

# Spring Boot Actuator Web API Documentation

Andy Wilkinson

# Table of Contents

1. Overview .....	2
1.1. URLs .....	2
1.2. Timestamps .....	2
2. Audit Events ( <b>auditevents</b> ) .....	3
2.1. Retrieving Audit Events .....	3
2.1.1. Query Parameters .....	3
2.1.2. Response Structure .....	3
3. Beans ( <b>beans</b> ) .....	5
3.1. Retrieving the Beans .....	5
3.1.1. Response Structure .....	6
4. Caches ( <b>caches</b> ) .....	8
4.1. Retrieving All Caches .....	8
4.1.1. Response Structure .....	8
4.2. Retrieving Caches by Name .....	9
4.2.1. Query Parameters .....	9
4.2.2. Response Structure .....	9
4.3. Evict All Caches .....	10
4.4. Evict a Cache by Name .....	10
4.4.1. Request Structure .....	10
5. Conditions Evaluation Report ( <b>conditions</b> ) .....	11
5.1. Retrieving the Report .....	11
5.1.1. Response Structure .....	12
6. Configuration Properties ( <b>configprops</b> ) .....	14
6.1. Retrieving the <b>@ConfigurationProperties</b> Bean .....	14
6.1.1. Response Structure .....	15
7. Environment ( <b>env</b> ) .....	16
7.1. Retrieving the Entire Environment .....	16
7.1.1. Response Structure .....	17
7.2. Retrieving a Single Property .....	18
7.2.1. Response Structure .....	18
8. Flyway ( <b>flyway</b> ) .....	20
8.1. Retrieving the Migrations .....	20
8.1.1. Response Structure .....	20
9. Health ( <b>health</b> ) .....	22
9.1. Retrieving the Health of the application .....	22
9.1.1. Response Structure .....	23
9.2. Retrieving the Health of a component .....	24
9.2.1. Response Structure .....	24

9.3. Retrieving the Health of a component instance .....	25
9.3.1. Response Structure .....	25
10. Heap Dump ( <b>heapdump</b> ) .....	26
10.1. Retrieving the Heap Dump .....	26
11. HTTP Trace ( <b>httptrace</b> ) .....	27
11.1. Retrieving the Traces .....	27
11.1.1. Response Structure .....	27
12. Info ( <b>info</b> ) .....	29
12.1. Retrieving the Info .....	29
12.1.1. Response Structure .....	29
<b>build</b> Response Structure .....	29
<b>git</b> Response Structure .....	30
13. Spring Integration graph ( <b>integrationgraph</b> ) .....	31
13.1. Retrieving the Spring Integration graph .....	31
13.1.1. Response Structure .....	32
13.2. Rebuilding the Spring Integration graph .....	32
14. Liquibase ( <b>liquibase</b> ) .....	34
14.1. Retrieving the Changes .....	34
14.1.1. Response Structure .....	34
15. Log File ( <b>logfile</b> ) .....	36
15.1. Retrieving the Log File .....	36
15.2. Retrieving Part of the Log File .....	38
16. Loggers ( <b>loggers</b> ) .....	39
16.1. Retrieving All Loggers .....	39
16.1.1. Response Structure .....	39
16.2. Retrieving a Single Logger .....	40
16.2.1. Response Structure .....	40
16.3. Setting a Log Level .....	40
16.3.1. Request Structure .....	40
16.4. Clearing a Log Level .....	41
17. Mappings ( <b>mappings</b> ) .....	42
17.1. Retrieving the Mappings .....	42
17.1.1. Response Structure .....	45
17.1.2. Dispatcher Servlets Response Structure .....	45
17.1.3. Servlets Response Structure .....	47
17.1.4. Servlet Filters Response Structure .....	47
17.1.5. Dispatcher Handlers Response Structure .....	47
18. Metrics ( <b>metrics</b> ) .....	50
18.1. Retrieving Metric Names .....	50
18.1.1. Response Structure .....	50
18.2. Retrieving a Metric .....	50

18.2.1. Query Parameters .....	51
18.2.2. Response structure .....	51
18.3. Drilling Down .....	52
19. Prometheus ( <b>prometheus</b> ) .....	53
19.1. Retrieving the Metrics.....	53
20. Scheduled Tasks ( <b>scheduledtasks</b> ) .....	55
20.1. Retrieving the Scheduled Tasks .....	55
20.1.1. Response Structure .....	55
21. Sessions ( <b>sessions</b> ).....	57
21.1. Retrieving Sessions .....	57
21.1.1. Query Parameters .....	58
21.1.2. Response Structure .....	58
21.2. Retrieving a Single Session .....	58
21.2.1. Response Structure .....	59
21.3. Deleting a Session .....	59
22. Shutdown ( <b>shutdown</b> ).....	60
22.1. Shutting Down the Application .....	60
22.1.1. Response Structure .....	60
23. Thread Dump ( <b>threaddump</b> ) .....	61
23.1. Retrieving the Thread Dump.....	61
23.1.1. Response Structure .....	64

This API documentation describes Spring Boot Actuators web endpoints.

# Chapter 1. Overview

Before you proceed, you should read the following topics:

- [URLs](#)
- [Timestamps](#)

## 1.1. URLs

By default, all web endpoints are available beneath the path `/actuator` with URLs of the form `/actuator/{id}`. The `/actuator` base path can be configured by using the `management.endpoints.web.base-path` property, as shown in the following example:

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

The preceding `application.properties` example changes the form of the endpoint URLs from `/actuator/{id}` to `/manage/{id}`. For example, the URL `info` endpoint would become `/manage/info`.

## 1.2. Timestamps

All timestamps that are consumed by the endpoints, either as query parameters or in the request body, must be formatted as an offset date and time as specified in [ISO 8601](#).

# Chapter 2. Audit Events (auditevents)

The `auditevents` endpoint provides information about the application's audit events.

## 2.1. Retrieving Audit Events

To retrieve the audit events, make a `GET` request to `/actuator/auditevents`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/auditevents?principal=alice&after=2018-11-30T08%3A30%3A01.643Z&type=logout' -i -X GET
```

The preceding example retrieves `logout` events for the principal, `alice`, that occurred after 09:37 on 7 November 2017 in the UTC timezone. The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 121

{
  "events" : [ {
    "timestamp" : "2018-11-30T08:30:01.645Z",
    "principal" : "alice",
    "type" : "logout"
  } ]
}
```

### 2.1.1. Query Parameters

The endpoint uses query parameters to limit the events that it returns. The following table shows the supported query parameters:

Parameter	Description
<code>after</code>	Restricts the events to those that occurred after the given time. Optional.
<code>principal</code>	Restricts the events to those with the given principal. Optional.
<code>type</code>	Restricts the events to those with the given type. Optional.

### 2.1.2. Response Structure

The response contains details of all of the audit events that matched the query. The following table describes the structure of the response:

<b>Path</b>	<b>Type</b>	<b>Description</b>
<code>events</code>	<code>Array</code>	An array of audit events.
<code>events[].timestamp</code>	<code>String</code>	The timestamp of when the event occurred.
<code>events[].principal</code>	<code>String</code>	The principal that triggered the event.
<code>events[].type</code>	<code>String</code>	The type of the event.



# Chapter 3. Beans (beans)

The `beans` endpoint provides information about the application's beans.

## 3.1. Retrieving the Beans

To retrieve the beans, make a `GET` request to `/actuator/beans`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/beans' -i -X GET
```

The resulting response is similar to the following:

HTTP/1.1 200 OK

Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8

Content-Length: 1128

```
{
  "contexts" : {
    "application" : {
      "beans" : {
        "defaultServletHandlerMapping" : {
          "aliases" : [ ],
          "scope" : "singleton",
          "type" : "org.springframework.web.servlet.HandlerMapping",
          "resource" : "class path resource
[org/springframework/boot/autoconfigure/web/servlet/WebMvcAutoConfiguration$EnableWebM
vcConfiguration.class]",
          "dependencies" : [ ]
        },

"org.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguration"
: {
          "aliases" : [ ],
          "scope" : "singleton",
          "type" :
"org.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguration$$
EnhancerBySpringCGLIB$$e861bdb5",
          "dependencies" : [ ]
        },

"org.springframework.boot.autoconfigure.web.servlet.DispatcherServletAutoConfiguration
" : {
          "aliases" : [ ],
          "scope" : "singleton",
          "type" :
"org.springframework.boot.autoconfigure.web.servlet.DispatcherServletAutoConfiguration
$$EnhancerBySpringCGLIB$$d60da039",
          "dependencies" : [ ]
        }
      }
    }
  }
}
```

### 3.1.1. Response Structure

The response contains details of the application's beans. The following table describes the structure of the response:

Path	Type	Description
contexts	Object	Application contexts keyed by id.
contexts.*.parentId	String	Id of the parent application context, if any.
contexts.*.beans	Object	Beans in the application context keyed by name.
contexts.*.beans.*.aliases	Array	Names of any aliases.
contexts.*.beans.*.scope	String	Scope of the bean.
contexts.*.beans.*.type	String	Fully qualified type of the bean.
contexts.*.beans.*.resource	String	Resource in which the bean was defined, if any.
contexts.*.beans.*.dependencies	Array	Names of any dependencies.

# Chapter 4. Caches (caches)

The `caches` endpoint provides access to the application's caches.

## 4.1. Retrieving All Caches

To retrieve the application's caches, make a `GET` request to `/actuator/caches`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/caches' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 435

{
  "cacheManagers" : {
    "anotherCacheManager" : {
      "caches" : {
        "countries" : {
          "target" : "java.util.concurrent.ConcurrentHashMap"
        }
      }
    },
    "cacheManager" : {
      "caches" : {
        "cities" : {
          "target" : "java.util.concurrent.ConcurrentHashMap"
        },
        "countries" : {
          "target" : "java.util.concurrent.ConcurrentHashMap"
        }
      }
    }
  }
}
```

### 4.1.1. Response Structure

The response contains details of the application's caches. The following table describes the structure of the response:

Path	Type	Description
<code>cacheManagers</code>	Object	Cache managers keyed by id.

