Spring Boot

Sang Shin
"Code with Passion!"

Topics

- What is and Why Spring Boot?
- Getting started with Spring Boot
- Building a Web app using Spring Boot
- Auto-configuration
- Application properties file
- External configuration
- *SpringApplication* class
- Actuator

What is & Why Spring Boot?

Things You Need for Building Web App

- Build tool (Maven, Gradle, Ant)
- Web framework (i.e. Spring MVC)
- Persistence (JPA, Spring Data)
- Template (JSP, Thymeleaf)
- Database server (MySQL)
- App server (Tomcat, Jetty)
- Logging (SLF4J)
- Testing (JUnit, Spring Test)
- ...

Manual Chores of Building Spring App

- Writing build script (Maven, Gradle)
- Version matching among modules
- Configuration of all necessary beans
- Setting up database server
- Setting up a runtime server (Tomcat)
- Writing and configuring management/monitoring features
- ...

What is and Why Spring Boot?

- Provide a radically faster and widely accessible getting started experience for all Spring application development
  - Automating manual chores (based on the idea of “convention over configuration”)
- Use best-practice defaults, but get out of the way quickly as requirements start to diverge from the defaults
- Provide a range of non-functional features that are common to majority of enterprise applications
  - embedded servers
  - metrics
  - health checks
  - externalized configuration
- Absolutely no code generation and no requirement for XML configuration

Spring Boot is made of

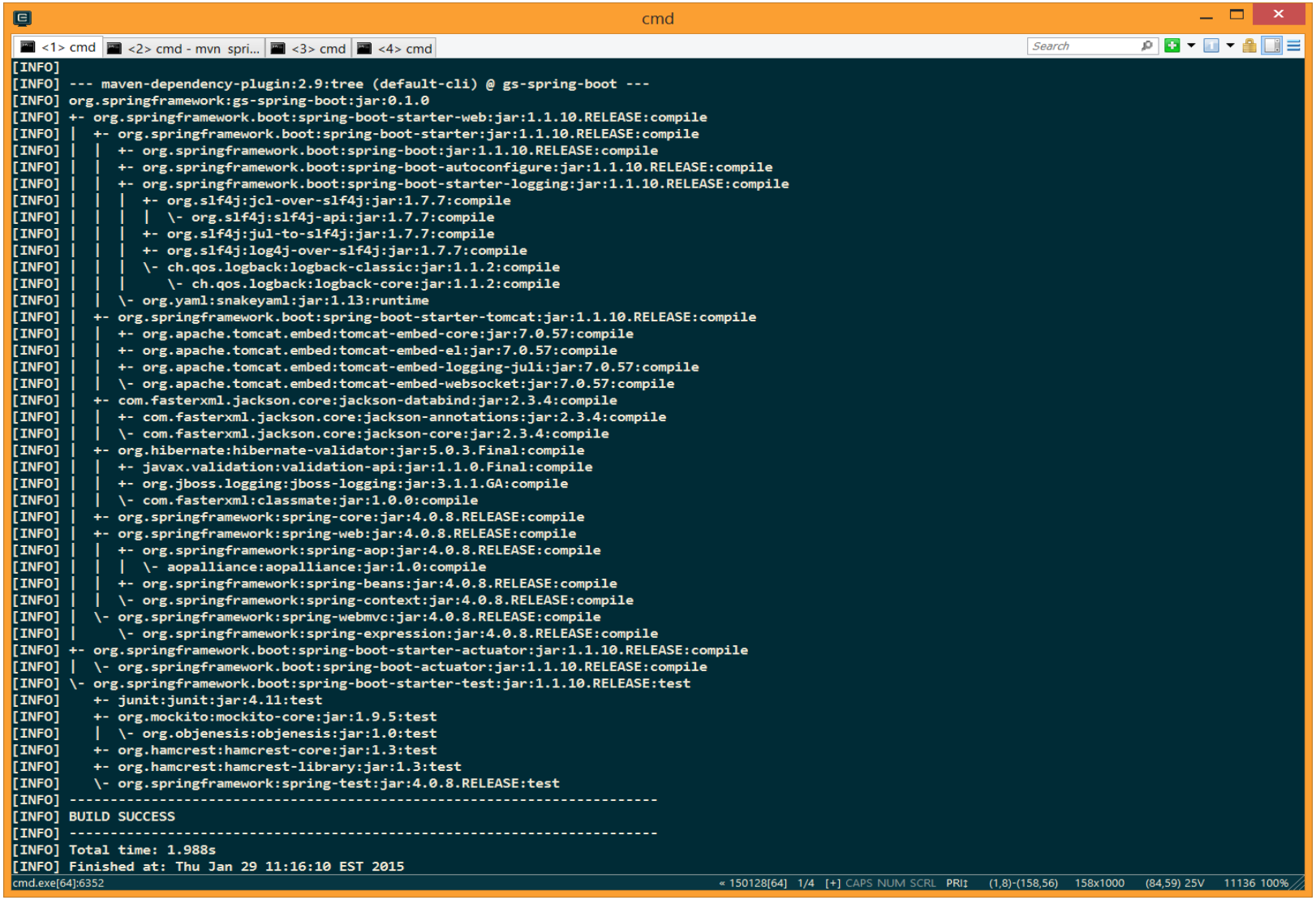
- Spring Boot Tools
  - Maven, Gradle, Custom Spring Boot loader
- Spring Boot Core
  - Base for other modules
- Spring Boot Auto Configuration
  - Handles auto-configuration with sensible defaults
- Spring Boot Actuator
  - Metrics, security, default error pages
- Spring Boot Starters (http://start.spring.io)
  - Quick starter projects – added Maven or Gradle dependencies
- Spring Boot CLI
  - Command line tools – start and stop Spring Boot application

Getting Started with Spring Boot

Maven pom.xml - Start with starter-parent

```
<groupId>com.example</groupId>
<artifactId>myproject</artifactId>
<version>0.0.1-SNAPSHOT</version>

<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.4.3.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>

<!-- Add typical dependencies for a web application -->
<dependencies>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
   </dependency>
</dependencies>

...
```

Dependencies added by Starter parent (mvn dependency: tree)



Examples of Spring Boot Child Starters

- spring-boot-starter-actuator
- spring-boot-starter-amqp
- spring-boot-starter-aop
- spring-boot-starter-batch
- spring-boot-starter-cloud-connectors
- spring-boot-starter-data-elasticsearch
- spring-boot-starter-data-gemfire
- spring-boot-starter-data-jpa
- spring-boot-starter-data-mongodb
- spring-boot-starter-data-rest
- spring-boot-starter-data-solr
- spring-boot-starter-jdbc

Spring Boot Starters are a set of convenient dependency descriptors that you can include in your application

Building a Web App using Spring Boot

Steps for Creating Spring MVC App

- Step #1: Create a project with a proper starter
- Step #2: Write controllers, domains, services, etc
- Step #3: Write Application class
- Step #4: Build and run

