Spring Boot Admin – Admin UI for administration of spring boot applications

javacodegeeks.com/2017/06/spring-boot-admin-admin-ui-administration-spring-boot-applications.html

As part of micro services development many of us are using Spring Boot along with Spring Cloud features. In micro services world we will have many Spring Boot applications which will be running on same/different hosts. If we add Spring Actuator to the Spring Boot applications, we will get a lot of out of the box end points to monitor and interact with Spring Boot applications. The list is given below.

ID	Description	Sensitive Default
`actuator`	Provides a hypermedia-based "discovery page" for the other endpoints. Requires Spring HATEOAS to be on the classpath.	true
`auditevents`	Exposes audit events information for the current application.	true
`autoconfig`	Displays an auto-configuration report showing all auto-configuration candidates and the reason why they 'were' or 'were not' applied.	true
`beans`	Displays a complete list of all the Spring beans in your application.	true
`configprops`	Displays a collated list of all `@ConfigurationProperties`.	true
`dump`	Performs a thread dump.	true
`env`	Exposes properties from Spring's `ConfigurableEnvironment`.	true
`flyway`	Shows any Flyway database migrations that have been applied.	true
`health`	Shows application health information (when the application is secure, a simple 'status' when accessed over an unauthenticated connection or full message details when authenticated).	false
`info`	Displays arbitrary application info.	false
`loggers`	Shows and modifies the configuration of loggers in the application.	true
`liquibase`	Shows any Liquibase database migrations that have been applied.	true
`metrics`	Shows 'metrics' information for the current application.	true
`mappings`	Displays a collated list of all `@RequestMapping` paths.	true
`shutdown`	Allows the application to be gracefully shutdown (not enabled by default).	true
`trace`	Displays trace information (by default the last 100 HTTP requests).	true

The above end points provides a lot of insights about Spring Boot application. But If you have many applications running then monitoring each application by hitting the end points and inspecting the JSON response is tedious process. To avoid this hassle Code Centric team came up with Spring Boot Admin module which will provide us Admin UI Dash board to administer Spring Boot applications. This module crunches the data from Actuator end points and provides insights about all the registered applications in single dash-board. Now we will demonstrate the Spring Boot Admin features in the following sections.

As a first step, create a Spring Boot application which we will make as Spring Boot Admin server module by adding the below maven dependencies.

```
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-server</artifactId>
```

```
    <version>1.5.1</version>
</dependency>
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-server-ui</artifactId>
    <version>1.5.1</version>
</dependency>
```

Add Spring Boot Admin Server configuration via adding `@EnableAdminServer` to your configuration.

```
package org.samrttechie;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import de.codecentric.boot.admin.config.EnableAdminServer;

@EnableAdminServer
@Configuration
@SpringBootApplication
            SpringBootAdminApplication public class {

```

```
    public static void main(String[] args) {
        SpringApplication.run(SpringBootAdminApplication.class, args);
    }

    @Configuration
    public static class SecurityConfig extends WebSecurityConfigurerAdapter {
        @Override
        protected void configure(HttpSecurity http) throws Exception {

            http.formLogin().loginPage("/login.html").loginProcessingUrl("/login").permitAll

            http.logout().logoutUrl("/logout");

            http.csrf().disable();


            http.authorizeRequests()
            .antMatchers("/login.html","/**/*.css","/img/**","/third-party/**")
            .permitAll();

            http.authorizeRequests().antMatchers("/**").authenticated();