Path	Type	Description
<code>cacheManagers.*.caches</code>	Object	Caches in the application context keyed by name.
<code>cacheManagers.*.caches.*.target</code>	String	Fully qualified name of the native cache.

## 4.2. Retrieving Caches by Name

To retrieve a cache by name, make a `GET` request to `/actuator/caches/{name}`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/caches/cities' -i -X GET
```

The preceding example retrieves information about the cache named `cities`. The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 113

{
  "target" : "java.util.concurrent.ConcurrentHashMap",
  "name" : "cities",
  "cacheManager" : "cacheManager"
}
```

### 4.2.1. Query Parameters

If the requested name is specific enough to identify a single cache, no extra parameter is required. Otherwise, the `cacheManager` must be specified. The following table shows the supported query parameters:

Parameter	Description
<code>cacheManager</code>	Name of the cacheManager to qualify the cache. May be omitted if the cache name is unique.

### 4.2.2. Response Structure

The response contains details of the requested cache. The following table describes the structure of the response:

Path	Type	Description
<code>name</code>	String	Cache name.
<code>cacheManager</code>	String	Cache manager name.

Path	Type	Description
<code>target</code>	<code>String</code>	Fully qualified name of the native cache.

### 4.3. Evict All Caches

To clear all available caches, make a `DELETE` request to `/actuator/caches` as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/caches' -i -X DELETE
```

### 4.4. Evict a Cache by Name

To evict a particular cache, make a `DELETE` request to `/actuator/caches/{name}` as shown in the following curl-based example:

```
$ curl
'http://localhost:8080/actuator/caches/countries?cacheManager=anotherCacheManager' -i
-X DELETE
```



As there are two caches named `countries`, the `cacheManager` has to be provided to specify which `Cache` should be cleared.

#### 4.4.1. Request Structure

If the requested name is specific enough to identify a single cache, no extra parameter is required. Otherwise, the `cacheManager` must be specified. The following table shows the supported query parameters:

Parameter	Description
<code>cacheManager</code>	Name of the <code>cacheManager</code> to qualify the cache. May be omitted if the cache name is unique.

# Chapter 5. Conditions Evaluation Report (conditions)

The `conditions` endpoint provides information about the evaluation of conditions on configuration and auto-configuration classes.

## 5.1. Retrieving the Report

To retrieve the report, make a `GET` request to `/actuator/conditions`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/conditions' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 3259

{
  "contexts" : {
    "application" : {
      "positiveMatches" : {
        "EndpointAutoConfiguration#endpointOperationParameterMapper" : [ {
          "condition" : "OnBeanCondition",
          "message" : "@ConditionalOnMissingBean (types:
org.springframework.boot.actuate.endpoint.invoke.ParameterValueMapper; SearchStrategy:
all) did not find any beans"
        } ],
        "EndpointAutoConfiguration#endpointCachingOperationInvokerAdvisor" : [ {
          "condition" : "OnBeanCondition",
          "message" : "@ConditionalOnMissingBean (types:
org.springframework.boot.actuate.endpoint.invoke.cache.CachingOperationInvokerAdvisor
; SearchStrategy: all) did not find any beans"
        } ],
        "WebEndpointAutoConfiguration" : [ {
          "condition" : "OnWebApplicationCondition",
          "message" : "@ConditionalOnWebApplication (required) found 'session' scope"
        } ]
      },
      "negativeMatches" : {
        "WebFluxEndpointManagementContextConfiguration" : {
          "notMatched" : [ {
            "condition" : "OnWebApplicationCondition",
            "message" : "not a reactive web application"
          } ],
          "matched" : [ {
```

```

        "condition" : "OnClassCondition",
        "message" : "@ConditionalOnClass found required classes
'org.springframework.web.reactive.DispatcherHandler',
'org.springframework.http.server.reactive.HttpHandler'"
    } ]
},
"GsonHttpMessageConvertersConfiguration.GsonHttpMessageConverterConfiguration"
: {
    "notMatched" : [ {
        "condition" :
"GsonHttpMessageConvertersConfiguration.PreferGsonOrJacksonAndJsonbUnavailableCondition",
        "message" : "AnyNestedCondition 0 matched 2 did not; NestedCondition on
GsonHttpMessageConvertersConfiguration.PreferGsonOrJacksonAndJsonbUnavailableCondition
.JacksonJsonbUnavailable NoneNestedConditions 1 matched 1 did not; NestedCondition on
GsonHttpMessageConvertersConfiguration.JacksonAndJsonbUnavailableCondition.JsonbPreferred
@ConditionalOnProperty (spring.http.converters.preferred-json-mapper=jsonb) did
not find property 'spring.http.converters.preferred-json-mapper'; NestedCondition on
GsonHttpMessageConvertersConfiguration.JacksonAndJsonbUnavailableCondition.JacksonAvai
lable @ConditionalOnBean (types:
org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
SearchStrategy: all) found bean 'mappingJackson2HttpMessageConverter'; NestedCondition
on
GsonHttpMessageConvertersConfiguration.PreferGsonOrJacksonAndJsonbUnavailableCondition
.GsonPreferred @ConditionalOnProperty (spring.http.converters.preferred-json-
mapper=gson) did not find property 'spring.http.converters.preferred-json-mapper'"
    } ],
    "matched" : [ ]
},
"JsonbHttpMessageConvertersConfiguration" : {
    "notMatched" : [ {
        "condition" : "OnClassCondition",
        "message" : "@ConditionalOnClass did not find required class
'javax.json.bind.Jsonb'"
    } ],
    "matched" : [ ]
}
},
"unconditionalClasses" : [
"org.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguration",
"org.springframework.boot.actuate.autoconfigure.endpoint.EndpointAutoConfiguration" ]
}
}
}

```

### 5.1.1. Response Structure

The response contains details of the application's condition evaluation. The following table describes the structure of the response:



Path	Type	Description
<code>contexts</code>	Object	Application contexts keyed by id.
<code>contexts.*.positiveMatches</code>	Object	Classes and methods with conditions that were matched.
<code>contexts.*.positiveMatches.*.[]condition</code>	String	Name of the condition.
<code>contexts.*.positiveMatches.*.[]message</code>	String	Details of why the condition was matched.
<code>contexts.*.negativeMatches</code>	Object	Classes and methods with conditions that were not matched.
<code>contexts.*.negativeMatches.*.notMatched</code>	Array	Conditions that were matched.
<code>contexts.*.negativeMatches.*.notMatched.[]condition</code>	String	Name of the condition.
<code>contexts.*.negativeMatches.*.notMatched.[]message</code>	String	Details of why the condition was not matched.
<code>contexts.*.negativeMatches.*.matched</code>	Array	Conditions that were matched.
<code>contexts.*.negativeMatches.*.matched.[]condition</code>	String	Name of the condition.
<code>contexts.*.negativeMatches.*.matched.[]message</code>	String	Details of why the condition was matched.
<code>contexts.*.unconditionalClasses</code>	Array	Names of unconditional auto-configuration classes if any.
<code>contexts.*.parentId</code>	String	Id of the parent application context, if any.

# Chapter 6. Configuration Properties (configprops)

The `configprops` endpoint provides information about the application's `@ConfigurationProperties` beans.

## 6.1. Retrieving the `@ConfigurationProperties` Bean

To retrieve the `@ConfigurationProperties` beans, make a `GET` request to `/actuator/configprops`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/configprops' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 1806

{
  "contexts" : {
    "application" : {
      "beans" : {
        "management.endpoints.web.cors-
org.springframework.boot.actuate.autoconfigure.endpoint.web.CorsEndpointProperties" :
{
  "prefix" : "management.endpoints.web.cors",
  "properties" : {
    "allowedHeaders" : [ ],
    "allowedMethods" : [ ],
    "allowedOrigins" : [ ],
    "maxAge" : {
      "units" : [ "SECONDS", "NANOS" ]
    },
    "exposedHeaders" : [ ]
  }
},
    "management.endpoints.web-
org.springframework.boot.actuate.autoconfigure.endpoint.web.WebEndpointProperties" : {
  "prefix" : "management.endpoints.web",
  "properties" : {
    "pathMapping" : { },
    "exposure" : {
      "include" : [ "*" ],
      "exclude" : [ ]
    },
    "basePath" : "/actuator"
```



# Chapter 7. Environment (env)

The `env` endpoint provides information about the application's `Environment`.

## 7.1. Retrieving the Entire Environment

To retrieve the entire environment, make a `GET` request to `/actuator/env`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/env' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 794
```

```
{
  "activeProfiles" : [ ],
  "propertySources" : [ {
    "name" : "systemProperties",
    "properties" : {
      "java.runtime.name" : {
        "value" : "OpenJDK Runtime Environment"
      },
      "java.vm.version" : {
        "value" : "25.192-b12"
      },
      "java.vm.vendor" : {
        "value" : "Oracle Corporation"
      }
    }
  }, {
    "name" : "systemEnvironment",
    "properties" : {
      "JAVA_HOME" : {
        "value" : "/opt/openjdk",
        "origin" : "System Environment Property \"JAVA_HOME\""
      }
    }
  }, {
    "name" : "applicationConfig: [classpath:/application.properties]",
    "properties" : {
      "com.example.cache.max-size" : {
        "value" : "1000",
        "origin" : "class path resource [application.properties]:1:29"
      }
    }
  }
]
}
```

### 7.1.1. Response Structure

The response contains details of the application's **Environment**. The following table describes the structure of the response:

Path	Type	Description
<code>activeProfiles</code>	Array	Names of the active profiles, if any.
<code>propertySources</code>	Array	Property sources in order of precedence.
<code>propertySources.[].name</code>	String	Name of the property source.

