.withUsername("admin")
                .password(passwordEncoder.encode("admin"))
                .roles("USER", "ADMIN", "ENDPOINT_ADMIN")
                .build();

        return new InMemoryUserDetailsManager(user, admin);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        PasswordEncoder encoder =
PasswordEncoderFactories.createDelegatingPasswordEncoder();
        return encoder;
    }

}
```

Integration with Monitoring Tools

Spring Actuator integrates well with various monitoring tools:

1. **Micrometer**: Default metrics facade
2. **Prometheus**: Add `micrometer-registry-prometheus` dependency
3. **Graphite**: Add `micrometer-registry-graphite` dependency
4. **New Relic**: Add `micrometer-registry-new-relic` dependency

Best Practices

1. **Selective Endpoint Exposure**: Only expose necessary endpoints, especially in production.
2. **Custom Endpoints**: Create custom endpoints for application-specific monitoring needs.
3. **Health Checks**: Implement custom health indicators for critical dependencies.
4. **Metrics**: Use custom metrics to track application-specific data.
5. **Security**: Always secure Actuator endpoints, especially in production environments.

6. **Monitoring Integration**: Integrate with a monitoring tool for better insights and alerting.
7. **Documentation**: Document custom endpoints and metrics for your operations team.
8. **Testing**: Include Actuator endpoints in your test suite to ensure they're working correctly.
9. **Version Control**: Keep Actuator configurations in version control.
10. **Environment-Specific Config**: Use different Actuator configurations for different environments (dev, staging, prod).

By following these practices and leveraging Spring Actuator's features, you can significantly improve the observability and manageability of your Spring Boot applications in production environments.