```

```

            http.httpBasic();
        }
    }

}
```

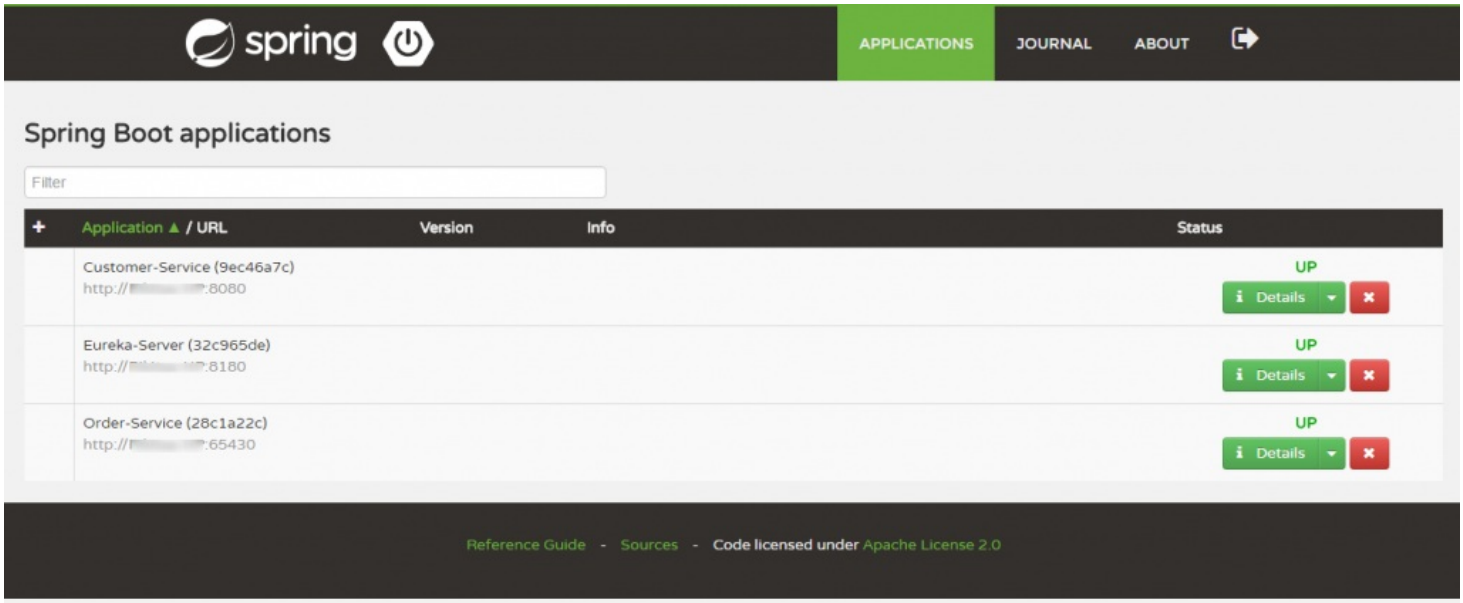
Let us create more Spring Boot applications to monitor via Spring Boot Admin server created in above steps. All the Spring Boot applications which will create now will be acted as Spring Boot Admin clients. To make application as Admin client, add the below dependency along with actuator dependency. In this demo I have created three applications like Eureka Server, Customer Service and Order Service.

```
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-client</artifactId>
    <version>1.5.1</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Add below property to application.properties file. This property tells that where the Spring Boot Admin server is running. Hence the clients will register with server.