Path	Type	Description
<code>propertySources[].properties</code>	Object	Properties in the property source keyed by property name.
<code>propertySources[].properties.*.value</code>	String	Value of the property.
<code>propertySources[].properties.*.origin</code>	String	Origin of the property, if any.

## 7.2. Retrieving a Single Property

To retrieve a single property, make a `GET` request to `/actuator/env/{property.name}`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/env/com.example.cache.max-size' -i -X GET
```

The preceding example retrieves information about the property named `com.example.cache.max-size`. The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 445
Content-Disposition: inline;filename=f.txt

{
  "property" : {
    "source" : "applicationConfig: [classpath:/application.properties]",
    "value" : "1000"
  },
  "activeProfiles" : [ ],
  "propertySources" : [ {
    "name" : "systemProperties"
  }, {
    "name" : "systemEnvironment"
  }, {
    "name" : "applicationConfig: [classpath:/application.properties]",
    "property" : {
      "value" : "1000",
      "origin" : "class path resource [application.properties]:1:29"
    }
  } ]
}
```

### 7.2.1. Response Structure

The response contains details of the requested property. The following table describes the structure of the response:

<b>Path</b>	<b>Type</b>	<b>Description</b>
property	Object	Property from the environment, if found.
property.source	String	Name of the source of the property.
property.value	String	Value of the property.
activeProfiles	Array	Names of the active profiles, if any.
propertySources	Array	Property sources in order of precedence.
propertySources.[].name	String	Name of the property source.
propertySources.[].property	Object	Property in the property source, if any.
propertySources.[].property.value	String	Value of the property.
propertySources.[].property.origin	String	Origin of the property, if any.

# Chapter 8. Flyway (flyway)

The `flyway` endpoint provides information about database migrations performed by Flyway.

## 8.1. Retrieving the Migrations

To retrieve the migrations, make a `GET` request to `/actuator/flyway`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/flyway' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 515

{
  "contexts" : {
    "application" : {
      "flywayBeans" : {
        "flyway" : {
          "migrations" : [ {
            "type" : "SQL",
            "checksum" : -156244537,
            "version" : "1",
            "description" : "init",
            "script" : "V1__init.sql",
            "state" : "SUCCESS",
            "installedBy" : "SA",
            "installedOn" : "2018-11-30T08:30:07.119Z",
            "installedRank" : 1,
            "executionTime" : 1
          } ]
        }
      }
    }
  }
}
```

### 8.1.1. Response Structure

The response contains details of the application's Flyway migrations. The following table describes the structure of the response:



Path	Type	Description
contexts	Object	Application contexts keyed by id
contexts.*.flywayBeans.*.migrations	Array	Migrations performed by the Flyway instance, keyed by Flyway bean name.
contexts.*.flywayBeans.*.migrations[].checksum	Number	Checksum of the migration, if any.
contexts.*.flywayBeans.*.migrations[].description	String	Description of the migration, if any.
contexts.*.flywayBeans.*.migrations[].executionTime	Number	Execution time in milliseconds of an applied migration.
contexts.*.flywayBeans.*.migrations[].installedBy	String	User that installed the applied migration, if any.
contexts.*.flywayBeans.*.migrations[].installedOn	String	Timestamp of when the applied migration was installed, if any.
contexts.*.flywayBeans.*.migrations[].installedRank	Number	Rank of the applied migration, if any. Later migrations have higher ranks.
contexts.*.flywayBeans.*.migrations[].script	String	Name of the script used to execute the migration, if any.
contexts.*.flywayBeans.*.migrations[].state	String	State of the migration. (PENDING, ABOVE_TARGET, BELOW_BASELINE, BASELINE, IGNORED, MISSING_SUCCESS, MISSING_FAILED, SUCCESS, UNDONE, AVAILABLE, FAILED, OUT_OF_ORDER, FUTURE_SUCCESS, FUTURE_FAILED, OUTDATED, SUPERSEDED)
contexts.*.flywayBeans.*.migrations[].type	String	Type of the migration. (SCHEMA, BASELINE, SQL, UNDO_SQL, JDBC, UNDO_JDBC, SPRING_JDBC, UNDO_SPRING_JDBC, CUSTOM, UNDO_CUSTOM)
contexts.*.flywayBeans.*.migrations[].version	String	Version of the database after applying the migration, if any.
contexts.*.parentId	String	Id of the parent application context, if any.

# Chapter 9. Health (health)

The `health` endpoint provides detailed information about the health of the application.

## 9.1. Retrieving the Health of the application

To retrieve the health of the application, make a `GET` request to `/actuator/health`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/health' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 670
```

```
{
  "status" : "UP",
  "details" : {
    "diskSpace" : {
      "status" : "UP",
      "details" : {
        "total" : 136535855104,
        "free" : 97493889024,
        "threshold" : 10485760
      }
    },
    "db" : {
      "status" : "UP",
      "details" : {
        "database" : "HSQL Database Engine",
        "hello" : 1
      }
    },
    "broker" : {
      "status" : "UP",
      "details" : {
        "us1" : {
          "status" : "UP",
          "details" : {
            "version" : "1.0.2"
          }
        },
        "us2" : {
          "status" : "UP",
          "details" : {
            "version" : "1.0.4"
          }
        }
      }
    }
  }
}
```

### 9.1.1. Response Structure

The response contains details of the health of the application. The following table describes the structure of the response:

Path	Type	Description
<code>status</code>	String	Overall status of the application.
<code>details</code>	Object	Details of the health of the application. Presence is controlled by <code>management.endpoint.health.show-details</code> ).
<code>details.*.status</code>	String	Status of a specific part of the application.
<code>details.*.details</code>	Object	Details of the health of a specific part of the application.

## 9.2. Retrieving the Health of a component

To retrieve the health of a particular component of the application, make a `GET` request to `/actuator/health/{component}`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/health/db' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 99

{
  "status" : "UP",
  "details" : {
    "database" : "HSQL Database Engine",
    "hello" : 1
  }
}
```

### 9.2.1. Response Structure

The response contains details of the health of a particular component of the application. The following table describes the structure of the response:

Path	Type	Description
<code>status</code>	String	Status of a specific part of the application
<code>details</code>	Object	Details of the health of a specific part of the application.

## 9.3. Retrieving the Health of a component instance

If a particular component consists of multiple instances (as the `broker` indicator in the example above), the health of a particular instance of that component can be retrieved by issuing a `GET` request to `/actuator/health/{component}/{instance}`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/health/broker/us1' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 66

{
  "status" : "UP",
  "details" : {
    "version" : "1.0.2"
  }
}
```

### 9.3.1. Response Structure

The response contains details of the health of an instance of a particular component of the application. The following table describes the structure of the response:

Path	Type	Description
<code>status</code>	<code>String</code>	Status of a specific part of the application
<code>details</code>	<code>Object</code>	Details of the health of a specific part of the application.

# Chapter 10. Heap Dump (heapdump)

The `heapdump` endpoint provides a heap dump from the application's JVM.

## 10.1. Retrieving the Heap Dump

To retrieve the heap dump, make a `GET` request to `/actuator/heapdump`. The response is binary data in `HPROF` format and can be large. Typically, you should save the response to disk for subsequent analysis. When using `curl`, this can be achieved by using the `-O` option, as shown in the following example:

```
$ curl 'http://localhost:8080/actuator/heapdump' -O
```

The preceding example results in a file named `heapdump` being written to the current working directory.

# Chapter 11. HTTP Trace (`httptrace`)

The `httptrace` endpoint provides information about HTTP request-response exchanges.

## 11.1. Retrieving the Traces

To retrieve the traces, make a `GET` request to `/actuator/httptrace`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/httptrace' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 503

{
  "traces" : [ {
    "timestamp" : "2018-11-30T08:30:09.839Z",
    "principal" : {
      "name" : "alice"
    },
    "session" : {
      "id" : "50abc90f-26e0-48c7-bdef-301d40972ef7"
    },
    "request" : {
      "method" : "GET",
      "uri" : "https://api.example.com",
      "headers" : {
        "Accept" : [ "application/json" ]
      }
    },
    "response" : {
      "status" : 200,
      "headers" : {
        "Content-Type" : [ "application/json" ]
      }
    },
    "timeTaken" : 2
  } ]
}
```

### 11.1.1. Response Structure

The response contains details of the traced HTTP request-response exchanges. The following table describes the structure of the response:

Path	Type	Description
<code>traces</code>	Array	An array of traced HTTP request-response exchanges.
<code>traces[].timestamp</code>	String	Timestamp of when the traced exchange occurred.
<code>traces[].principal</code>	Object	Principal of the exchange, if any.
<code>traces[].principal.name</code>	String	Name of the principal.
<code>traces[].request.method</code>	String	HTTP method of the request.
<code>traces[].request.remoteAddress</code>	String	Remote address from which the request was received, if known.
<code>traces[].request.uri</code>	String	URI of the request.
<code>traces[].request.headers</code>	Object	Headers of the request, keyed by header name.
<code>traces[].request.headers.*.[]</code>	Array	Values of the header
<code>traces[].response.status</code>	Number	Status of the response
<code>traces[].response.headers</code>	Object	Headers of the response, keyed by header name.
<code>traces[].response.headers.*.[]</code>	Array	Values of the header
<code>traces[].session</code>	Object	Session associated with the exchange, if any.
<code>traces[].session.id</code>	String	ID of the session.
<code>traces[].timeTaken</code>	Number	Time, in milliseconds, taken to handle the exchange.



# Chapter 12. Info (*info*)

The *info* endpoint provides general information about the application.

## 12.1. Retrieving the Info

To retrieve the information about the application, make a *GET* request to */actuator/info*, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/info' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 235

{
  "git" : {
    "commit" : {
      "time" : "+50883-09-05T06:49:15Z",
      "id" : "df027cf"
    },
    "branch" : "master"
  },
  "build" : {
    "version" : "1.0.3",
    "artifact" : "application",
    "group" : "com.example"
  }
}
```

### 12.1.1. Response Structure

The response contains general information about the application. Each section of the response is contributed by an *InfoContributor*. Spring Boot provides *build* and *git* contributions.

