Securing Actuator Endpoints and Creating Customizations

Dustin Schultz
PRINCIPAL SOFTWARE ENGINEER

@schultzdustin dustin.schultz.io dustin@schultz.io

Overview

Enabling and disabling endpoints

Exposing endpoints

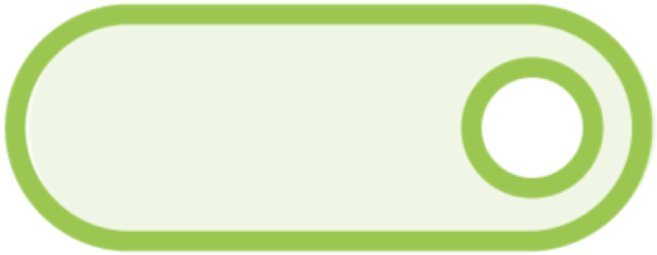
Enabling vs. exposing

Securing endpoints

Customizing existing endpoints

Implementing custom endpoints

Enabling and Disabling Actuator Endpoints



Individual endpoints can be enabled or disabled

Controls the creation of the endpoint

By default, all actuator endpoints, *except shutdown*, are enabled

management.endpoint.<NAME>.enabled = true

management.endpoint.<NAME>.enabled = false

Enabling/Disabling Specific Endpoints

Configure application.properties **or** application.yml

Replace <NAME> **with ID of endpoint**

https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-features.html#production-ready-endpoints

management.endpoints.enabled-by-default = false

management.endpoint.health.enabled = true

management.endpoint.loggers.enabled = true

management.endpoint.env.enabled = true

Disabled by Default

All endpoints enabled-by-default, can be turned off

Useful when you want only a few endpoints enabled

Exposing Actuator Endpoints

Exposing an endpoint makes it available for consumption

Individual endpoints can be included / excluded for exposure

Supported Methods for Exposing Endpoints

All endpoints *are* exposed,
via JMX, by default.

Most endpoints are *not* exposed *via HTTP* by default, only the health and info endpoints.

Use caution when exposing endpoints, particularly via HTTP

Ensure endpoints are secured

Controlling Which Endpoints Are and Aren't Exposed

application.properties

```
management.endpoint.<PROTOCOL>.expose.include=
management.endpoint.<PROTOCOL>.expose.exclude=
```

```
# included http endpoints
management.endpoint.http.expose.include=metrics

# excluded jmx endpoints
management.endpoint.jmx.expose.exclude=beans

# Enable ALL http endpoints
management.endpoint.http.expose.include=*
```

Note: Surround the asterisk with double quotes if configuring via application.yml ("*")

Enabling vs. Exposing Endpoints

enabling != exposing

Enabling and Exposing: The Differences

Enabling	Exposing
Controls creation	Controls consumption
Almost all endpoints are enabled by default	Almost all endpoints are *not* exposed via HTTP by default
Efficiency purposes	Security purposes

Demo

Enabling and exposing endpoints

Securing Actuator Endpoints

Endpoint security is automatically configured if Spring Security is on the classpath

- Adds HTTP basic authentication to all endpoints except
  - Health
  - Info

Securing Endpoints with Spring Security and Maven

pom.xml

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

Securing Endpoints with Spring Security and Gradle

build.gradle

```
dependencies {
   compile("org.springframework.boot:spring-boot-starter-security")
}
```

INFO 1234 --- [main] b.a.s.AuthenticationManagerConfiguration :

Using default security password: <random string>

Note: The default username is ‘user’

