Spring Actuator Features and Best Practices

1. Overview

Spring Actuator is a powerful module in Spring Boot that provides production-ready features to help monitor and manage applications. It offers built-in endpoints and metrics to track the application's health, configuration, and other operational aspects.

2. Key Features

1. **Health Checks**
    - **Endpoint**: /actuator/health
    - Provides detailed information about the application's health status, including checks for database connectivity, disk space, and other critical services.
    - Custom health indicators can be implemented to monitor specific aspects of your application.
2. **Application Information**
    - **Endpoint**: /actuator/info
    - Displays general information about the application, such as build version, description, and other metadata.
    - This information can be customized by adding properties to the application.properties.
3. **Metrics**
    - **Endpoint**: /actuator/metrics
    - Exposes metrics about the application's performance, such as memory usage, garbage collection, and custom metrics.
    - Metrics can be used to monitor application performance and troubleshoot issues.
4. **Environment**
    - **Endpoint**: /actuator/env
    - Shows properties and configuration values from the environment.
    - Useful for debugging and verifying configuration settings.
5. **Loggers**
    - **Endpoint**: /actuator/loggers
    - Provides access to logging levels of different loggers and allows for dynamic changes to logging levels.

6. **Thread Dumps**
    - **Endpoint**: /actuator/threaddump
    - Provides a snapshot of the current thread states, which can be used to diagnose performance issues and deadlocks.
7. **Dump**
    - **Endpoint**: /actuator/dump
    - Offers a dump of the application's thread stack traces for analysis.
8. **Shutdown**
    - **Endpoint**: /actuator/shutdown
    - Gracefully shuts down the application. This endpoint needs to be enabled explicitly and secured to avoid accidental shutdowns.

3. Best Practices

1. **Secure Actuator Endpoints**
    - **Authentication and Authorization**: Ensure that sensitive endpoints (like /actuator/health and /actuator/env) are protected using authentication and authorization mechanisms.
    - **Example**: Configure security settings in application.properties or application.yml to restrict access.

management.endpoints.web.exposure.include=health,info,metrics,loggers

management.endpoints.web.exposure.exclude=shutdown

management.endpoint.health.show-details=always

management.endpoint.health.roles=ADMIN

2. **Customizing Endpoints**
    - **Add Custom Endpoints**: Implement custom endpoints to expose additional application-specific information.
    - **Example**: Create a new endpoint using @Endpoint and @ReadOperation.
3. **Monitoring and Metrics**
    - **Use Micrometer**: Leverage Micrometer for advanced metrics and monitoring integration with tools like Prometheus, Grafana, and others.
    - **Example**: Configure Micrometer in application.properties or application.yml.

4. **Health Checks**
    - **Custom Health Indicators**: Implement custom health indicators for application-specific checks.
    - **Example**: Create a class that implements HealthIndicator and register it as a Spring Bean.
5. **Performance Optimization**
    - **Endpoint Exposure**: Limit the number of exposed endpoints to reduce potential security risks and performance overhead.
    - **Example**: Use management.endpoints.web.exposure.include and management.endpoints.web.exposure.exclude properties to control visibility.
6. **Testing Actuator Endpoints**
    - **Integration Tests**: Write integration tests to ensure that actuator endpoints are correctly exposed and return expected results.
    - **Example**: Use MockMvc to test endpoint responses.
7. **Graceful Shutdown**
    - **Proper Configuration**: If using the /actuator/shutdown endpoint, ensure that it is properly secured and used carefully in production environments.
    - **Example**: Disable or restrict this endpoint in production environments unless needed.
8. **Documentation and Training**
    - **Document Actuator Usage**: Document the purpose and usage of each exposed endpoint for team members and maintainers.
    - **Example**: Include endpoint information in internal documentation or developer guides.

4. Conclusion

Spring Actuator provides essential features for monitoring and managing Spring Boot applications. By following best practices for security, customization, and performance optimization, you can leverage Actuator to maintain a healthy and well-monitored application.