#### *build* Response Structure

The following table describe the structure of the *build* section of the response:

Path	Type	Description
<i>artifact</i>	<i>String</i>	Artifact ID of the application, if any.
<i>group</i>	<i>String</i>	Group ID of the application, if any.
<i>name</i>	<i>String</i>	Name of the application, if any.

Path	Type	Description
version	String	Version of the application, if any.
time	Varies	Timestamp of when the application was built, if any.

### git Response Structure

The following table describes the structure of the `git` section of the response:

Path	Type	Description
branch	String	Name of the Git branch, if any.
commit	Object	Details of the Git commit, if any.
commit.time	Varies	Timestamp of the commit, if any.
commit.id	String	ID of the commit, if any.

# Chapter 13. Spring Integration graph (`integrationgraph`)

The `integrationgraph` endpoint exposes a graph containing all Spring Integration components.

## 13.1. Retrieving the Spring Integration graph

To retrieve the information about the application, make a `GET` request to `/actuator/integrationgraph`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/integrationgraph' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 662
```

```
{
  "contentDescriptor" : {
    "providerVersion" : "5.1.1.RELEASE",
    "providerFormatVersion" : 1.0,
    "provider" : "spring-integration"
  },
  "nodes" : [ {
    "nodeId" : 1,
    "name" : "nullChannel",
    "componentType" : "channel",
    "properties" : { }
  }, {
    "nodeId" : 2,
    "name" : "errorChannel",
    "componentType" : "publish-subscribe-channel",
    "properties" : { }
  }, {
    "nodeId" : 3,
    "name" : "_org.springframework.integration.errorLogger",
    "componentType" : "logging-channel-adapter",
    "properties" : { },
    "input" : "errorChannel"
  } ],
  "links" : [ {
    "from" : 2,
    "to" : 3,
    "type" : "input"
  } ]
}
```

### 13.1.1. Response Structure

The response contains all Spring Integration components used within the application, as well as the links between them. More information about the structure can be found in the [reference documentation](#).

## 13.2. Rebuilding the Spring Integration graph

To rebuild the exposed graph, make a **POST** request to `/actuator/integrationgraph`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/integrationgraph' -i -X POST
```

This will result in a **204 - No Content** response:

```
HTTP/1.1 204 No Content
```

# Chapter 14. Liquibase (liquibase)

The `liquibase` endpoint provides information about database change sets applied by Liquibase.

## 14.1. Retrieving the Changes

To retrieve the changes, make a `GET` request to `/actuator/liquibase`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/liquibase' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 688

{
  "contexts" : {
    "application" : {
      "liquibaseBeans" : {
        "liquibase" : {
          "changeSets" : [ {
            "author" : "marceloverdijk",
            "changeLog" : "classpath:/db/changelog/db.changelog-master.yaml",
            "comments" : "",
            "contexts" : [ ],
            "dateExecuted" : "2018-11-30T08:29:24.489Z",
            "deploymentId" : "3566564472",
            "description" : "createTable tableName=customer",
            "execType" : "EXECUTED",
            "id" : "1",
            "labels" : [ ],
            "checksum" : "8:46debf252cce6d7b25e28ddeb9fc4bf6",
            "orderExecuted" : 1
          } ]
        }
      }
    }
  }
}
```

### 14.1.1. Response Structure

The response contains details of the application's Liquibase change sets. The following table describes the structure of the response:

Path	Type	Description
contexts	Object	Application contexts keyed by id
contexts.*.liquibaseBeans.*.changeSets	Array	Change sets made by the Liquibase beans, keyed by bean name.
contexts.*.liquibaseBeans.*.changeSets[].author	String	Author of the change set.
contexts.*.liquibaseBeans.*.changeSets[].changeLog	String	Change log that contains the change set.
contexts.*.liquibaseBeans.*.changeSets[].comments	String	Comments on the change set.
contexts.*.liquibaseBeans.*.changeSets[].contexts	Array	Contexts of the change set.
contexts.*.liquibaseBeans.*.changeSets[].dateExecuted	String	Timestamp of when the change set was executed.
contexts.*.liquibaseBeans.*.changeSets[].deploymentId	String	ID of the deployment that ran the change set.
contexts.*.liquibaseBeans.*.changeSets[].description	String	Description of the change set.
contexts.*.liquibaseBeans.*.changeSets[].execType	String	Execution type of the change set ( <b>EXECUTED</b> , <b>FAILED</b> , <b>SKIPPED</b> , <b>RERAN</b> , <b>MARK_RAN</b> ).
contexts.*.liquibaseBeans.*.changeSets[].id	String	ID of the change set.
contexts.*.liquibaseBeans.*.changeSets[].labels	Array	Labels associated with the change set.
contexts.*.liquibaseBeans.*.changeSets[].checksum	String	Checksum of the change set.
contexts.*.liquibaseBeans.*.changeSets[].orderExecuted	Number	Order of the execution of the change set.
contexts.*.liquibaseBeans.*.changeSets[].tag	String	Tag associated with the change set, if any.
contexts.*.parentId	String	Id of the parent application context, if any.

# Chapter 15. Log File (logfile)

The `logfile` endpoint provides access to the contents of the application's log file.

## 15.1. Retrieving the Log File

To retrieve the log file, make a `GET` request to `/actuator/logfile`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/logfile' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 4723
Content-Type: text/plain

.
/\ \ / _ _ _ ' _ _ _ _ ( _ ) _ _ _ _ _ \ \ \ \ \
( ( ) \ _ _ | ' _ | ' _ | | ' _ \ _ ' | \ \ \ \ \
\ \ / _ _ ) | | _ | | | | | | | ( _ | | ) ) ) )
' | _ _ _ | . _ _ | | | _ | | \ _ _ , | / / / /
=====|_|=====|_ _ _ / = / _ / _ / _ /
:: Spring Boot ::

2017-08-08 17:12:30.910 INFO 19866 --- [           main]
s.f.SampleWebFreeMarkerApplication      : Starting SampleWebFreeMarkerApplication on
host.local with PID 19866
2017-08-08 17:12:30.913 INFO 19866 --- [           main]
s.f.SampleWebFreeMarkerApplication      : No active profile set, falling back to
default profiles: default
2017-08-08 17:12:30.952 INFO 19866 --- [           main]
ConfigServletWebServerApplicationContext : Refreshing
org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicati
onContext@76b10754: startup date [Tue Aug 08 17:12:30 BST 2017]; root of context
hierarchy
2017-08-08 17:12:31.878 INFO 19866 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080
(http)
2017-08-08 17:12:31.889 INFO 19866 --- [           main]
o.apache.catalina.core.StandardService  : Starting service [Tomcat]
2017-08-08 17:12:31.890 INFO 19866 --- [           main]
org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache
Tomcat/8.5.16
2017-08-08 17:12:31.978 INFO 19866 --- [ost-startStop-1]
o.a.c.c.C.[Tomcat].[localhost].[/]      : Initializing Spring embedded
WebApplicationContext
```



```

2017-08-08 17:12:31.978 INFO 19866 --- [ost-startStop-1]
o.s.web.context.ContextLoader      : Root WebApplicationContext: initialization
completed in 1028 ms
2017-08-08 17:12:32.080 INFO 19866 --- [ost-startStop-1]
o.s.b.w.servlet.ServletRegistrationBean : Mapping servlet: 'dispatcherServlet' to [/]
2017-08-08 17:12:32.084 INFO 19866 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean  : Mapping filter: 'characterEncodingFilter'
to: [/*]
2017-08-08 17:12:32.084 INFO 19866 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean  : Mapping filter: 'hiddenHttpMethodFilter'
to: [/*]
2017-08-08 17:12:32.084 INFO 19866 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean  : Mapping filter: 'httpPutFormContentFilter'
to: [/*]
2017-08-08 17:12:32.084 INFO 19866 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean  : Mapping filter: 'requestContextFilter' to:
[/*]
2017-08-08 17:12:32.349 INFO 19866 --- [          main]
s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice:
org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicati
onContext@76b10754: startup date [Tue Aug 08 17:12:30 BST 2017]; root of context
hierarchy
2017-08-08 17:12:32.420 INFO 19866 --- [          main]
s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public
org.springframework.http.ResponseEntity<java.util.Map<java.lang.String,
java.lang.Object>>
org.springframework.boot.autoconfigure.web.servlet.error.BasicErrorController.error(ja
vax.servlet.http.HttpServletRequest)
2017-08-08 17:12:32.421 INFO 19866 --- [          main]
s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}"
onto public org.springframework.web.servlet.ModelAndView
org.springframework.boot.autoconfigure.web.servlet.error.BasicErrorController.errorHtm
l(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)
2017-08-08 17:12:32.444 INFO 19866 --- [          main]
o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/webjars/**] onto handler
of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2017-08-08 17:12:32.444 INFO 19866 --- [          main]
o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**] onto handler of type
[class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2017-08-08 17:12:32.471 INFO 19866 --- [          main]
o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**/favicon.ico] onto
handler of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2017-08-08 17:12:32.600 INFO 19866 --- [          main]
o.s.w.s.v.f.FreeMarkerConfigurer      : ClassTemplateLoader for Spring macros added
to FreeMarker configuration
2017-08-08 17:12:32.681 INFO 19866 --- [          main]
o.s.j.e.a.AnnotationMBeanExporter      : Registering beans for JMX exposure on
startup
2017-08-08 17:12:32.744 INFO 19866 --- [          main]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http)

```

```
2017-08-08 17:12:32.750 INFO 19866 --- [          main]
s.f.SampleWebFreeMarkerApplication      : Started SampleWebFreeMarkerApplication in
2.172 seconds (JVM running for 2.479)
```

## 15.2. Retrieving Part of the Log File



Retrieving part of the log file is not supported when using Jersey.