Step #1: Create a project with a starter

- Maven or Gradle project

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

Step #2: Write Code

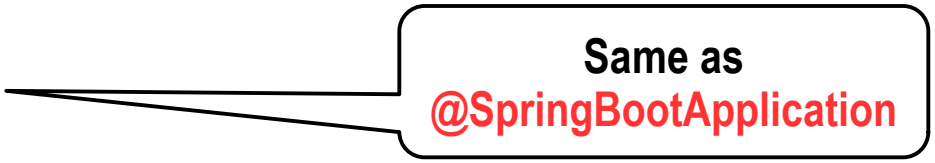
- Write Controller, Domain, Service classes

```
com
 +- jpassion
    +- myproject
       +- MainApplication.java
       +- AppConfiguration.java
       |
       +- domains
       |   +- Customer.java
       |   +- CustomerRepository.java
       |
       +- services
       |   +- CustomerService.java
       |
       +- controllers
           +- CustomerController.java
```

Step #3: Write Main Application Class

- The MainApplication.java file would declare the main method, along with the @SpringBootApplication

```
package com.example.myproject;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@EnableAutoConfiguration
@ComponentScan
public class MainApplication {

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

}
```

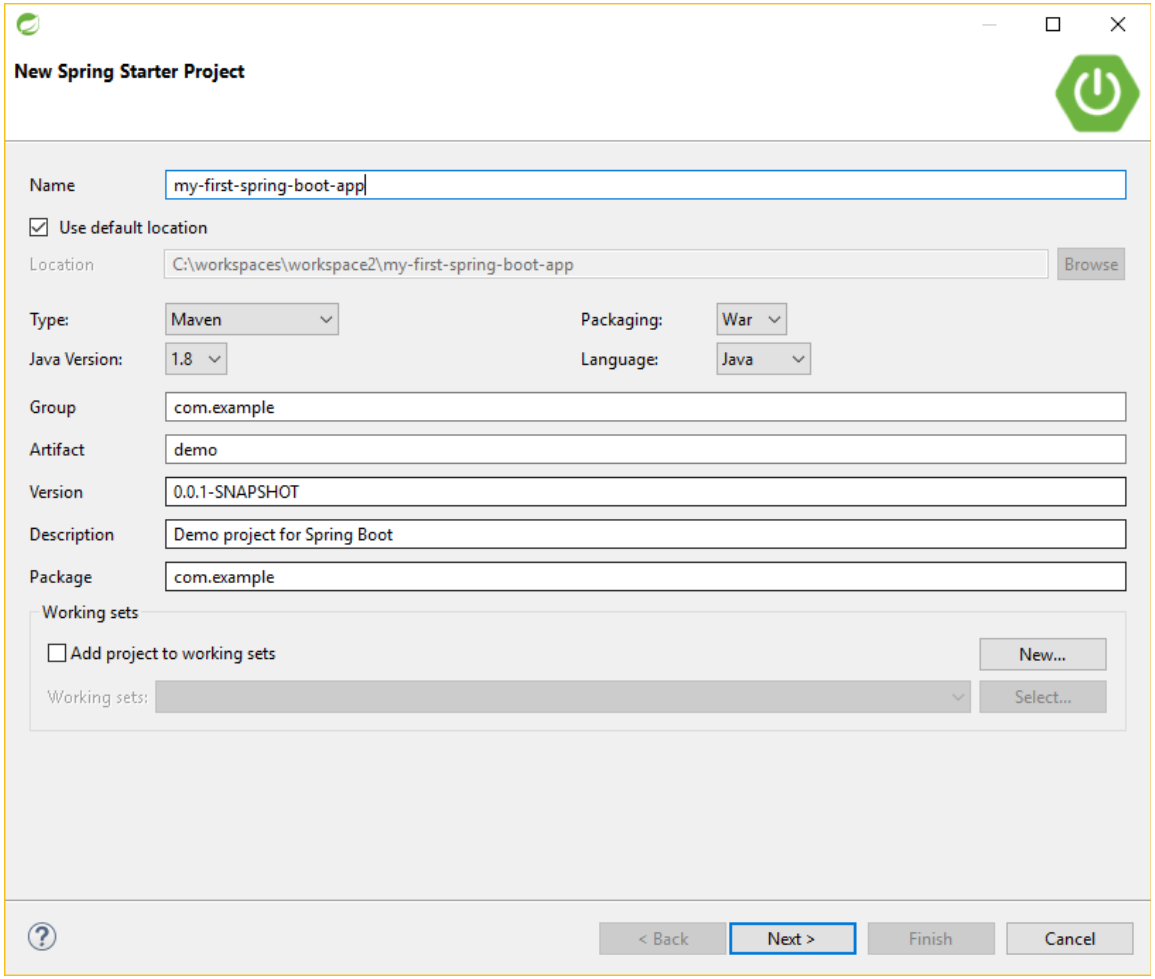


Step #4: Build and run

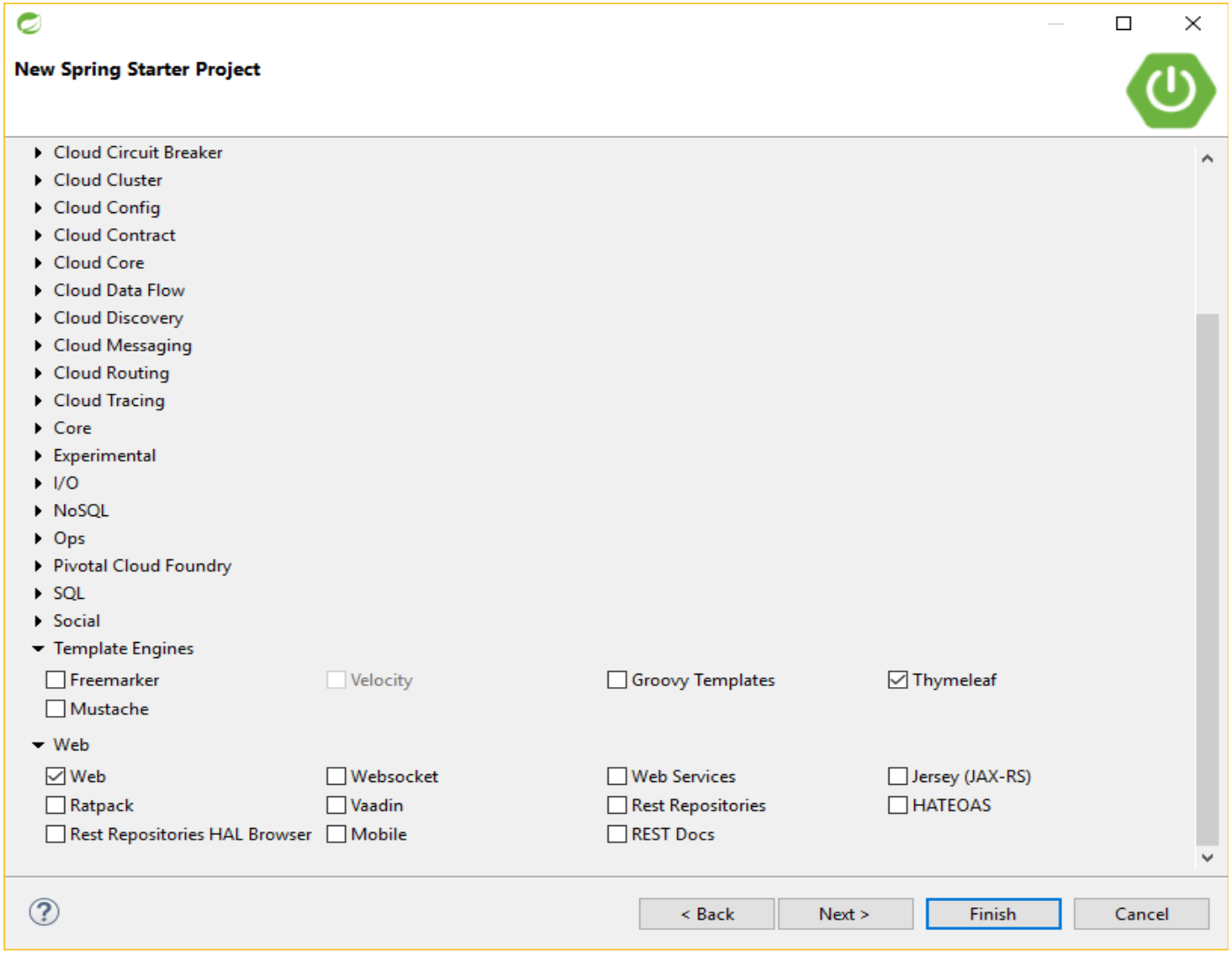
- If “war” file has been built, you can run the “war” file using “java” command - this war file can be deployed over any Servlet container
  - > mvn package
  - > java -jar target/mywebapp.war
- If “jar” file
  - > java -jar target/mywebapp.jar
- As a maven project
  - > mvn spring-boot:run

