**Netflix Hystrix: Hystrix is a latency and fault tolerance library designed to stop cascading failure and enable resilience in complex distributed systems where failure is inevitable.**

**1- DISCOVERY SERVER**

**package** io.schultz.dustin;  
  
**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
**import** org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  
  
@EnableEurekaServer  
@SpringBootApplication  
**public class** DiscoveryServerApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(DiscoveryServerApplication.**class**, args);  
 }  
}

…

**application.properties**

**spring.application.name**=**discovery-server  
eureka.client.register-with-eureka**=**false  
eureka.client.fetch-registry**=**false  
server.port**=**8761**

…

**pom.xml**

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**groupId**>io.schultz.dustin</**groupId**>  
 <**artifactId**>discovery-server</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**packaging**>jar</**packaging**>  
  
 <**name**>discovery-server</**name**>  
 <**description**>Demo project for Spring Boot</**description**>  
  
 <**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>1.4.2.RELEASE</**version**>  
 <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**>  
  
 <**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**project.reporting.outputEncoding**>UTF-8</**project.reporting.outputEncoding**>  
 <**java.version**>1.8</**java.version**>  
 </**properties**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-actuator</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-eureka-server</**artifactId**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-devtools</**artifactId**>  
 <**scope**>runtime</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 </**dependencies**>  
  
 <**dependencyManagement**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-dependencies</**artifactId**>  
 <**version**>Camden.SR2</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 </**dependencies**>  
 </**dependencyManagement**>  
  
 <**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-maven-plugin</**artifactId**>  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
  
  
</**project**>

**2- WEATHER SERVICE**

**package** io.schultz.dustin.weatherservice;  
  
**import** java.util.concurrent.ThreadLocalRandom;  
  
**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
**import** org.springframework.cloud.client.discovery.EnableDiscoveryClient;  
**import** org.springframework.web.bind.annotation.GetMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
  
@SpringBootApplication  
@RestController  
**@EnableDiscoveryClient  
public class** WeatherServiceApplication {  
   
 **private** String[] **weather** = **new** String[] { **"sunny"**, **"cloudy"**, **"rainy"**, **"windy"** };  
   
 **public static void** main(String[] args) {  
 SpringApplication.*run*(WeatherServiceApplication.**class**, args);  
 }  
   
 @GetMapping(**"/weather"**)  
 **public** String getWeather() {  
 **int** rand = ThreadLocalRandom.*current*().nextInt(0, 4);  
 **return weather**[rand];  
 }  
}

…

**application.properties**

**server.port**=**9000  
spring.application.name**=**weather-service  
eureka.client.service-url.defaultZone**=**http://localhost:8761/eureka**

…

<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-eureka</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
</**dependencies**>

**3- WEATHER APP**

**package** io.schultz.dustin.weatherapp;  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
**import** org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;  
**import** org.springframework.cloud.client.discovery.EnableDiscoveryClient;  
**import** org.springframework.cloud.client.loadbalancer.LoadBalanced;  
**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.web.bind.annotation.GetMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
**import** org.springframework.web.client.RestTemplate;  
  
@SpringBootApplication  
**@EnableCircuitBreaker  
@EnableDiscoveryClient  
@RestController  
public class WeatherAppApplication** {  
   
 **@Autowired** **private** WeatherService **weatherService**;  
  
 **public static void main**(String[] args) {  
 **SpringApplication**.*run*(WeatherAppApplication.**class**, args);  
 }  
   
 **@GetMapping("/current/weather")** **public** String **getWeather**() {  
 **return "The current weather is "** + **weatherService**.**getWeather**();  
 }  
   