To retrieve part of the log file, make a **GET** request to `/actuator/logfile` by using the **Range** header, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/logfile' -i -X GET \
-H 'Range: bytes=0-1023'
```

The preceding example retrieves the first 1024 bytes of the log file. The resulting response is similar to the following:

```
HTTP/1.1 206 Partial Content
Accept-Ranges: bytes
Content-Range: bytes 0-1023/4723
Content-Length: 1024
Content-Type: text/plain

.
/\ / _ _ _ _ _ ( _ ) _ _ _ _ _ \ \ \ \ \
( ( ) \ _ _ | ' _ | ' _ | ' _ \ _ ' | \ \ \ \ \
\ \ \ \ \ | | ) | | | | | | | ( _ | | ) ) ) )
' | _ _ _ | . _ _ | | _ _ | | \ _ _ , | / / / /
=====|_|=====|___/=/_/_/_/
:: Spring Boot ::

2017-08-08 17:12:30.910 INFO 19866 --- [          main]
s.f.SampleWebFreeMarkerApplication      : Starting SampleWebFreeMarkerApplication on
host.local with PID 19866
2017-08-08 17:12:30.913 INFO 19866 --- [          main]
s.f.SampleWebFreeMarkerApplication      : No active profile set, falling back to
default profiles: default
2017-08-08 17:12:30.952 INFO 19866 --- [          main]
ConfigServletWebServerApplicationContext : Refreshing
org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicati
onContext@76b10754: startup date [Tue Aug 08 17:12:30 BST 2017]; root of context
hierarchy
2017-08-08 17:12:31.878 INFO 19866 --- [          main]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(
```

# Chapter 16. Loggers (loggers)

The `loggers` endpoint provides access to the application's loggers and the configuration of their levels.

## 16.1. Retrieving All Loggers

To retrieve the application's loggers, make a `GET` request to `/actuator/loggers`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/loggers' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Length: 283
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8

{
  "levels" : [ "OFF", "FATAL", "ERROR", "WARN", "INFO", "DEBUG", "TRACE" ],
  "loggers" : {
    "ROOT" : {
      "configuredLevel" : "INFO",
      "effectiveLevel" : "INFO"
    },
    "com.example" : {
      "configuredLevel" : "DEBUG",
      "effectiveLevel" : "DEBUG"
    }
  }
}
```

### 16.1.1. Response Structure

The response contains details of the application's loggers. The following table describes the structure of the response:

Path	Type	Description
<code>levels</code>	Array	Levels support by the logging system.
<code>loggers</code>	Object	Loggers keyed by name.
<code>loggers.*.configuredLevel</code>	String	Configured level of the logger, if any.
<code>loggers.*.effectiveLevel</code>	String	Effective level of the logger.

## 16.2. Retrieving a Single Logger

To retrieve a single logger, make a **GET** request to `/actuator/loggers/{logger.name}`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/loggers/com.example' -i -X GET
```

The preceding example retrieves information about the logger named `com.example`. The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 61
Content-Disposition: inline;filename=f.txt

{
  "configuredLevel" : "INFO",
  "effectiveLevel" : "INFO"
}
```

### 16.2.1. Response Structure

The response contains details of the requested logger. The following table describes the structure of the response:

Path	Type	Description
<code>configuredLevel</code>	<code>String</code>	Configured level of the logger, if any.
<code>effectiveLevel</code>	<code>String</code>	Effective level of the logger.

## 16.3. Setting a Log Level

To set the level of a logger, make a **POST** request to `/actuator/loggers/{logger.name}` with a JSON body that specifies the configured level for the logger, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/loggers/com.example' -i -X POST \
  -H 'Content-Type: application/json' \
  -d '{"configuredLevel":"debug"}'
```

The preceding example sets the `configuredLevel` of the `com.example` logger to `DEBUG`.

### 16.3.1. Request Structure

The request specifies the desired level of the logger. The following table describes the structure of

the request:

Path	Type	Description
<code>configuredLevel</code>	<code>String</code>	Level for the logger. May be omitted to clear the level.

## 16.4. Clearing a Log Level

To clear the level of a logger, make a **POST** request to `/actuator/loggers/{logger.name}` with a JSON body containing an empty object, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/loggers/com.example' -i -X POST \  
  -H 'Content-Type: application/json' \  
  -d '{}'
```

The preceding example clears the configured level of the `com.example` logger.

# Chapter 17. Mappings (mappings)

The `mappings` endpoint provides information about the application's request mappings.

## 17.1. Retrieving the Mappings

To retrieve the mappings, make a `GET` request to `/actuator/mappings`, as shown in the following curl-based example:

```
$ curl 'http://localhost:33632/actuator/mappings' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Date: Fri, 30 Nov 2018 08:30:16 GMT
Content-Length: 5676
Transfer-Encoding: chunked

{
  "contexts" : {
    "application" : {
      "mappings" : {
        "dispatcherServlets" : {
          "dispatcherServlet" : [ {
            "handler" : "ResourceHttpRequestHandler [class path resource [META-INF/resources/], class path resource [resources/], class path resource [static/], class path resource [public/], ServletContext resource [/], class path resource []]",
            "predicate" : "/*/favicon.ico"
          }, {
            "handler" : "Actuator web endpoint 'mappings'",
            "predicate" : "{GET /actuator/mappings, produces [application/vnd.spring-boot.actuator.v2+json || application/json]}",
            "details" : {
              "handlerMethod" : {
                "className" :
"org.springframework.boot.actuate.endpoint.web.servlet.AbstractWebMvcEndpointHandlerMapping.OperationHandler",
                "name" : "handle",
                "descriptor" :
"(Ljavax/servlet/http/HttpServletRequest;Ljava/util/Map;)Ljava/lang/Object;"
              },
              "requestMappingConditions" : {
                "consumes" : [ ],
                "headers" : [ ],
                "methods" : [ "GET" ],
                "params" : [ ],
                "patterns" : [ "/actuator/mappings" ],
```



```

    },
    "requestMappingConditions" : {
        "consumes" : [ {
            "mediaType" : "application/json",
            "negated" : false
        }, {
            "mediaType" : "application/xml",
            "negated" : true
        } ],
        "headers" : [ {
            "name" : "X-Custom",
            "value" : "Foo",
            "negated" : false
        } ],
        "methods" : [ "POST" ],
        "params" : [ {
            "name" : "a",
            "value" : "alpha",
            "negated" : true
        } ],
        "patterns" : [ "/" ],
        "produces" : [ {
            "mediaType" : "text/plain",
            "negated" : false
        } ]
    }
}
}, {
    "handler" : "ResourceHttpRequestHandler [\"classpath:/META-INF/resources/webjars/\"]",
    "predicate" : "/webjars/**"
}, {
    "handler" : "ResourceHttpRequestHandler [\"classpath:/META-INF/resources/\", \"classpath:/resources/\", \"classpath:/static/\", \"classpath:/public/\", \"/\"]",
    "predicate" : "/**"
} ]
},
"servletFilters" : [ {
    "servletNameMappings" : [ ],
    "urlPatternMappings" : [ "/"* ],
    "name" : "requestContextFilter",
    "className" :
"org.springframework.boot.web.servlet.filter.OrderedRequestContextFilter"
}, {
    "servletNameMappings" : [ ],
    "urlPatternMappings" : [ "/"* ],
    "name" : "hiddenHttpMethodFilter",
    "className" :
"org.springframework.boot.web.servlet.filter.OrderedHiddenHttpMethodFilter"
}, {

```



```

        "servletNameMappings" : [ ],
        "urlPatternMappings" : [ "/"* ],
        "name" : "formContentFilter",
        "className" :
"org.springframework.boot.web.servlet.filter.OrderedFormContentFilter"
    } ],
    "servlets" : [ {
        "mappings" : [ ],
        "name" : "default",
        "className" : "org.apache.catalina.servlets.DefaultServlet"
    }, {
        "mappings" : [ "/" ],
        "name" : "dispatcherServlet",
        "className" : "org.springframework.web.servlet.DispatcherServlet"
    } ]
    }
}
}
}
}

```

### 17.1.1. Response Structure

The response contains details of the application's mappings. The items found in the response depend on the type of web application (reactive or Servlet-based). The following table describes the structure of the common elements of the response:

Path	Type	Description
<code>contexts</code>	Object	Application contexts keyed by id.
<code>contexts.*.mappings</code>	Object	Mappings in the context, keyed by mapping type.
<code>contexts.*.mappings.dispatcherServlets</code>	Object	Dispatcher servlet mappings, if any.
<code>contexts.*.mappings.servletFilters</code>	Array	Servlet filter mappings, if any.
<code>contexts.*.mappings.servlets</code>	Array	Servlet mappings, if any.
<code>contexts.*.mappings.dispatcherHandlers</code>	Object	Dispatcher handler mappings, if any.
<code>contexts.*.parentId</code>	String	Id of the parent application context, if any.