Getting Started with Spring Boot using STS

Create Starter Project



Create Starter Project

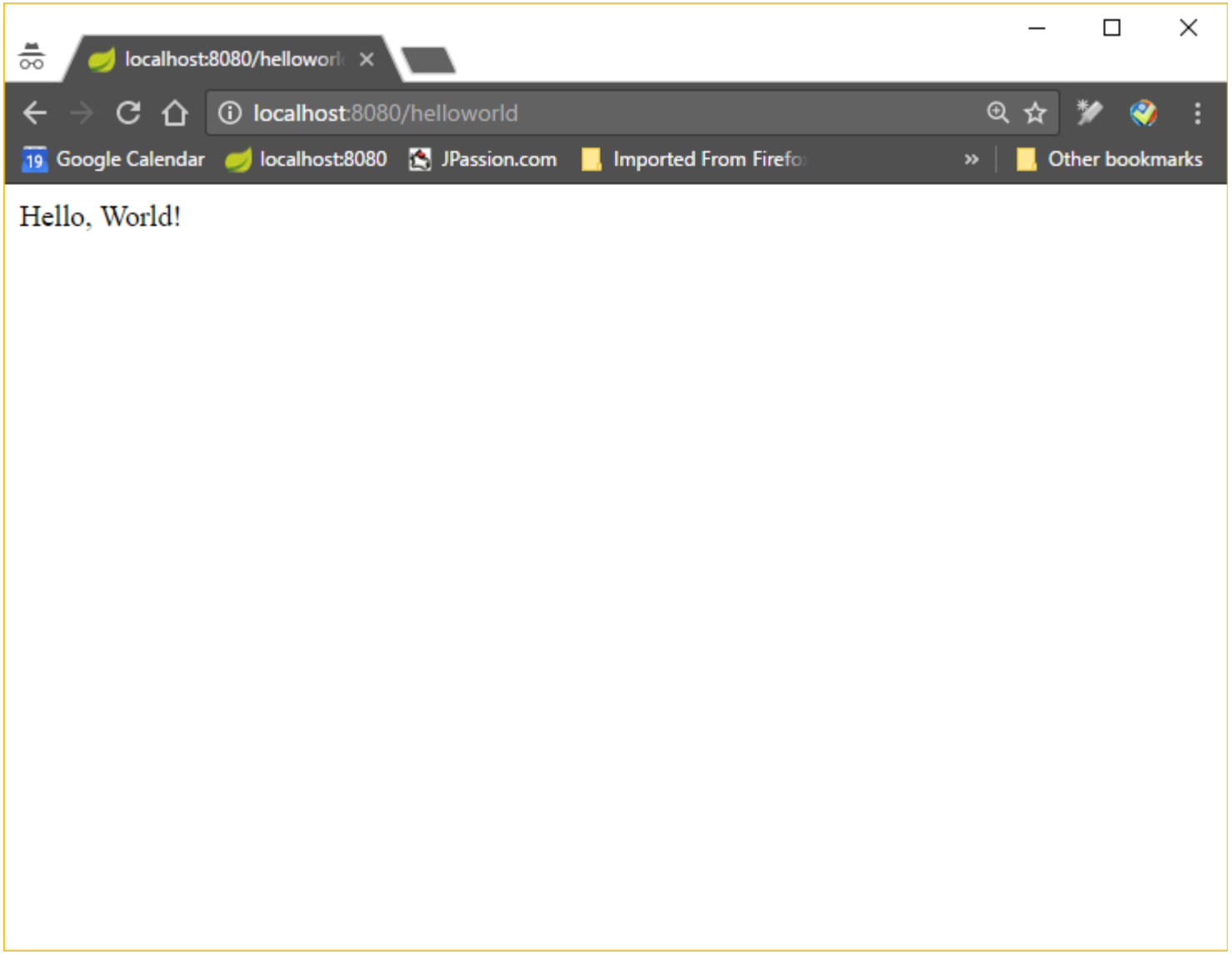


Add Code

- Write a simplest controller

```
package com.example;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @RequestMapping("helloworld")
    @ResponseBody
    public String getMessage(){
        return "Hello, World!";
    }

}
```

Run the application



Lab:

Exercise 1: Build a Spring Boot Application using STS

4979_spring_boot.zip

Auto-configuration

How does Auto-configuration work?

- Spring Boot auto-configuration attempts to automatically configure your Spring application based on the jar dependencies that you have added
  - If HSQLDB is on your classpath, and you have not manually configured any database connection beans, then HSQLDB in-memory database gets automatically configured
  - If *tomcat-embedded.jar* is on your classpath, then a *TomcatEmbeddedServletContainerFactory* gets automatically configured (unless you have defined your own *EmbeddedServletContainerFactory* bean)

- You need to enable auto-configuration
  - By adding the *@EnableAutoConfiguration* or *@SpringBootApplication* annotation to one of your *@Configuration* classes

Tuning Auto-configuration

- Auto-configuration tries to be as intelligent as possible and noninvasive
  - It will back-away as you define more of your own configuration - for example, if you add your own *DataSource* bean, the default embedded database support will back away
  - You can always manually exclude any configuration that you never want to apply
  - Auto-configuration is always applied after user-defined beans have been registered
- If you need to find out what auto-configuration is currently being applied, and why, starting your application with the *--debug* switch
  - This will log an auto-configuration report to the console.