 **@Bean  
 @LoadBalanced** **public** RestTemplate restTemplate() {  
 **return new** RestTemplate();  
 }  
}

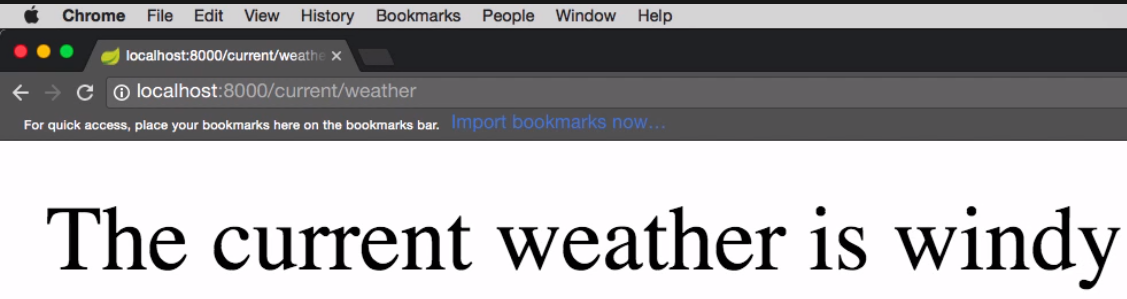
…

**package** io.schultz.dustin.weatherapp;  
  
**import** javax.inject.Inject;  
  
**import** org.springframework.stereotype.Service;  
**import** org.springframework.web.client.RestTemplate;  
  
**import** com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;  
  
**@Service  
public class WeatherService** {  
  
 **@Inject** **private** RestTemplate **restTemplate**;  
   
 **@HystrixCommand(fallbackMethod = "unknown")** **public** String **getWeather**() {  
 **return restTemplate**.**getForEntity**(**"http://weather-service/weather"**,   
 String.**class**)  
 .**getBody**();  
 }  
   
