Spring Boot Actuator

Monitoraggio, Metriche e Endpoint Custom

Scopri come monitorare e gestire le tue applicazioni Spring Boot in produzione con strumenti potenti e flessibili.

Cos'è Spring Boot Actuator?

Un modulo essenziale che fornisce funzionalità production-grade per il monitoraggio e la gestione delle applicazioni.

Consente di esporre endpoint HTTP e JMX per ottenere informazioni dettagliate sullo stato del sistema.

- Monitoraggio in tempo reale dello stato
- Accesso a metriche dettagliate (JVM, sistema)
- Health checks automatici e personalizzabili
- Endpoint configurabili e sicuri
- Integrazione con sistemi di monitoring esterni

Configurazione Base di Actuator

Per abilitare Spring Boot Actuator, è necessario aggiungere la seguente dipendenza al file di configurazione del progetto (Maven):

pom.xml

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

Una volta aggiunta la dipendenza, Actuator espone automaticamente gli endpoint con il prefisso /actuator.
Ad esempio: **/actuator/health**

Configurazione dei Health Indicators

Controllare il livello di dettaglio

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

never

Default. Non mostra mai i dettagli dei componenti, solo lo stato aggregato (UP/DOWN).

when-authorized

Mostra i dettagli completi solo agli utenti autenticati e autorizzati.

always

Mostra sempre i dettagli a tutti gli utenti (pubblico). Usare con cautela.

Disabilitare un indicatore specifico

È possibile disattivare singoli indicatori se non necessari o se causano falsi positivi.

```
management.health.diskspace.enabled=false
```

Personalizzazione dei Percorsi

È possibile personalizzare il prefisso e i percorsi degli endpoint tramite proprietà di configurazione per adattare Actuator alle esigenze dell'infrastruttura.

Personalizzare il prefisso base

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

`/actuator/health` → `/manage/health`

Remappare endpoint specifici

```
management.endpoints.web.path-mapping.health=healthcheck
```

`/manage/health` → `/manage/healthcheck`

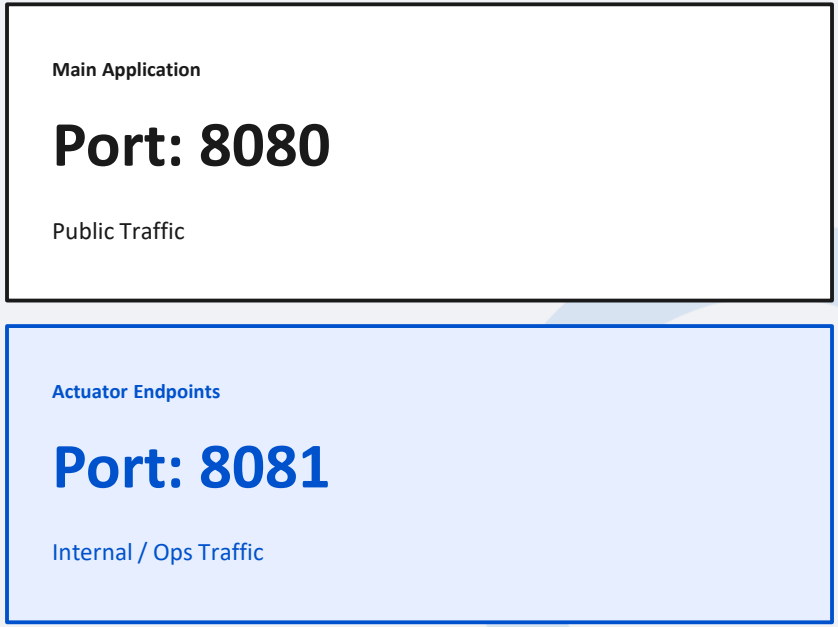
Configurazione della Porta di Gestione

Per ambienti cloud e data center, è spesso utile esporre gli endpoint di gestione su una porta diversa da quella dell'applicazione principale.