@EnableAutoConfiguration

- Built-in Auto-configuration classes
  - @DataSourceAutoConfiguration
  - @JdbcTemplateAutoConfiguration
  - @HibernateJpaAutoConfiguration
  - @JpaRepositoriesAutoConfiguration
  - @WebMvcAutoConfiguration
  - …
- You can manually exclude configuration

```
@Configuration
@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})
public class MyConfiguration {

}
```

Application Properties File (application.properties)

Application Property Files

- SpringApplication will load properties from *application.properties* files in the following locations and add them to the Spring Environment
  - A /config subdirectory of the current directory
  - The current directory
  - A classpath /config package
  - The classpath root

```
# application.properties
app.name=MyApp
app.description=${app.name} is a Spring Boot application
```

External Configuration

External Configuration

- Spring Boot allows you to externalize your configuration so you can work with the same application code in different environments
- You can use properties files, YAML files, environment variables and command-line arguments to externalize configuration
- Property values can be injected directly into your beans using the *@Value* annotation, accessed via Spring's Environment abstraction or bound to structured objects

application.yaml

```yaml
contact:
    tel: 555-111-2222
    email: contact@jpassion.com
    web: http://jpassion.com
server:
    address: 127.0.0.1
    port: 8000
---
spring:
    profiles: development
server:
    address: 127.0.0.1
    port: 8080
---
spring:
    profiles: production
server:
    address: 127.0.0.1
    port: 8888
```

)

```java
@Value("${contact.tel}")
private String tel;

@Value("${contact.email}"

private String email;

@Value("${contact.web}")
private String web;
```

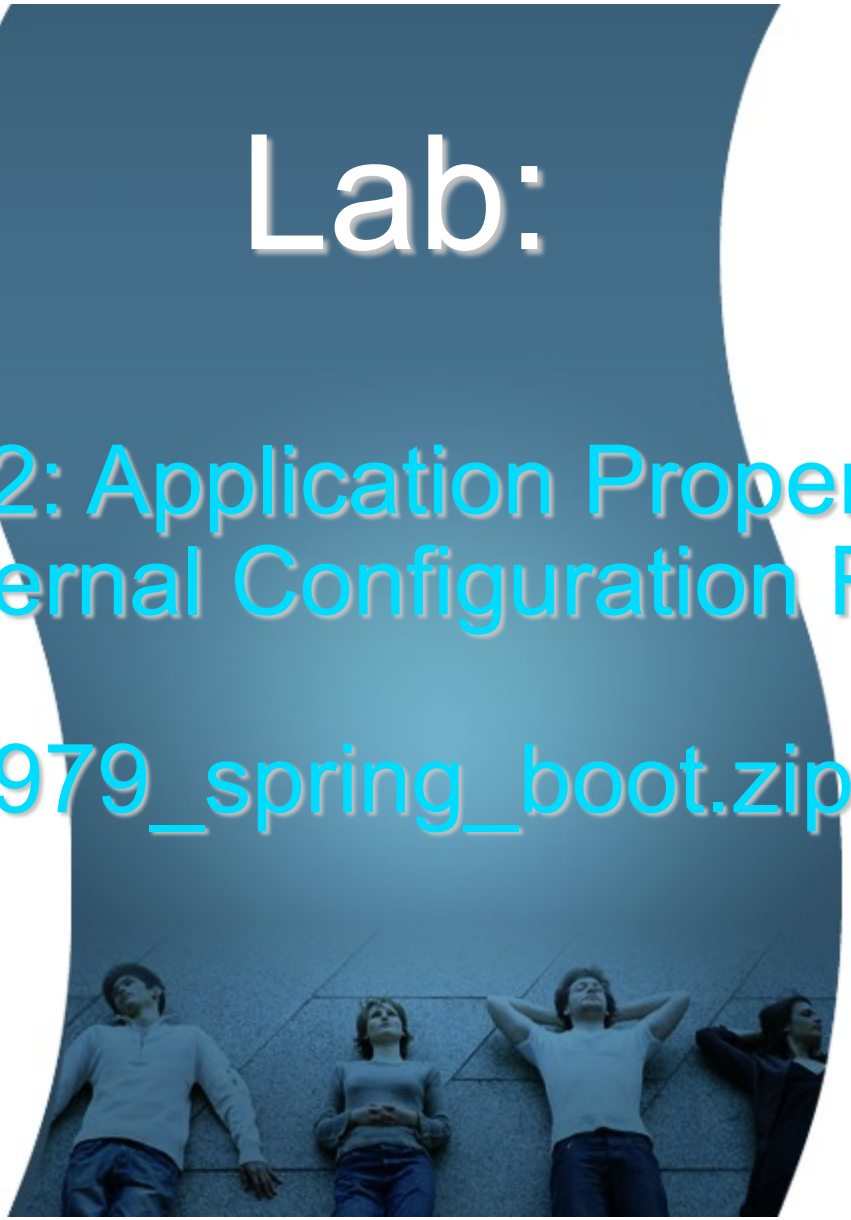
Property Order

1. Command line arguments.
2. JNDI attributes from java:comp/env.
3. Java System properties (System.getProperties()).
4. OS environment variables.
5. A RandomValuePropertySource that only has properties in random.*.
6. Profile-specific application properties outside of your packaged jar (application-{profile}.properties and YAML variants)
7. Profile-specific application properties packaged inside your jar (application-{profile}.properties and YAML variants)
8. Application properties outside of your packaged jar (application.properties and YAML variants).
9. Application properties packaged inside your jar (application.properties and YAML variants).
10. @PropertySource annotations on your @Configuration classes.
11. Default properties (specified using SpringApplication.setDefaultProperties).

Lab:

Exercise 2: Application Properties File & External Configuration Files

4979_spring_boot.zip



SpringApplication Class

What does *SpringApplication* class do?

- When *SpringApplication.run(MyConfiguration.class. args)* is called, it bootstraps and launches a Spring application from a Java main method by performing the following steps
  - Create an appropriate ApplicationContext instance (depending on your classpath) - By default, an *AnnotationConfigApplicationContext* (for Java application) or *AnnotationConfigEmbeddedWebApplicationContext* (for Web application) will be used
  - Register a *CommandLinePropertySource* to expose command line arguments as Spring properties
  - Refresh the application context, loading all singleton beans
  - Trigger any *CommandLineRunner* beans

How do you run the application?

- In most circumstances, the static run(Object, String[]) method can be called directly from your main method to bootstrap your application
- Provides a convenient way to bootstrap a Spring application that will be started from a main() method

```
// Example #1
public static void main(String[] args) {
    SpringApplication.run(MySpringConfiguration.class, args);
}

// Example #2
public static void main(String[] args) {
    SpringApplication app = new SpringApplication(MySpringConfiguration.class);
    app.setBanner(new MyBanner());
    app.run(args);
}
```

Lab:

Exercise 3: Spring Application Class 4979_spring_boot.zip

Actuator

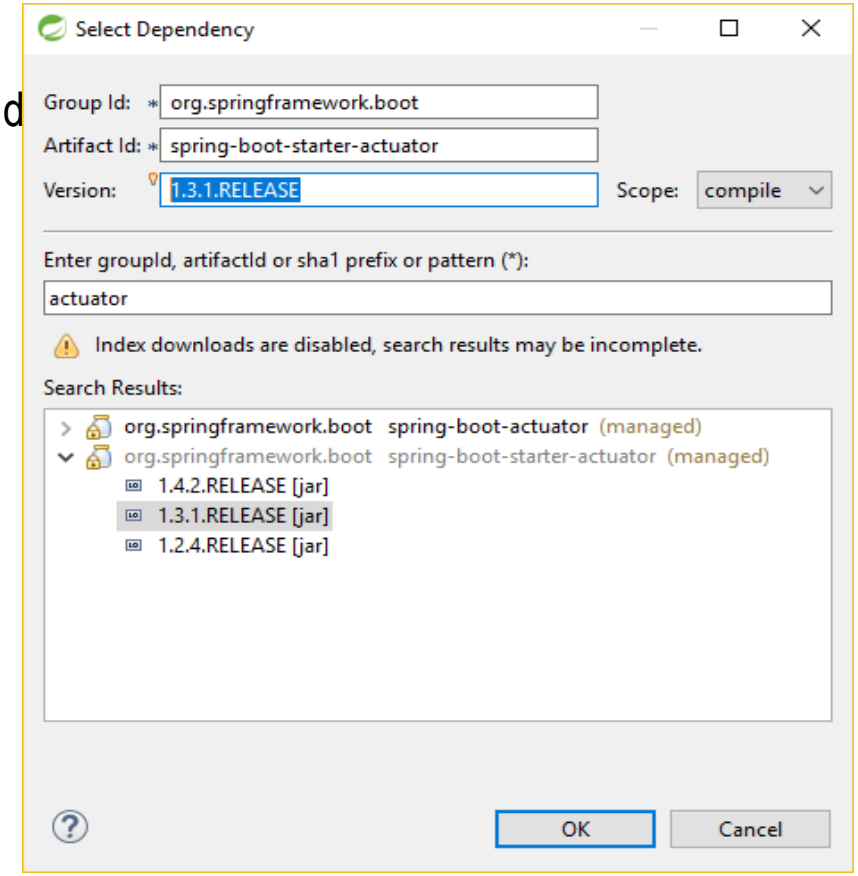
Monitoring and Managing Application

- Spring Boot includes a number of additional features to help you monitor and manage your application when it's pushed to production
- You can choose to manage and monitor your application using
  - HTTP endpoints, with JMX or even by remote shell (SSH or Telnet)
- Auditing, health and metrics gathering can be automatically applied to your application.
- Actuator HTTP endpoints are only available with a Spring MVC-based application

Adding Actuator to Maven App

- Add the following ‘starter’ dependency

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId
    <version>1.3.1.RELEASE</version>
</dependency>
```



Endpoints

- Actuator endpoints allow you to monitor and interact with your application
- Spring Boot includes a number of built-in endpoints and you can also add your own
  - autoconfig - displays an auto-configuration report
  - beans - displays a complete list of all Spring beans in the app
  - configprops - displays a collated list of all @ConfigurationProperties
  - dump - performs a thread dump
  - env - exposes properties from Spring's ConfigurableEnvironment
  - health - shows application health information
  - info - displays arbitrary application info
  - mappings – displays endpoints

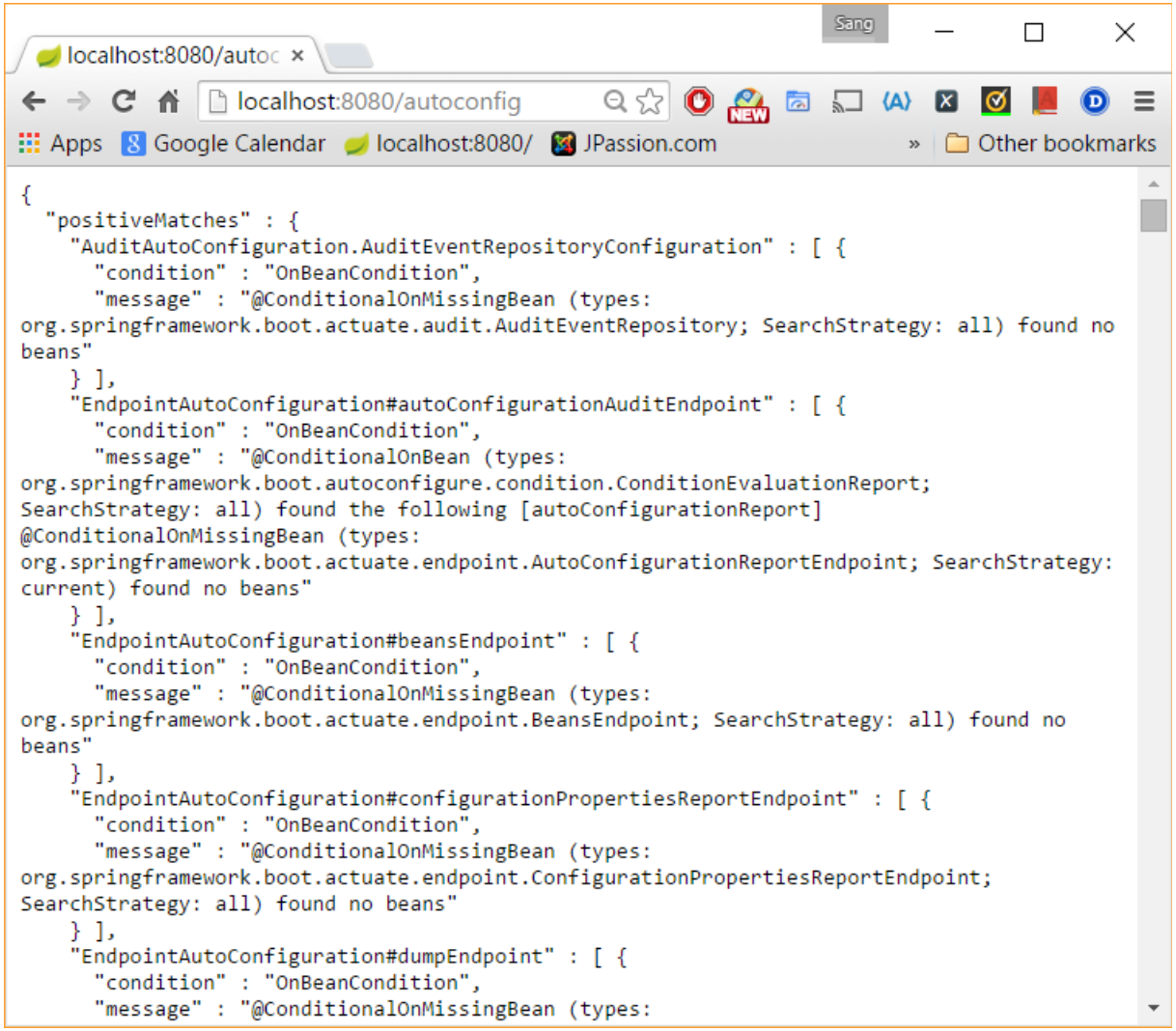
Endpoints

- Built-in endpoints (continued from previous slide)
  - metrics - shows ‘metrics’ information for the current application
  - shutdown - allows the application to be gracefully shutdown (not enabled by default)
  - trace - displays trace information (by default the last few HTTP requests)

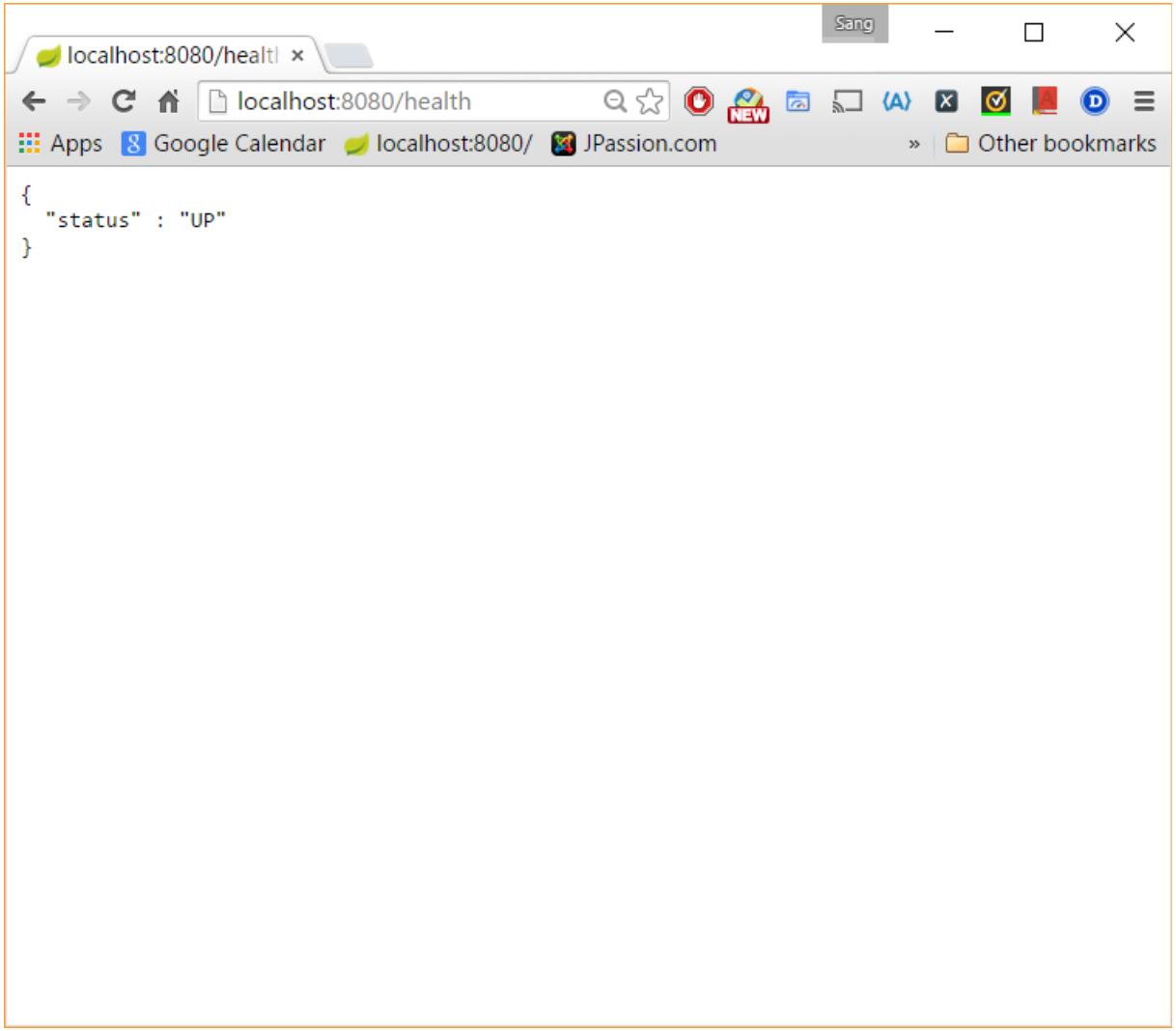
- Note: for pretty JSON print, add the following to *application.properties* file