```
spring.boot.admin.url=http:
```

Now If we start the Admin Server and other Spring Boot applications we can able to see all the admin clients information in the Admin server dashboard. As we started our admin server on 1111 port in this example we can see dash-board at http ://<host_name>:1111. Below is the screenshot of the Admin Server UI.



Detailed view of an application is given below. In this view we can see the tail of the log file, metrics, environment variables, log configuration where we can dynamically switch the log levels at the component level, root level or package level and other information.

spring

APPLICATIONS JOURNAL ABOUT

UP Customer-Service (9ec46a7c)

http://[redacted]:8080/health
http://[redacted]:8080
http://[redacted]:8080

Details | Log | Metrics | Environment | Logging | JMX | Threads | Audit | Trace | Heapdump

1 sec

Application

raw JSON

Health

raw JSON

Application	UP
Description	Spring Cloud Eureka Discovery Client
DiscoveryComposite	UP
Description	Spring Cloud Eureka Discovery Client
DiscoveryClient	UP
Description	Spring Cloud Eureka Discovery Client
Services	customer-service, order-service
Eureka	UP
Description	Remote status from Eureka server
Applications	
CUSTOMER-SERVICE	1
ORDER-SERVICE	1
DiskSpace	UP
Free	354.6G
Threshold	10M
RefreshScope	UP
Hystrix	UP

Memory

raw JSON

Memory (233.8M / 455.5M)	51.33%
Heap Memory (145.8M / 367.5M)	39.68%
Initial Heap	128M
Maximum Heap	1.8G
Non-Heap Memory (88M / 89.6M)	98.19%
Initial Non-Heap	2.4M
Maximum Non-Heap	unbounded

JVM

raw JSON

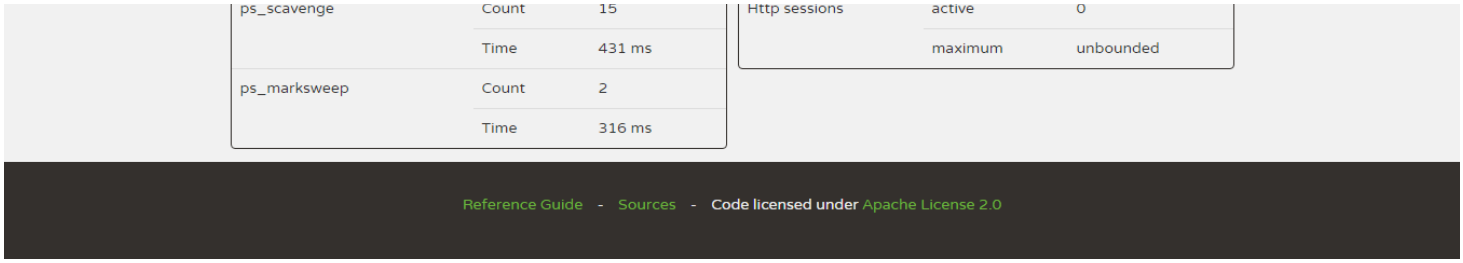
Uptime	00:10:33:29 [d:h:m:s]	
Systemload	-1.00 (last min. ⌀ runq-sz)	
Available Processors	4	
Classes	current loaded	9470
	total loaded	9470
	unloaded	0
Threads	current	40
	total started	259
	daemon	37
	peak	44

Garbage Collection

raw JSON

Servlet Container

raw JSON



Now we will see another feature called notifications from Spring Boot Admin. This will notify the administrators when the application status is DOWN or application status is coming UP. Spring Boot admin supports the below channels to notify the user.

- Email Notifications
- Pagerduty Notifications
- Hipchat Notifications
- Slack Notifications
- Let's Chat Notifications

In this article we will configure Slack notifications. Add the below properties to the Spring Boot Admin Server's application.properties file.

```
spring.boot.admin.notify.slack.webhook-url=https:
spring.boot.admin.notify.slack.message="*#{application.names *#{to.status}*"
```

With Spring Boot Admin we are managing all the applications. So we need to secure Spring Boot Admin UI with login feature. Let us enable login feature to Spring Boot Admin server. Here I am going with basic authentication. Add below maven dependencies to the Admin Server module.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-server-ui-login</artifactId>
    <version>1.5.1</version>
</dependency>
```

Add the below properties to the application.properties file.

```
security.user.name=admin
security.user.password=admin123
```

As we added security to the Admin Server, Admin clients should be able to connect to server by authenticating. Hence add the below properties to the Admin client's application.properties files.

```
spring.boot.admin.username=admin
```

```
spring.boot.admin.password=admin123
```

There are additional UI features like Hystrix, Turbine UI which we can enable to the dash-board. You can find more details here. The sample code created for this demonstration is available on Github.

Reference: Spring Boot Admin – Admin UI for administration of spring boot applications from our JCG partner Siva Janapati at the Smart Techie blog.