Questo migliora la sicurezza e consente una gestione più granulare del traffico (es. firewall rules diverse).

```
management.server.port=8081
```

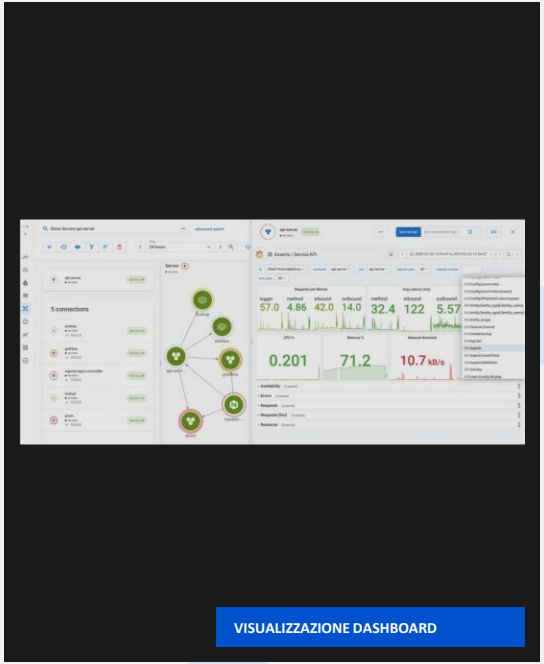
Nota: È anche possibile configurare SSL specifico per il server di gestione, consentendo scenari come HTTPS per l'applicazione principale e HTTP per la gestione interna.



Health Checks Fondamenti

L'endpoint **/actuator/health** mostra lo stato dell'applicazione. Spring Boot usa **HealthIndicator** predefiniti per verificare componenti critici come database e spazio disco.


```
{
  "status": "UP",
  "components": {
    "db": {"status": "UP", "details": {"database": "PostgreSQL"}},
    "diskSpace": {"status": "UP", "details": {"total": 499963170816, "free": 134414831616, "threshold": 10485760}}
  }
}
```

Health Indicators Predefiniti

Spring Boot registra automaticamente diversi health indicator in base alle dipendenze rilevate nel classpath dell'applicazione:

PingHealthIndicator

Verifica semplicemente che l'applicazione sia in esecuzione e risponda.

DiskSpaceHealthIndicator

Controlla lo spazio disponibile su disco e avvisa se scende sotto una soglia.

DataSourceHealthIndicator

Verifica la validità della connessione al database relazionale (se presente).

RedisHealthIndicator

Controlla lo stato della connessione al server Redis.

RabbitHealthIndicator

Verifica la connettività con il broker di messaggi RabbitMQ.

ElasticsearchHealthIndicator

Verifica la connessione al cluster Elasticsearch.

CassandraHealthIndicator

Controlla lo stato del database Cassandra.

MailHealthIndicator

Verifica la connessione al server di posta SMTP.

Ogni indicatore fornisce dettagli specifici (es. versione DB, spazio libero) nel payload JSON della risposta.

Custom Health Indicators

Interfaccia

Implementare l'interfaccia **HealthIndicator** e annotare la classe con @Component.

Logica

Eseguire i controlli necessari nel metodo health() (es. connessione DB, ping servizio esterno).

Risposta

Utilizzare i builder Health.up() o Health.down() per restituire lo stato e aggiungere dettagli.

```
@Component
public class DatabaseConnectionHealthIndicator implements HealthIndicator {
    @Override
    public Health health() {
        try {
            // Verifica la connessione al database
            boolean isConnected = checkDatabaseConnection();
            if (isConnected) {
                return Health.up()
                        .withDetail("database", "Connected")
                        .withDetail("responseTime", "45ms")
                        .build();
            } else {
                return Health.down()
                        .withDetail("database", "Connection failed")
                        .build();
            }
        } catch (Exception e) {
                return Health.down(e)
                        .withDetail("error", "Check failed")
                        .build();
        }
    }
}
```

Health Groups

Spring Boot permette di organizzare gli health indicator in gruppi logici. Questo è fondamentale per creare probe specifici per orchestratori come Kubernetes.

```
management.endpoint.health.group.readiness.include=db,diskSpace
management.endpoint.health.group.liveness.include=ping
```

Readiness Probe

`/actuator/health/readiness`

Verifica se l'applicazione è pronta ad accettare traffico. Se fallisce, il load balancer smette di inviare richieste.

Checks: Database, Disk Space

Liveness Probe

`/actuator/health/liveness`

Verifica se l'applicazione è ancora in esecuzione. Se fallisce, l'orchestratore riavvia il container.