```
spring.jackson.serialization.INDENT_OUTPUT=true
```

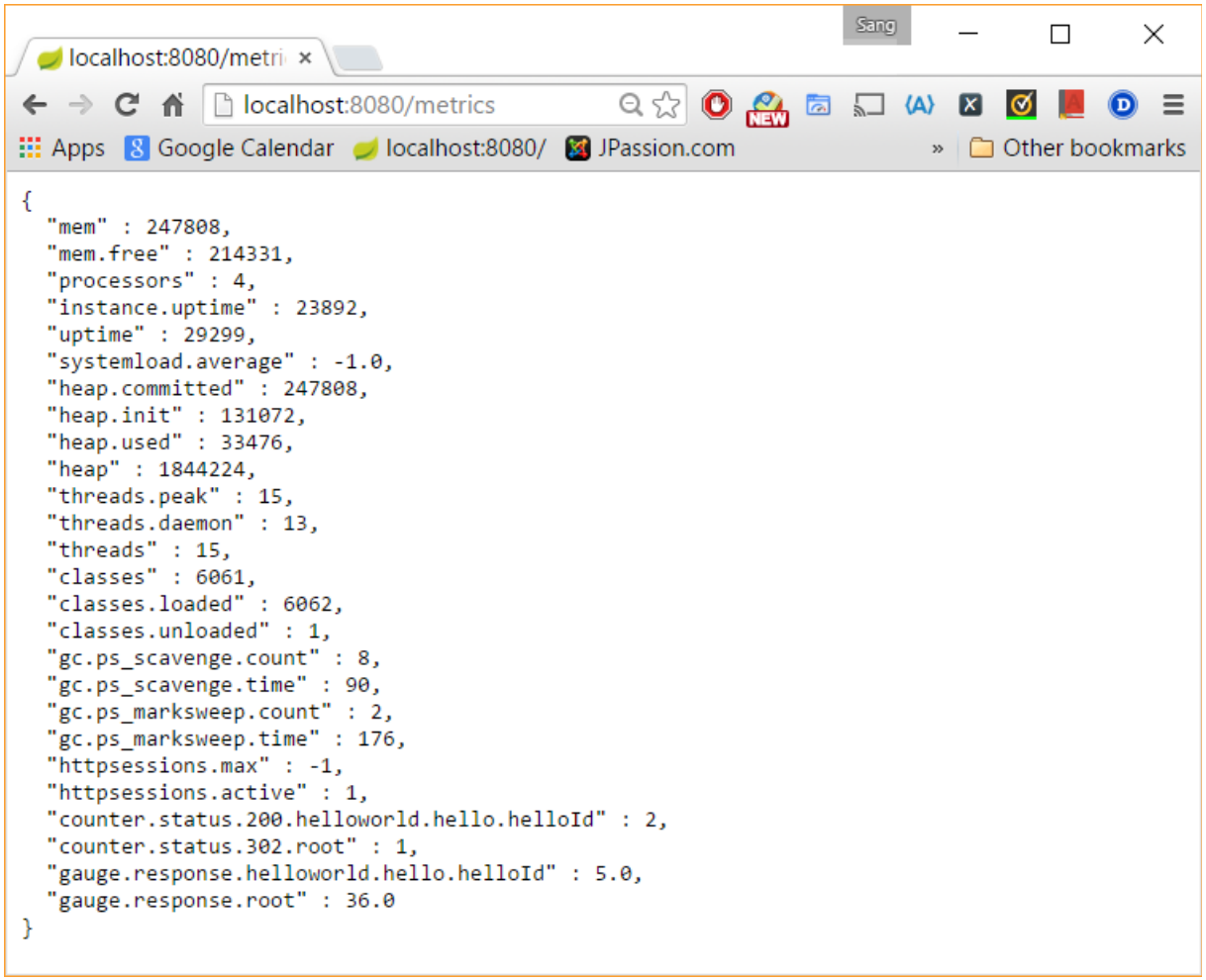
autoconfig


Sang
localhost:8080/autoc
localhost:8080/autoconfig
Apps
Google Calendar
localhost:8080/
JPassion.com
Other bookmarks
{
"positiveMatches" : {
"AuditAutoConfiguration.AuditEventRepositoryConfiguration" : [{
"condition" : "OnBeanCondition",
"message" : "@ConditionalOnMissingBean (types:
org.springframework.boot.actuate.audit.AuditEventRepository; SearchStrategy: all) found no
beans"
}],
"EndpointAutoConfiguration#autoConfigurationAuditEndpoint" : [{
"condition" : "OnBeanCondition",
"message" : "@ConditionalOnBean (types:
org.springframework.boot.autoconfigure.condition.ConditionEvaluationReport;
SearchStrategy: all) found the following [autoConfigurationReport]
@ConditionalOnMissingBean (types:
org.springframework.boot.actuate.endpoint.AutoConfigurationReportEndpoint; SearchStrategy:
current) found no beans"
}],
"EndpointAutoConfiguration#beansEndpoint" : [{
"condition" : "OnBeanCondition",
"message" : "@ConditionalOnMissingBean (types:
org.springframework.boot.actuate.endpoint.BeansEndpoint; SearchStrategy: all) found no
beans"
}],
"EndpointAutoConfiguration#configurationPropertiesReportEndpoint" : [{
"condition" : "OnBeanCondition",
"message" : "@ConditionalOnMissingBean (types:
org.springframework.boot.actuate.endpoint.ConfigurationPropertiesReportEndpoint;
SearchStrategy: all) found no beans"
}],
"EndpointAutoConfiguration#dumpEndpoint" : [{
"condition" : "OnBeanCondition",
"message" : "@ConditionalOnMissingBean (types:

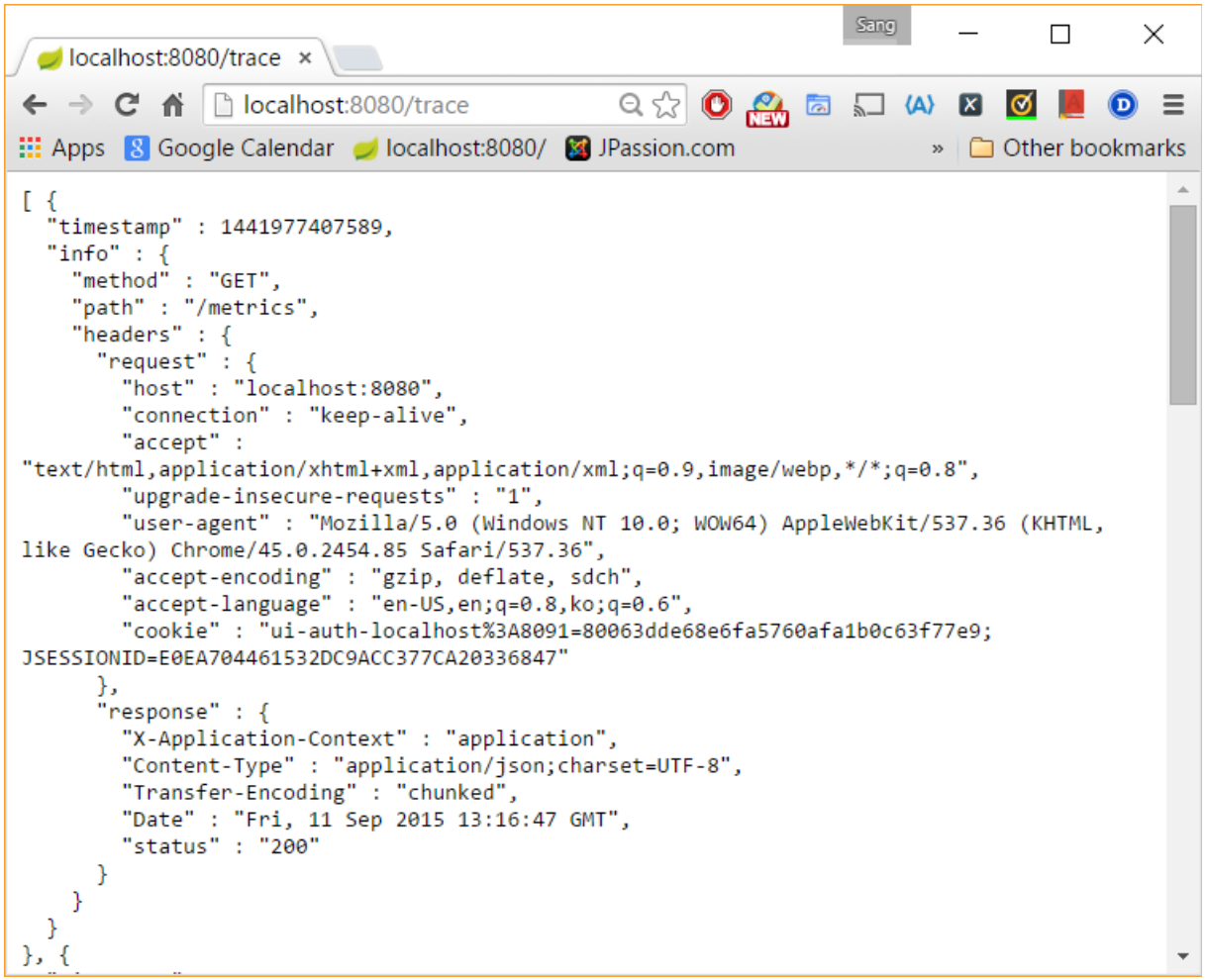
health


Sang
localhost:8080/healtl
localhost:8080/health
Apps
Google Calendar
localhost:8080/
JPassion.com
Other bookmarks
{
"status" : "UP"
}

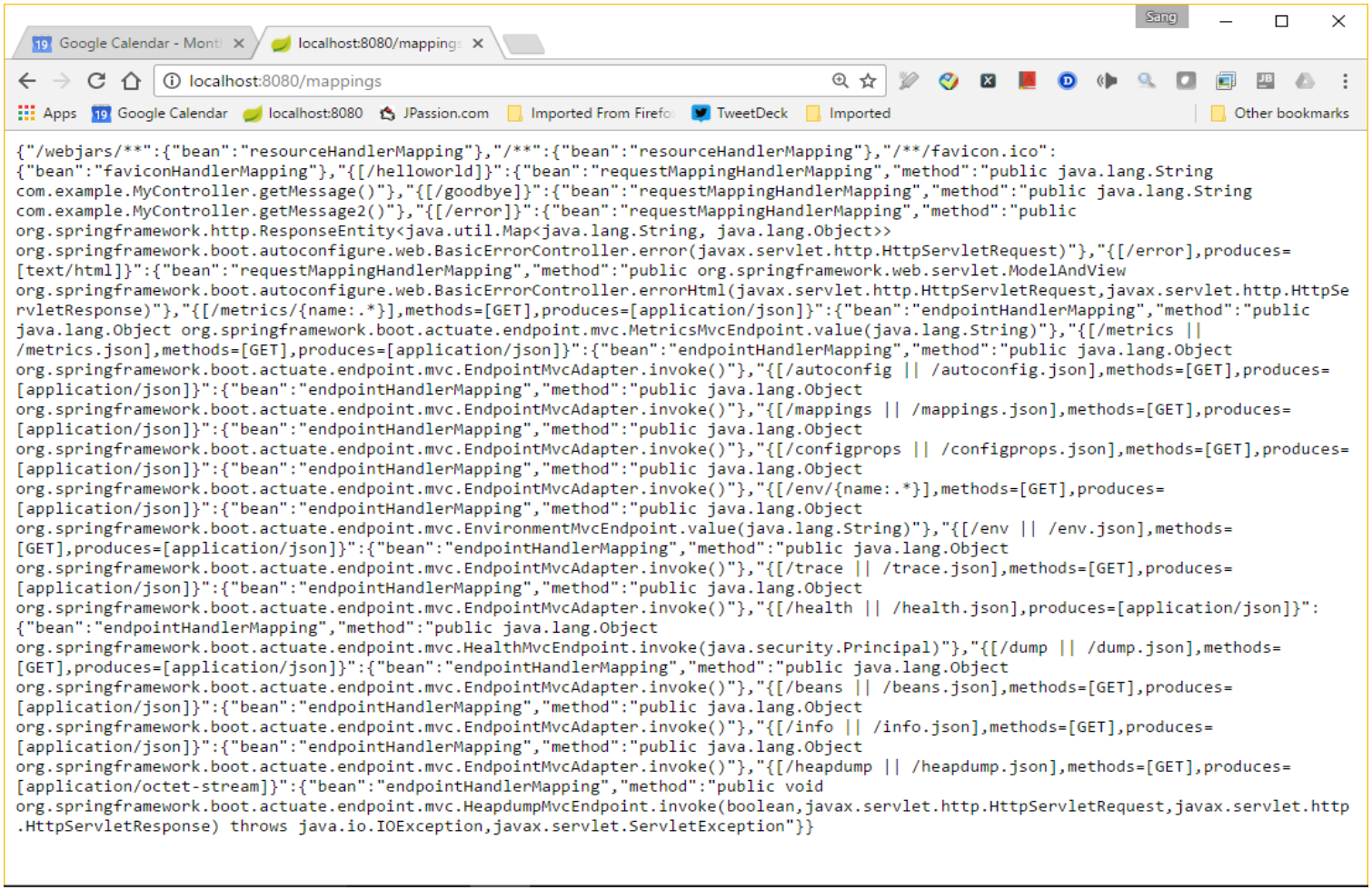
Metrics


Sang
localhost:8080/metri
localhost:8080/metrics
Apps
Google Calendar
localhost:8080/
JPassion.com
Other bookmarks
{
"mem" : 247808,
"mem.free" : 214331,
"processors" : 4,
"instance.uptime" : 23892,
"uptime" : 29299,
"systemload.average" : -1.0,
"heap.committed" : 247808,
"heap.init" : 131072,
"heap.used" : 33476,
"heap" : 1844224,
"threads.peak" : 15,
"threads.daemon" : 13,
"threads" : 15,
"classes" : 6061,
"classes.loaded" : 6062,
"classes.unloaded" : 1,
"gc.ps_scavenge.count" : 8,
"gc.ps_scavenge.time" : 90,
"gc.ps_marksweep.count" : 2,
"gc.ps_marksweep.time" : 176,
"httpsessions.max" : -1,
