Spring Actuator is a powerful tool in the Spring Boot ecosystem that provides production-ready features to monitor and manage applications. It exposes various endpoints that allow you to gather insights into the application's health, metrics, and more, without modifying application code.

Key Features of Spring Actuator

1. **Health Checks**

   - The `/actuator/health` endpoint provides a summary of the application's health status, allowing you to integrate it with monitoring tools or orchestrators like Kubernetes to assess the health of your application.
   - Custom health indicators can also be implemented to include application-specific health information.

   Example:

```
@Component
public class CustomHealthIndicator implements HealthIndicator
{
    @Override
    public Health health() {
        // custom health check logic
        return Health.up().withDetail("status", "Everything
is OK!").build();
    }
}
```

2. **Metrics**

   - The `/actuator/metrics` endpoint exposes various metrics related to CPU, memory, garbage collection, and more. This allows you to track performance and identify bottlenecks.
   - Micrometer integration allows you to export metrics to various monitoring systems such as Prometheus, Datadog, or Grafana.

   Example metrics:

- `jvm.memory.used`: Tracks memory usage.
- `system.cpu.usage`: Tracks CPU usage.

3. **Info**
   - The `/actuator/info` endpoint can be customized to expose information about the application, such as build details, version, or any custom information.
   - You can define static information in the `application.properties` file:

```
info.app.name=Spring Actuator Application
info.app.description=This is a demo application for Actuator
info.app.version=1.0.0
```

4. **Environment and Config Properties**
   - The `/actuator/env` and `/actuator/configprops` endpoints allow you to inspect the environment variables and the configuration properties loaded by the application, making it easier to debug configuration issues.

   Best Practice: Secure these endpoints as they expose sensitive information.
5. **Thread Dump and Heap Dump**
   - The `/actuator/threaddump` and `/actuator/heapdump` endpoints can provide insights into the current state of application threads and heap memory, making it easier to debug performance and memory issues.
6. **Custom Endpoints**
   - You can create custom Actuator endpoints for specific use cases. For instance, if you want to expose application-specific operational data, you can define custom endpoints using the `@Endpoint` annotation.

   Example:

```
@Component
@Endpoint(id = "custom")
public class AppStatusEndpoint {
    @ReadOperation
    public String getAppStatus() {
        return "Application is running smoothly!";
```

```
    }
}
```

7. **Auditing**
    - Spring Actuator integrates with the Spring Security Auditing framework to track security-related events, such as user authentication and access control decisions.
    - The `/actuator/auditevents` endpoint provides access to the audit events.
8. **HTTP Tracing**
    - The `/actuator/httptrace` endpoint provides insights into the recent HTTP request and response exchanges, which can help you monitor traffic patterns and debug network issues.

Best Practices for Using Spring Actuator

1. **Limit Exposed Endpoints in Production**
    - In production environments, it is best to limit which Actuator endpoints are exposed to the outside world. Only expose the endpoints that are essential for monitoring.

    Example:

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

2. **Secure Sensitive Endpoints**
    - Many Actuator endpoints provide sensitive information, such as environment variables and configuration properties. Use Spring Security to protect these endpoints.

    Example: Securing all Actuator endpoints using basic authentication:

```
management.endpoints.web.exposure.include=*
spring.security.user.name=admin
spring.security.user.password=secretpassword
```

3. **Customize Health Indicators**

    - Tailor health checks to your specific business logic by creating custom health indicators. This ensures that your application health reflects both system and application-specific metrics.

    Example:

```
@Component
public class DatabaseHealthIndicator implements
HealthIndicator {
    @Override
    public Health health() {
        // check database connectivity
        return Health.up().withDetail("database",
"reachable").build();
    }
}
```

4. **Leverage External Monitoring Tools**

    - Integrate Actuator with tools like Prometheus, Grafana, or Datadog for advanced monitoring and visualization. Micrometer provides an easy way to push Actuator metrics to various monitoring systems.

    Example: Enabling Prometheus in `application.properties`:

```
management.metrics.export.prometheus.enabled=true
management.metrics.export.prometheus.endpoint=/prometheus
```

5. **Enable Detailed Health Information for Authorized Users**
    - By default, Spring Actuator only exposes summary health information. You can configure it to show detailed health information to authorized users:

```
management.endpoint.health.show-details=when_authorized
```

6. **Customize Actuator Path**
    - For security and organizational purposes, you may want to customize the base path for Actuator endpoints:

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

7. **Monitor Microservices with Distributed Tracing**

    - If you are running a microservices architecture, use Actuator's tracing capabilities with tools like Zipkin or Sleuth for distributed tracing, which helps monitor requests as they flow through multiple services.

    Example:

```
spring.zipkin.enabled=true
spring.sleuth.sampler.probability=1.0
```

8. **Use Actuator in Combination with Spring Boot Admin**
    - Spring Boot Admin can be used as a central management interface for Spring Boot applications with Actuator enabled. This provides a dashboard to monitor and manage multiple applications from a single point.