Checks: Ping (Internal state)

Endpoint Predefiniti di Actuator

Endpoint	Descrizione
/actuator/health	Stato di salute dell'applicazione
/actuator/metrics	Metriche dell'applicazione
/actuator/beans	Elenco di tutti i bean Spring
/actuator/env	Proprietà di configurazione
/actuator/info	Informazioni sull'applicazione
/actuator/loggers	Configurazione dei logger
/actuator/threaddump	Dump dei thread attivi
/actuator/heapdump	Dump della memoria heap

Esposizione degli Endpoint

Per impostazione predefinita, solo l'endpoint health è esposto via HTTP. È necessario configurare esplicitamente quali altri endpoint rendere accessibili.

Includere Endpoint

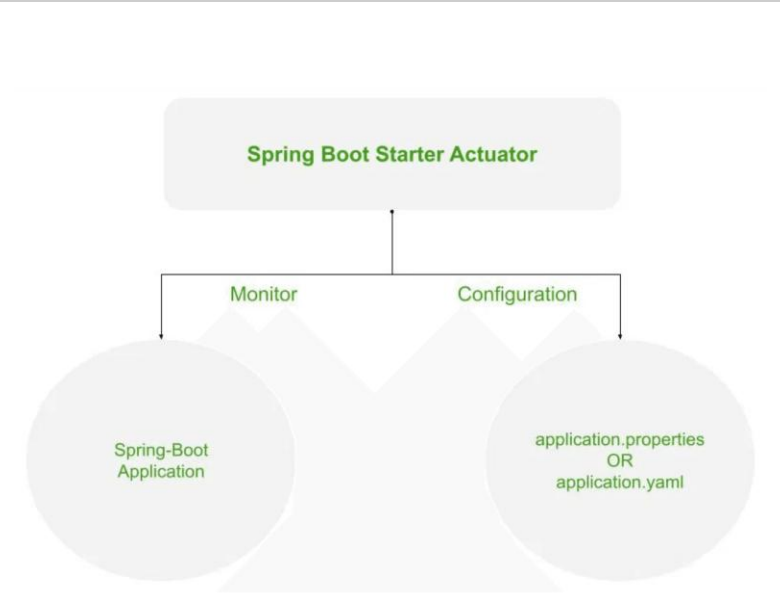
```
management.endpoints.web.exposure.include=*

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

Escludere Endpoint