The entries that may be found in `contexts.*.mappings` are described in the following sections.

### 17.1.2. Dispatcher Servlets Response Structure

When using Spring MVC, the response contains details of any `DispatcherServlet` request mappings beneath `contexts.*.mappings.dispatcherServlets`. The following table describes the structure of this section of the response:

Path	Type	Description
*	Array	Dispatcher servlet mappings, if any, keyed by dispatcher servlet bean name.
*.[] <b>.details</b>	Object	Additional implementation-specific details about the mapping. Optional.
*.[] <b>.handler</b>	String	Handler for the mapping.
*.[] <b>.predicate</b>	String	Predicate for the mapping.
*.[] <b>.details.handlerMethod</b>	Object	Details of the method, if any, that will handle requests to this mapping.
*.[] <b>.details.handlerMethod.className</b>	String	Fully qualified name of the class of the method.
*.[] <b>.details.handlerMethod.name</b>	String	Name of the method.
*.[] <b>.details.handlerMethod.descriptor</b>	String	Descriptor of the method as specified in the Java Language Specification.
*.[] <b>.details.requestMappingConditions</b>	Object	Details of the request mapping conditions.
*.[] <b>.details.requestMappingConditions.consumes</b>	Array	Details of the consumes condition
*.[] <b>.details.requestMappingConditions.consumes.[]<b>.mediaType</b></b>	String	Consumed media type.
*.[] <b>.details.requestMappingConditions.consumes.[]<b>.negated</b></b>	Boolean	Whether the media type is negated.
*.[] <b>.details.requestMappingConditions.headers</b>	Array	Details of the headers condition.
*.[] <b>.details.requestMappingConditions.headers.[]<b>.name</b></b>	String	Name of the header.
*.[] <b>.details.requestMappingConditions.headers.[]<b>.value</b></b>	String	Required value of the header, if any.
*.[] <b>.details.requestMappingConditions.headers.[]<b>.negated</b></b>	Boolean	Whether the value is negated.
*.[] <b>.details.requestMappingConditions.methods</b>	Array	HTTP methods that are handled.
*.[] <b>.details.requestMappingConditions.params</b>	Array	Details of the params condition.
*.[] <b>.details.requestMappingConditions.params.[]<b>.name</b></b>	String	Name of the parameter.

Path	Type	Description
*.[] details.requestMappingConditions.params.[] value	String	Required value of the parameter, if any.
*.[] details.requestMappingConditions.params.[] negated	Boolean	Whether the value is negated.
*.[] details.requestMappingConditions.patterns	Array	Patterns identifying the paths handled by the mapping.
*.[] details.requestMappingConditions.produces	Array	Details of the produces condition.
*.[] details.requestMappingConditions.produces.[] mediaType	String	Produced media type.
*.[] details.requestMappingConditions.produces.[] negated	Boolean	Whether the media type is negated.

### 17.1.3. Servlets Response Structure

When using the Servlet stack, the response contains details of any `Servlet` mappings beneath `contexts.*.mappings.servlets`. The following table describes the structure of this section of the response:

Path	Type	Description
[].mappings	Array	Mappings of the servlet.
[].name	String	Name of the servlet.
[].className	String	Class name of the servlet

### 17.1.4. Servlet Filters Response Structure

When using the Servlet stack, the response contains details of any `Filter` mappings beneath `contexts.*.mappings.servletFilters`. The following table describes the structure of this section of the response:

Path	Type	Description
[].servletNameMappings	Array	Names of the servlets to which the filter is mapped.
[].urlPatternMappings	Array	URL pattern to which the filter is mapped.
[].name	String	Name of the filter.
[].className	String	Class name of the filter

### 17.1.5. Dispatcher Handlers Response Structure

When using Spring WebFlux, the response contains details of any `DispatcherHandler` request

mappings beneath `contexts.*.mappings.dispatcherHandlers`. The following table describes the structure of this section of the response:

Path	Type	Description
<code>*</code>	Array	Dispatcher handler mappings, if any, keyed by dispatcher handler bean name.
<code>*.[].details</code>	Object	Additional implementation-specific details about the mapping. Optional.
<code>*.[].handler</code>	String	Handler for the mapping.
<code>*.[].predicate</code>	String	Predicate for the mapping.
<code>*.[].details.requestMappingConditions</code>	Object	Details of the request mapping conditions.
<code>*.[].details.requestMappingConditions.consumes</code>	Array	Details of the consumes condition
<code>*.[].details.requestMappingConditions.consumes.[].mediaType</code>	String	Consumed media type.
<code>*.[].details.requestMappingConditions.consumes.[].negated</code>	Boolean	Whether the media type is negated.
<code>*.[].details.requestMappingConditions.headers</code>	Array	Details of the headers condition.
<code>*.[].details.requestMappingConditions.headers.[].name</code>	String	Name of the header.
<code>*.[].details.requestMappingConditions.headers.[].value</code>	String	Required value of the header, if any.
<code>*.[].details.requestMappingConditions.headers.[].negated</code>	Boolean	Whether the value is negated.
<code>*.[].details.requestMappingConditions.methods</code>	Array	HTTP methods that are handled.
<code>*.[].details.requestMappingConditions.params</code>	Array	Details of the params condition.
<code>*.[].details.requestMappingConditions.params.[].name</code>	String	Name of the parameter.
<code>*.[].details.requestMappingConditions.params.[].value</code>	String	Required value of the parameter, if any.
<code>*.[].details.requestMappingConditions.params.[].negated</code>	Boolean	Whether the value is negated.
<code>*.[].details.requestMappingConditions.patterns</code>	Array	Patterns identifying the paths handled by the mapping.

<b>Path</b>	<b>Type</b>	<b>Description</b>
*. [].details.requestMappingConditions.produces	Array	Details of the produces condition.
*. [].details.requestMappingConditions.produces [].mediaType	String	Produced media type.
*. [].details.requestMappingConditions.produces [].negated	Boolean	Whether the media type is negated.
*. [].details.handlerMethod	Object	Details of the method, if any, that will handle requests to this mapping.
*. [].details.handlerMethod.className	String	Fully qualified name of the class of the method.
*. [].details.handlerMethod.name	String	Name of the method.
*. [].details.handlerMethod.descriptor	String	Descriptor of the method as specified in the Java Language Specification.
*. [].details.handlerFunction	Object	Details of the function, if any, that will handle requests to this mapping.
*. [].details.handlerFunction.className	String	Fully qualified name of the class of the function.

# Chapter 18. Metrics (**metrics**)

The **metrics** endpoint provides access to application metrics.

## 18.1. Retrieving Metric Names

To retrieve the names of the available metrics, make a **GET** request to **/actuator/metrics**, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/metrics' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 154

{
  "names" : [ "jvm.memory.max", "jvm.memory.used", "jvm.memory.committed",
    "jvm.buffer.memory.used", "jvm.buffer.count", "jvm.buffer.total.capacity" ]
}
```

### 18.1.1. Response Structure

The response contains details of the metric names. The following table describes the structure of the response:

Path	Type	Description
<b>names</b>	<b>Array</b>	Names of the known metrics.

## 18.2. Retrieving a Metric

To retrieve a metric, make a **GET** request to **/actuator/metrics/{metric.name}**, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/metrics/jvm.memory.max' -i -X GET
```

The preceding example retrieves information about the metric named **jvm.memory.max**. The resulting response is similar to the following:

```

HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Disposition: inline;filename=f.txt
Content-Length: 474

{
  "name" : "jvm.memory.max",
  "description" : "The maximum amount of memory in bytes that can be used for memory
management",
  "baseUnit" : "bytes",
  "measurements" : [ {
    "statistic" : "VALUE",
    "value" : 2.384986111E9
  } ],
  "availableTags" : [ {
    "tag" : "area",
    "values" : [ "heap", "nonheap" ]
  }, {
    "tag" : "id",
    "values" : [ "Compressed Class Space", "PS Survivor Space", "PS Old Gen",
"Metaspace", "PS Eden Space", "Code Cache" ]
  } ]
}

```

### 18.2.1. Query Parameters

The endpoint uses query parameters to [drill down](#) into a metric by using its tags. The following table shows the single supported query parameter:

Parameter	Description
<code>tag</code>	A tag to use for drill-down in the form <code>name:value</code> .

### 18.2.2. Response structure

The response contains details of the metric. The following table describes the structure of the response:

Path	Type	Description
<code>name</code>	String	Name of the metric
<code>description</code>	String	Description of the metric
<code>baseUnit</code>	String	Base unit of the metric
<code>measurements</code>	Array	Measurements of the metric

Path	Type	Description
<code>measurements[].statistic</code>	String	Statistic of the measurement. (TOTAL, TOTAL_TIME, COUNT, MAX, VALUE, UNKNOWN, ACTIVE_TASKS, DURATION).
<code>measurements[].value</code>	Number	Value of the measurement.
<code>availableTags</code>	Array	Tags that are available for drill-down.
<code>availableTags[].tag</code>	String	Name of the tag.
<code>availableTags[].values</code>	Array	Possible values of the tag.