 **public** String **unknown**() {  
 **return "unknown"**;  
 }  
}

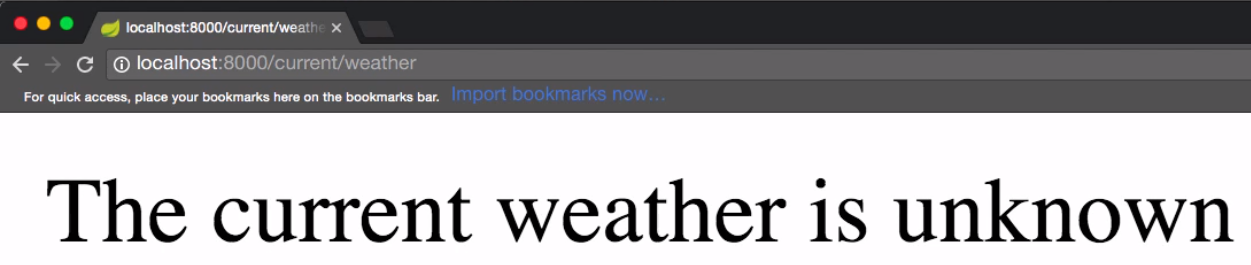
…

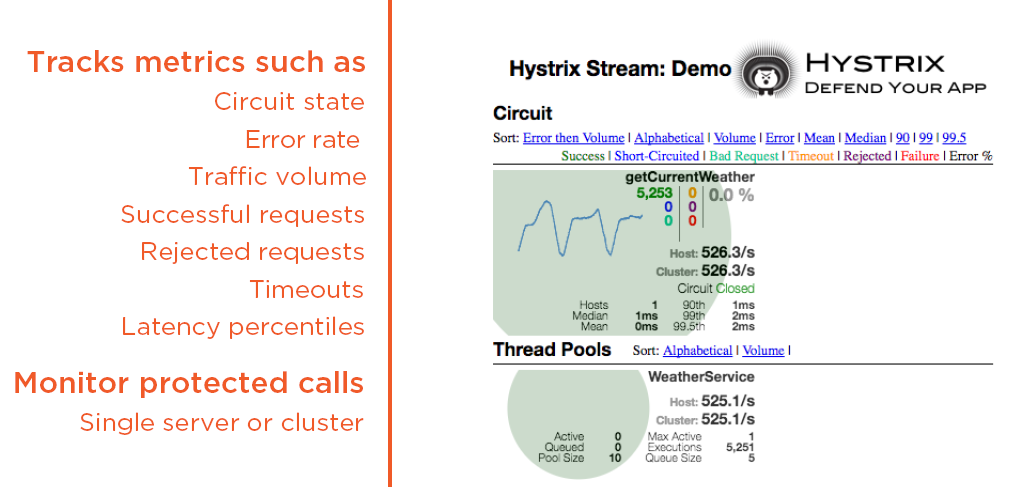
**application.properties**

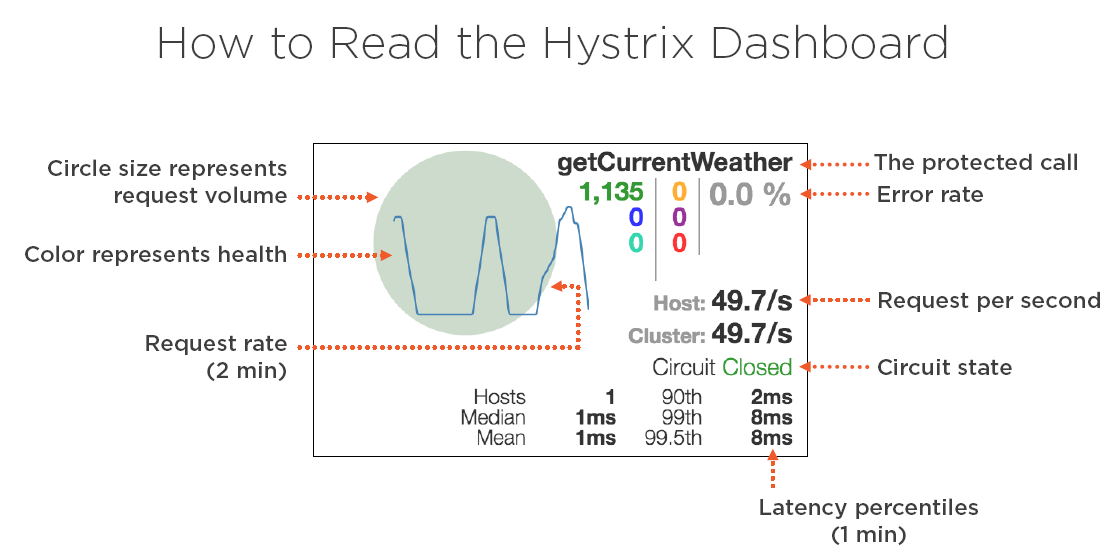
**server.port**=**8000  
spring.application.name**=**weather-app  
eureka.client.service-url.defaultZone**=**http://localhost:8761/eureka**



\*\*\* Si paro el weather service:





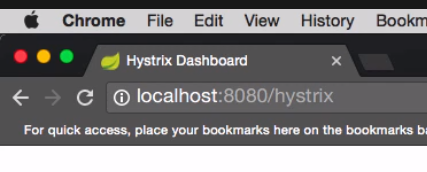


**4- Hystrix dashboard**

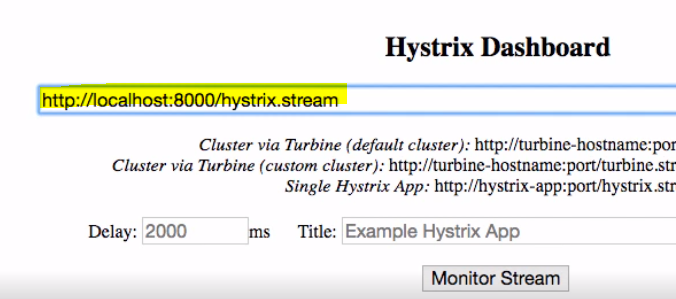
**package** io.schultz.dustin.hystrixdashboard;  
**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
**import** org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard;  
  
**@SpringBootApplication  
@EnableHystrixDashboard  
public class HystrixDashboardApplication** {  
  
 **public static void main**(String[] args) {  
 **SpringApplication**.*run*(HystrixDashboardApplication.**class**, args);  
 }  
}

<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-hystrix-dashboard</**artifactId**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
</**dependencies**>

\*\*\* si la corro me tiene que mostrar el dashboard aquí:



\*\*\* Me voy a la app, la refresco 10 veces llamando a su método y después me voy al dashboard y escribo:



**5- Turbine**

**package** io.schultz.dustin;  
  
**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
**import** org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  
  