"httpsessions.active" : 1,
"counter.status.200.helloworld.hello.helloId" : 2,
"counter.status.302.root" : 1,
"gauge.response.helloworld.hello.helloId" : 5.0,
"gauge.response.root" : 36.0
}

Trace


Sang
localhost:8080/trace
localhost:8080/trace
Apps
Google Calendar
localhost:8080/
JPassion.com
Other bookmarks
[{
"timestamp" : 1441977407589,
"info" : {
"method" : "GET",
"path" : "/metrics",
"headers" : {
"request" : {
"host" : "localhost:8080",
"connection" : "keep-alive",
"accept" :
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8",
"upgrade-insecure-requests" : "1",
"user-agent" : "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/45.0.2454.85 Safari/537.36",
"accept-encoding" : "gzip, deflate, sdch",
"accept-language" : "en-US,en;q=0.8,ko;q=0.6",
"cookie" : "ui-auth-localhost%3A8091=80063dde68e6fa5760afa1b0c63f77e9;
JSESSIONID=E0EA704461532DC9ACC377CA20336847"
},
"response" : {
"X-Application-Context" : "application",
"Content-Type" : "application/json;charset=UTF-8",
"Transfer-Encoding" : "chunked",
"Date" : "Fri, 11 Sep 2015 13:16:47 GMT",
"status" : "200"
}
}
}
}, {

Mappings


Google Calendar - Mont
localhost:8080/mapping
localhost:8080/mappings
Apps
Google Calendar
localhost:8080
JPassion.com
Imported From Firefo
TweetDeck
Imported
Other bookmarks
{"/webjars/**":{"bean":"resourceHandlerMapping"},"/**":{"bean":"resourceHandlerMapping"},"/**/favicon.ico":
{"bean":"faviconHandlerMapping"},"{[/helloworld]}":{"bean":"requestMappingHandlerMapping","method":"public java.lang.String
com.example.MyController.getMessage()"},"{[/goodbye]}":{"bean":"requestMappingHandlerMapping","method":"public java.lang.String
com.example.MyController.getMessage2()"},"{[/error]}":{"bean":"requestMappingHandlerMapping","method":"public
org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>>
org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequest)"},"{[/error],produces=
[text/html]}":{"bean":"requestMappingHandlerMapping","method":"public org.springframework.web.servlet.ModelAndView
org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpSe
rvletResponse)"},"{[/metrics/{name:.*}],methods=[GET],produces=[application/json]}":{"bean":"endpointHandlerMapping","method":"public
java.lang.Object org.springframework.boot.actuate.endpoint.mvc.MetricsMvcEndpoint.value(java.lang.String)"},"{[/metrics ||