## 18.3. Drilling Down

To drill down into a metric, make a `GET` request to `/actuator/metrics/{metric.name}` using the `tag` query parameter, as shown in the following curl-based example:

```
$ curl
'http://localhost:8080/actuator/metrics/jvm.memory.max?tag=area%3Anonheap&tag=id%3ACom
pressed+Class+Space' -i -X GET
```

The preceding example retrieves the `jvm.memory.max` metric, where the `area` tag has a value of `nonheap` and the `id` attribute has a value of `Compressed Class Space`. The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 263
Content-Disposition: inline;filename=f.txt

{
  "name" : "jvm.memory.max",
  "description" : "The maximum amount of memory in bytes that can be used for memory
management",
  "baseUnit" : "bytes",
  "measurements" : [ {
    "statistic" : "VALUE",
    "value" : 1.073741824E9
  } ],
  "availableTags" : [ ]
}
```



# Chapter 19. Prometheus (prometheus)

The `prometheus` endpoint provides Spring Boot application's metrics in the format required for scraping by a Prometheus server.

## 19.1. Retrieving the Metrics

To retrieve the metrics, make a `GET` request to `/actuator/prometheus`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/prometheus' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Length: 2373
Content-Type: text/plain;version=0.0.4;charset=utf-8
```

```
# HELP jvm_memory_committed_bytes The amount of memory in bytes that is committed for
the Java virtual machine to use
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes{area="heap",id="PS Survivor Space",} 1.2058624E7
jvm_memory_committed_bytes{area="heap",id="PS Old Gen",} 3.26631424E8
jvm_memory_committed_bytes{area="heap",id="PS Eden Space",} 3.29777152E8
jvm_memory_committed_bytes{area="nonheap",id="Metaspace",} 1.60079872E8
jvm_memory_committed_bytes{area="nonheap",id="Code Cache",} 5.3542912E7
jvm_memory_committed_bytes{area="nonheap",id="Compressed Class Space",} 2.2462464E7
# HELP jvm_memory_max_bytes The maximum amount of memory in bytes that can be used for
memory management
# TYPE jvm_memory_max_bytes gauge
jvm_memory_max_bytes{area="heap",id="PS Survivor Space",} 1.2058624E7
jvm_memory_max_bytes{area="heap",id="PS Old Gen",} 7.16177408E8
jvm_memory_max_bytes{area="heap",id="PS Eden Space",} 3.32398592E8
jvm_memory_max_bytes{area="nonheap",id="Metaspace",} -1.0
jvm_memory_max_bytes{area="nonheap",id="Code Cache",} 2.5165824E8
jvm_memory_max_bytes{area="nonheap",id="Compressed Class Space",} 1.073741824E9
# HELP jvm_memory_used_bytes The amount of used memory
# TYPE jvm_memory_used_bytes gauge
jvm_memory_used_bytes{area="heap",id="PS Survivor Space",} 3403120.0
jvm_memory_used_bytes{area="heap",id="PS Old Gen",} 1.82522408E8
jvm_memory_used_bytes{area="heap",id="PS Eden Space",} 9.8273272E7
jvm_memory_used_bytes{area="nonheap",id="Metaspace",} 1.50800488E8
jvm_memory_used_bytes{area="nonheap",id="Code Cache",} 5.3160704E7
jvm_memory_used_bytes{area="nonheap",id="Compressed Class Space",} 2.0613816E7
# HELP jvm_buffer_count_buffers An estimate of the number of buffers in the pool
# TYPE jvm_buffer_count_buffers gauge
jvm_buffer_count_buffers{id="direct",} 20.0
jvm_buffer_count_buffers{id="mapped",} 0.0
# HELP jvm_buffer_total_capacity_bytes An estimate of the total capacity of the
buffers in this pool
# TYPE jvm_buffer_total_capacity_bytes gauge
jvm_buffer_total_capacity_bytes{id="direct",} 802948.0
jvm_buffer_total_capacity_bytes{id="mapped",} 0.0
# HELP jvm_buffer_memory_used_bytes An estimate of the memory that the Java virtual
machine is using for this buffer pool
# TYPE jvm_buffer_memory_used_bytes gauge
jvm_buffer_memory_used_bytes{id="direct",} 802949.0
jvm_buffer_memory_used_bytes{id="mapped",} 0.0
```

# Chapter 20. Scheduled Tasks (scheduledtasks)

The `scheduledtasks` endpoint provides information about the application's scheduled tasks.

## 20.1. Retrieving the Scheduled Tasks

To retrieve the scheduled tasks, make a `GET` request to `/actuator/scheduledtasks`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/scheduledtasks' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 451

{
  "cron" : [ {
    "runnable" : {
      "target" : "com.example.Processor.processOrders"
    },
    "expression" : "0 0 0/3 1/1 * ?"
  } ],
  "fixedDelay" : [ {
    "runnable" : {
      "target" : "com.example.Processor.purge"
    },
    "initialDelay" : 5000,
    "interval" : 5000
  } ],
  "fixedRate" : [ {
    "runnable" : {
      "target" : "com.example.Processor.retrieveIssues"
    },
    "initialDelay" : 10000,
    "interval" : 3000
  } ]
}
```

### 20.1.1. Response Structure

The response contains details of the application's scheduled tasks. The following table describes the structure of the response:

Path	Type	Description
<code>cron</code>	Array	Cron tasks, if any.
<code>cron.[].runnable.target</code>	String	Target that will be executed.
<code>cron.[].expression</code>	String	Cron expression.
<code>fixedDelay</code>	Array	Fixed delay tasks, if any.
<code>fixedDelay.[].runnable.target</code>	String	Target that will be executed.
<code>fixedDelay.[].initialDelay</code>	Number	Delay, in milliseconds, before first execution.
<code>fixedDelay.[].interval</code>	Number	Interval, in milliseconds, between the end of the last execution and the start of the next.
<code>fixedRate</code>	Array	Fixed rate tasks, if any.
<code>fixedRate.[].runnable.target</code>	String	Target that will be executed.
<code>fixedRate.[].interval</code>	Number	Interval, in milliseconds, between the start of each execution.
<code>fixedRate.[].initialDelay</code>	Number	Delay, in milliseconds, before first execution.

# Chapter 21. Sessions (sessions)

The `sessions` endpoint provides information about the application's HTTP sessions that are managed by Spring Session.

## 21.1. Retrieving Sessions

To retrieve the sessions, make a `GET` request to `/actuator/sessions`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/sessions?username=alice' -i -X GET
```

The preceding examples retrieves all of the sessions for the user whose username is `alice`.

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 753

{
  "sessions" : [ {
    "id" : "2c42105e-82da-4274-990d-228420c30eed",
    "attributeNames" : [ ],
    "creationTime" : "2018-11-30T06:30:18.989Z",
    "lastAccessedTime" : "2018-11-30T08:30:06.989Z",
    "maxInactiveInterval" : 1800,
    "expired" : false
  }, {
    "id" : "c01bcc7b-d78e-4bd7-99fc-73c128d42025",
    "attributeNames" : [ ],
    "creationTime" : "2018-11-29T20:30:18.988Z",
    "lastAccessedTime" : "2018-11-30T08:29:33.988Z",
    "maxInactiveInterval" : 1800,
    "expired" : false
  }, {
    "id" : "4db5efcc-99cb-4d05-a52c-b49acfbb7ea9",
    "attributeNames" : [ ],
    "creationTime" : "2018-11-30T03:30:18.989Z",
    "lastAccessedTime" : "2018-11-30T08:29:41.989Z",
    "maxInactiveInterval" : 1800,
    "expired" : false
  } ]
}
```

### 21.1.1. Query Parameters

The endpoint uses query parameters to limit the sessions that it returns. The following table shows the single required query parameter:

Parameter	Description
<code>username</code>	Name of the user.

### 21.1.2. Response Structure

The response contains details of the matching sessions. The following table describes the structure of the response:

Path	Type	Description
<code>sessions</code>	Array	Sessions for the given username.
<code>sessions.[].id</code>	String	ID of the session.
<code>sessions.[].attributeNames</code>	Array	Names of the attributes stored in the session.
<code>sessions.[].creationTime</code>	String	Timestamp of when the session was created.
<code>sessions.[].lastAccessedTime</code>	String	Timestamp of when the session was last accessed.
<code>sessions.[].maxInactiveInterval</code>	Number	Maximum permitted period of inactivity, in seconds, before the session will expire.
<code>sessions.[].expired</code>	Boolean	Whether the session has expired.

## 21.2. Retrieving a Single Session

To retrieve a single session, make a `GET` request to `/actuator/sessions/{id}`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/sessions/4db5efcc-99cb-4d05-a52c-b49acfb7ea9'
-i -X GET
```

The preceding example retrieves the session with the `id` of `4db5efcc-99cb-4d05-a52c-b49acfb7ea9`. The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 228
```

```
{
  "id" : "4db5efcc-99cb-4d05-a52c-b49acfb7ea9",
  "attributeNames" : [ ],
  "creationTime" : "2018-11-30T03:30:18.989Z",
  "lastAccessedTime" : "2018-11-30T08:29:41.989Z",
  "maxInactiveInterval" : 1800,
  "expired" : false
}
```

### 21.2.1. Response Structure

The response contains details of the requested session. The following table describes the structure of the response:

Path	Type	Description
<code>id</code>	String	ID of the session.
<code>attributeNames</code>	Array	Names of the attributes stored in the session.
<code>creationTime</code>	String	Timestamp of when the session was created.
<code>lastAccessedTime</code>	String	Timestamp of when the session was last accessed.
<code>maxInactiveInterval</code>	Number	Maximum permitted period of inactivity, in seconds, before the session will expire.
<code>expired</code>	Boolean	Whether the session has expired.

## 21.3. Deleting a Session

To delete a session, make a **DELETE** request to `/actuator/sessions/{id}`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/sessions/4db5efcc-99cb-4d05-a52c-b49acfb7ea9'
-i -X DELETE
```

The preceding example deletes the session with the `id` of `4db5efcc-99cb-4d05-a52c-b49acfb7ea9`.

# Chapter 22. Shutdown (shutdown)

The `shutdown` endpoint is used to shut down the application.