spring.security.user.name=<username>
spring.security.user.password=<password>

Credentials

Default credentials will automatically be configured

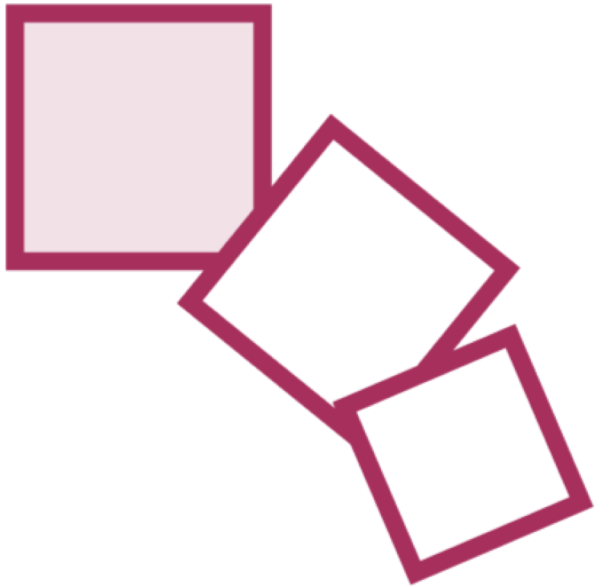
Credentials are configurable

Endpoint Security Is Completely Customizable

```
@Configuration(proxyBeanMethods = false)
public class ActuatorSecurity extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http
        .requestMatcher(EndpointRequest.toAnyEndpoint())
        .authorizeRequests( (req) -> req.anyRequest().hasRole("ADMIN"));

        http.httpBasic();

    }


}
```

Customizing Built-in Endpoints

What Can Be Customized?



A subset of endpoints support customizations

- Health endpoint
- Info endpoint
- Metrics endpoint

Customizing the Health Endpoint

Adding additional checks and details to /health

Adding Custom Health Checks and Details to the Health Endpoint

Create and register a new bean that implements HealthIndicator

Implement the health() **method. Return a** Health **object with the appropriate status**

Creating a Custom HealthIndicator

```
@Component
public class FooServiceHealthIndicator implements HealthIndicator {

    @Override
    public Health health() {
      // perform a custom health check

      // inspect the status

      // healthy
      return Health.up().build();
    }

}
```

Creating a Custom HealthIndicator

```
@Component
public class FooServiceHealthIndicator implements HealthIndicator {

    @Override
    public Health health() {
      // perform a custom health check

      // inspect the status

      // unhealthy
      return Health.down().build();
    }

}
```

Creating a Custom HealthIndicator with Details

```
@Component
public class FooServiceHealthIndicator implements HealthIndicator {

    @Override
    public Health health() {
      // perform a custom health check

      // inspect the status

      // unhealthy
      return Health.down()
              .withDetail("response_code", "...")
              .withDetail("response_ms", "...")
              .withDetail("num_retries", "...")
              .build();

}
```

Customizing the Info Endpoint

Adding additional information points to /info

Adding Custom Data to the Info Endpoint

Create and register a new bean that implements InfoContributor

Implement the contribute() **method and add data with the provided** Info.Builder

Creating a Custom InfoContributor

```
@Component
public class ProjectInfoContributor implements InfoContributor {

    @Override
    public void contribute(Info.Builder builder) {
      // add new info
      builder.withDetail("project_name", "...")
             .withDetail("owned_by_team", "...")
             .withDetail("point_of_contact", "...");
    }

}
```

Customizing the Metrics Endpoint

Adding new custom metrics

Adding Custom Metrics to the Metrics Endpoint

Inject a MeterRegistry into the class that generates the metrics and register a new metric

Use the metric (e.g. timer.record(...) or counter.increment())

Exposing Custom Metrics Using the MeterRegistry

```
@Service
public class ComplexService {

  private Timer timer;

  public ComplexService(MeterRegistry registry) {
    // give the timer a name
    timer = registry.timer("long.running.op.timer")
  }

  public void longRunningOperation() {
    timer.record(() -> {
      // a long running operation
    })
  }
}
```

Demo

Writing a custom HealthIndicator

Creating Your Own Actuator Endpoints

Implementing a Custom Endpoint

Create and register a new bean annotated with @Endpoint

Annotate methods with one of @ReadOperation, @WriteOperation, @DeleteOperation

Implementing a Custom Actuator Endpoint

```
@Component
@Endpoint(id = "container")
public class DockerEndpoint {

  @ReadOperation
  public String foo() {
    // Gather and return information (e.g. get info about the docker container)
  }

  @WriteOperation
  public void bar() {
    // Do some action (e.g. restart the docker container)
  }
}
```

Demo

Implementing a custom actuator endpoint

Summary

Enable, disable, expose, and secure endpoints

Customizing built-in endpoints

Creating custom actuator endpoints