/metrics.json],methods=[GET],produces=[application/json]}":{"bean":"endpointHandlerMapping","method":"public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke()"},"{[/autoconfig || /autoconfig.json],methods=[GET],produces=
[application/json]}":{"bean":"endpointHandlerMapping","method":"public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke()"},"{[/mappings || /mappings.json],methods=[GET],produces=
[application/json]}":{"bean":"endpointHandlerMapping","method":"public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke()"},"{[/configprops || /configprops.json],methods=[GET],produces=
[application/json]}":{"bean":"endpointHandlerMapping","method":"public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke()"},"{[/env/{name:.*}],methods=[GET],produces=
[application/json]}":{"bean":"endpointHandlerMapping","method":"public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EnvironmentMvcEndpoint.value(java.lang.String)"},"{[/env || /env.json],methods=
[GET],produces=[application/json]}":{"bean":"endpointHandlerMapping","method":"public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke()"},"{[/trace || /trace.json],methods=[GET],produces=
[application/json]}":{"bean":"endpointHandlerMapping","method":"public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke()"},"{[/health || /health.json],produces=[application/json]}":
{"bean":"endpointHandlerMapping","method":"public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.HealthMvcEndpoint.invoke(java.security.Principal)"},"{[/dump || /dump.json],methods=
[GET],produces=[application/json]}":{"bean":"endpointHandlerMapping","method":"public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke()"},"{[/beans || /beans.json],methods=[GET],produces=
[application/json]}":{"bean":"endpointHandlerMapping","method":"public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke()"},"{[/info || /info.json],methods=[GET],produces=
[application/json]}":{"bean":"endpointHandlerMapping","method":"public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke()"},"{[/heapdump || /heapdump.json],methods=[GET],produces=
[application/octet-stream]}":{"bean":"endpointHandlerMapping","method":"public void
org.springframework.boot.actuate.endpoint.mvc.HeapdumpMvcEndpoint.invoke(boolean,javax.servlet.http.HttpServletRequest,javax.servlet.http
.HttpServletResponse) throws java.io.IOException,javax.servlet.ServletException"}}

Lab:

Exercise 4: Actuator
4979_spring_boot.zip

Code with Passion!