## 22.1. Shutting Down the Application

To shut down the application, make a `POST` request to `/actuator/shutdown`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/shutdown' -i -X POST
```

A response similar to the following is produced:

```
HTTP/1.1 200 OK
Content-Length: 41
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8

{
  "message" : "Shutting down, bye..."
}
```

### 22.1.1. Response Structure

The response contains details of the result of the shutdown request. The following table describes the structure of the response:

Path	Type	Description
<code>message</code>	<code>String</code>	Message describing the result of the request.



# Chapter 23. Thread Dump (threaddump)

The `threaddump` endpoint provides a thread dump from the application's JVM.

## 23.1. Retrieving the Thread Dump

To retrieve the thread dump, make a `GET` request to `/actuator/threaddump`, as shown in the following curl-based example:

```
$ curl 'http://localhost:8080/actuator/threaddump' -i -X GET
```

The resulting response is similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.spring-boot.actuator.v2+json;charset=UTF-8
Content-Length: 4520

{
  "threads" : [ {
    "threadName" : "Thread-134",
    "threadId" : 613,
    "blockedTime" : -1,
    "blockedCount" : 0,
    "waitedTime" : -1,
    "waitedCount" : 1,
    "lockOwnerId" : -1,
    "inNative" : false,
    "suspended" : false,
    "threadState" : "TIMED_WAITING",
    "stackTrace" : [ {
      "methodName" : "sleep",
      "fileName" : "Thread.java",
      "lineNumber" : -2,
      "className" : "java.lang.Thread",
      "nativeMethod" : true
    }, {
      "methodName" : "performShutdown",
      "fileName" : "ShutdownEndpoint.java",
      "lineNumber" : 67,
      "className" : "org.springframework.boot.actuate.context.ShutdownEndpoint",
      "nativeMethod" : false
    }, {
      "methodName" : "run",
      "lineNumber" : -1,
      "className" :
"org.springframework.boot.actuate.context.ShutdownEndpoint$$Lambda$1425/2079068994",
      "nativeMethod" : false
    }
  ]
}
```

```

    "methodName" : "run",
    "fileName" : "Thread.java",
    "lineNumber" : 748,
    "className" : "java.lang.Thread",
    "nativeMethod" : false
  } ],
  "lockedMonitors" : [ ],
  "lockedSynchronizers" : [ ]
}, {
  "threadName" : "pool-10-thread-1",
  "threadId" : 605,
  "blockedTime" : -1,
  "blockedCount" : 0,
  "waitedTime" : -1,
  "waitedCount" : 3,
  "lockName" :
"java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@1e1cffa",
  "lockOwnerId" : -1,
  "inNative" : false,
  "suspended" : false,
  "threadState" : "TIMED_WAITING",
  "stackTrace" : [ {
    "methodName" : "park",
    "fileName" : "Unsafe.java",
    "lineNumber" : -2,
    "className" : "sun.misc.Unsafe",
    "nativeMethod" : true
  }, {
    "methodName" : "parkNanos",
    "fileName" : "LockSupport.java",
    "lineNumber" : 215,
    "className" : "java.util.concurrent.locks.LockSupport",
    "nativeMethod" : false
  }, {
    "methodName" : "awaitNanos",
    "fileName" : "AbstractQueuedSynchronizer.java",
    "lineNumber" : 2078,
    "className" :
"java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject",
    "nativeMethod" : false
  }, {
    "methodName" : "take",
    "fileName" : "ScheduledThreadPoolExecutor.java",
    "lineNumber" : 1093,
    "className" :
"java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue",
    "nativeMethod" : false
  }, {
    "methodName" : "take",
    "fileName" : "ScheduledThreadPoolExecutor.java",
    "lineNumber" : 809,

```

```

    "className" :
"java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue",
    "nativeMethod" : false
  }, {
    "methodName" : "getTask",
    "fileName" : "ThreadPoolExecutor.java",
    "lineNumber" : 1074,
    "className" : "java.util.concurrent.ThreadPoolExecutor",
    "nativeMethod" : false
  }, {
    "methodName" : "runWorker",
    "fileName" : "ThreadPoolExecutor.java",
    "lineNumber" : 1134,
    "className" : "java.util.concurrent.ThreadPoolExecutor",
    "nativeMethod" : false
  }, {
    "methodName" : "run",
    "fileName" : "ThreadPoolExecutor.java",
    "lineNumber" : 624,
    "className" : "java.util.concurrent.ThreadPoolExecutor$Worker",
    "nativeMethod" : false
  }, {
    "methodName" : "run",
    "fileName" : "Thread.java",
    "lineNumber" : 748,
    "className" : "java.lang.Thread",
    "nativeMethod" : false
  } ],
  "lockedMonitors" : [ ],
  "lockedSynchronizers" : [ ],
  "lockInfo" : {
    "className" :
"java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject",
    "identityHashCode" : 31576058
  }
}, {
  "threadName" : "http-nio-auto-15-33632-AsyncTimeout",
  "threadId" : 598,
  "blockedTime" : -1,
  "blockedCount" : 0,
  "waitedTime" : -1,
  "waitedCount" : 6,
  "lockOwnerId" : -1,
  "inNative" : false,
  "suspended" : false,
  "threadState" : "TIMED_WAITING",
  "stackTrace" : [ {
    "methodName" : "sleep",
    "fileName" : "Thread.java",
    "lineNumber" : -2,
    "className" : "java.lang.Thread",

```

```

    "nativeMethod" : true
  }, {
    "methodName" : "run",
    "fileName" : "AbstractProtocol.java",
    "lineNumber" : 1134,
    "className" : "org.apache.coyote.AbstractProtocol$AsyncTimeout",
    "nativeMethod" : false
  }, {
    "methodName" : "run",
    "fileName" : "Thread.java",
    "lineNumber" : 748,
    "className" : "java.lang.Thread",
    "nativeMethod" : false
  } ],
  "lockedMonitors" : [ ],
  "lockedSynchronizers" : [ ]
} ]
}

```

### 23.1.1. Response Structure

The response contains details of the JVM's threads. The following table describes the structure of the response:

Path	Type	Description
<code>threads</code>	Array	JVM's threads.
<code>threads[].blockedCount</code>	Number	Total number of times that the thread has been blocked.
<code>threads[].blockedTime</code>	Number	Time in milliseconds that the thread has spent blocked. -1 if thread contention monitoring is disabled.
<code>threads[].daemon</code>	Boolean	Whether the thread is a daemon thread. Only available on Java 9 or later.
<code>threads[].inNative</code>	Boolean	Whether the thread is executing native code.
<code>threads[].lockName</code>	String	Description of the object on which the thread is blocked, if any.
<code>threads[].lockInfo</code>	Object	Object for which the thread is blocked waiting.
<code>threads[].lockInfo.className</code>	String	Fully qualified class name of the lock object.

Path	Type	Description
<code>threads[].lockInfo.identityHashCode</code>	Number	Identity hash code of the lock object.
<code>threads[].lockedMonitors</code>	Array	Monitors locked by this thread, if any
<code>threads[].lockedMonitors[].className</code>	String	Class name of the lock object.
<code>threads[].lockedMonitors[].identityHashCode</code>	Number	Identity hash code of the lock object.
<code>threads[].lockedMonitors[].lockedStackDepth</code>	Number	Stack depth where the monitor was locked.
<code>threads[].lockedMonitors[].lockedStackFrame</code>	Object	Stack frame that locked the monitor.
<code>threads[].lockedSynchronizers</code>	Array	Synchronizers locked by this thread.
<code>threads[].lockedSynchronizers[].className</code>	String	Class name of the locked synchronizer.
<code>threads[].lockedSynchronizers[].identifyHashCode</code>	Number	Identity hash code of the locked synchronizer.
<code>threads[].lockOwnerId</code>	Number	ID of the thread that owns the object on which the thread is blocked. <code>-1</code> if the thread is not blocked.
<code>threads[].lockOwnerName</code>	String	Name of the thread that owns the object on which the thread is blocked, if any.
<code>threads[].priority</code>	Number	Priority of the thread. Only available on Java 9 or later.
<code>threads[].stackTrace</code>	Array	Stack trace of the thread.
<code>threads[].stackTrace[].classLoaderName</code>	String	Name of the class loader of the class that contains the execution point identified by this entry, if any. Only available on Java 9 or later.
<code>threads[].stackTrace[].className</code>	String	Name of the class that contains the execution point identified by this entry.
<code>threads[].stackTrace[].fileName</code>	String	Name of the source file that contains the execution point identified by this entry, if any.

Path	Type	Description
<code>threads[].stackTrace[].lineNumber</code>	Number	Line number of the execution point identified by this entry. Negative if unknown.
<code>threads[].stackTrace[].methodName</code>	String	Name of the method.
<code>threads[].stackTrace[].moduleName</code>	String	Name of the module that contains the execution point identified by this entry, if any. Only available on Java 9 or later.
<code>threads[].stackTrace[].moduleVersion</code>	String	Version of the module that contains the execution point identified by this entry, if any. Only available on Java 9 or later.
<code>threads[].stackTrace[].nativeMethod</code>	Boolean	Whether the execution point is a native method.
<code>threads[].suspended</code>	Boolean	Whether the thread is suspended.
<code>threads[].threadId</code>	Number	ID of the thread.
<code>threads[].threadName</code>	String	Name of the thread.
<code>threads[].threadState</code>	String	State of the thread ( <b>NEW</b> , <b>RUNNABLE</b> , <b>BLOCKED</b> , <b>WAITING</b> , <b>TIMED_WAITING</b> , <b>TERMINATED</b> ).
<code>threads[].waitedCount</code>	Number	Total number of times that the thread has waited for notification.
<code>threads[].waitedTime</code>	Number	Time in milliseconds that the thread has spent waiting. -1 if thread contention monitoring is disabled