```
management.endpoints.web.exposure.exclude=env,beans
```

Attenzione: L'uso del wildcard * espone TUTTI gli endpoint. Assicurarsi di proteggerli con Spring Security.



Flusso di Configurazione e Monitoraggio

Metriche Introduzione

L'endpoint **/actuator/metrics** fornisce accesso a metriche dettagliate dell'applicazione, inclusi dati su JVM, sistema e richieste HTTP.

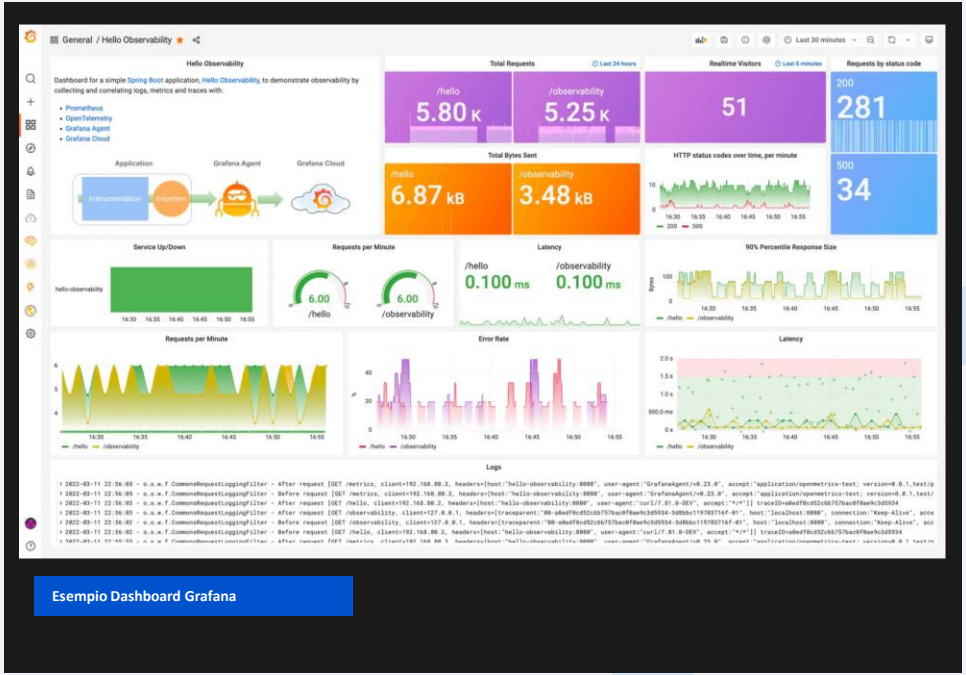
Micrometer

Una facade per le metriche che supporta numerosi sistemi di monitoring esterni (Prometheus, Datadog, ecc.).

GET/actuator/metrics

GET/actuator/metrics/jvm.memory.used

GET/actuator/metrics/http.server.requests



Esempio Dashboard Grafana

Tipi di Metriche Disponibili

JVM Metrics

Dati su utilizzo memoria, garbage collection, thread attivi e caricamento classi.

`jvm.memory.used, jvm.gc.pause`

HTTP Metrics

Statistiche su richieste ricevute, tempi di risposta, codici di stato e latenza.

`http.server.requests`

System Metrics

Informazioni su utilizzo CPU, file system, uptime e carico del sistema operativo.

`system.cpu.usage, disk.free`

Logger Metrics

Conteggio degli eventi di log raggruppati per livello (INFO, ERROR, WARN).

`logback.events`

Application Startup

Metriche relative al tempo di avvio dell'applicazione e inizializzazione dei bean.

`application.started.time`

Task Execution

Monitoraggio di executor service, thread pool e task schedulati.

`executor.active, executor.completed`

Endpoint Custom Fondamenti

Quando le metriche e gli health check non bastano, è possibile creare endpoint personalizzati per esporre logiche di dominio o funzionalità di gestione specifiche.

```
@Endpoint(id = "features")
```

Identificativo & URL

Il parametro id definisce l'identificativo dell'endpoint. Questo viene mappato automaticamente nell'URL esposto.

Esempio: /actuator/features

Technology Agnostic

L'annotazione @Endpoint rende la funzionalità disponibile automaticamente sia via **HTTP** che via **JMX**, senza bisogno di codice specifico per il protocollo.

Creazione di Endpoint Custom

```java
@Component
@Endpoint(id = "systemStatus")
public class SystemStatusEndpoint {
  @ReadOperation
  public Map<String, Object> getStatus() {
    Map<String, Object> status = new HashMap<>();
    status.put("ready", true);
    status.put("load", "low");
    return status;
  }


  @ReadOperation
  public String getDetail(@Selector String key) {
    return "Detail for " + key;
  }
}
```

@Endpoint(id = "...")

Definisce l'endpoint. L'ID determina l'URL di accesso (es. /actuator/systemStatus).

@ReadOperation

Espone il metodo tramite chiamata HTTP **GET**. Restituisce automaticamente JSON.

@Selector

Cattura una parte del percorso URL come variabile (simile a @PathVariable).

```
GET /actuator/systemStatus
GET /actuator/systemStatus/{key}
```

Operazioni su Endpoint Custom

@ReadOperation

GET

Utilizzata per recuperare informazioni. Deve essere idempotente e non modificare lo stato.

@WriteOperation

POST

Utilizzata per eseguire azioni che modificano lo stato dell'applicazione (es. reset cache).

@DeleteOperation

DELETE

Utilizzata per rimuovere risorse o resettare configurazioni specifiche.

Parametri negli Endpoint Custom

L'annotazione **@Selector** permette di catturare parti del percorso URL e passarle come argomenti ai metodi dell'endpoint, simile a @PathVariable nei controller REST.

HTTP Request URL		Java Method Argument
/actuator/features/	→	@Selector String

```
@ReadOperation public Feature getFeature(@Selector String name) { return featureService.findByName(name); }
```

Nota: È possibile utilizzare più annotazioni @Selector per creare percorsi gerarchici (es. /actuator/features/{name}/{version}).