@EnableEurekaServer  
@SpringBootApplication  
**public class** DiscoveryServerApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(DiscoveryServerApplication.**class**, args);  
 }  
}

…

**application.properties**

**spring.application.name**=**discovery-server  
eureka.client.register-with-eureka**=**false  
eureka.client.fetch-registry**=**false  
server.port**=**8761**

<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-turbine</**artifactId**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
</**dependencies**>

[**Spring Boot: Efficient Development, Configuration, and Deployment**](https://app.pluralsight.com/courses/a5598dd9-72ea-4b85-b350-bd96b9d229bc/table-of-contents) **--- Dustin Schultz**

**package** schultz.dustin.io;  
  
**import** org.springframework.beans.factory.annotation.Value;  
**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;  
**import** org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  
**import** org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;  
  
**import** javax.annotation.PostConstruct;  
**import** java.io.File;  
  
@SpringBootApplication  
**public class** JustGifItApplication {  
  
 @Value(**"${multipart.location}/gif/"**)  
 **private** String **gifLocation**;  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(JustGifItApplication.**class**, args);  
 }  
  
 @PostConstruct  
 **private void** init() {  
 File gifFolder = **new** File(**gifLocation**);  
 **if** (!gifFolder.exists()) {  
 gifFolder.mkdir();  
 }  
 }  
  
 @Bean  
 **public** WebMvcConfigurer webMvcConfigurer() {  
 **return new** WebMvcConfigurerAdapter() {  
 @Override  
 **public void** addResourceHandlers(ResourceHandlerRegistry registry) {  
 registry.addResourceHandler(**"/gif/\*\*"**)  
 .addResourceLocations(**"file:"** + **gifLocation**);  
 **super**.addResourceHandlers(registry);  
 }  
 };  
 }  
}

…

**package** schultz.dustin.io.controller;  
  
**import** com.madgag.gif.fmsware.AnimatedGifEncoder;  
**import** org.bytedeco.javacv.FFmpegFrameGrabber;  
**import** org.bytedeco.javacv.FrameGrabber;  
**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
**import** org.springframework.beans.factory.annotation.Value;  
**import** org.springframework.http.MediaType;  
**import** org.springframework.web.bind.annotation.\*;  
**import** org.springframework.web.multipart.MultipartFile;  
**import** schultz.dustin.io.services.ConverterService;  
**import** schultz.dustin.io.services.GifEncoderService;  
**import** schultz.dustin.io.services.VideoDecoderService;  
  
**import** javax.inject.Inject;  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.lang.invoke.MethodHandles;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
  
@RestController  
**public class** UploadController {  
  
 **private final static** Logger ***log*** = LoggerFactory.*getLogger*(MethodHandles.*lookup*()  
 .lookupClass());  
  
 @Value(**"${multipart.location}"**)  
 **private** String **location**;  
  
 @Inject  
 **private** ConverterService **converterService**;  
  
 @Inject  
 **private** GifEncoderService **gifEncoderService**;  
  
 @Inject  
 **private** VideoDecoderService **videoDecoderService**;  
  
 @RequestMapping(value = **"/upload"**, method = RequestMethod.***POST***, produces =   
 MediaType.***IMAGE\_GIF\_VALUE***)  
 **public** String upload(@RequestPart(**"file"**) MultipartFile file,  
 @RequestParam(**"start"**) **int** start,  
 @RequestParam(**"end"**) **int** end,  
 @RequestParam(**"speed"**) **int** speed,  
 @RequestParam(**"repeat"**) **boolean** repeat) **throws** IOException, FrameGrabber.Exception {  
 File videoFile = **new** File(**location** + **"/"** + System  
 .*currentTimeMillis*() + **".mp4"**);  
 file.transferTo(videoFile);  
  
 ***log***.info(**"Saved video file to {}"**, videoFile.getAbsolutePath());  
  
 Path output = Paths.*get*(**location** + **"/gif/"** + System.*currentTimeMillis*() + **".gif"**);  
  
 FFmpegFrameGrabber frameGrabber = **videoDecoderService**.read(videoFile);  
 AnimatedGifEncoder gifEncoder = **gifEncoderService**.getGifEncoder(repeat,  
 (**float**) frameGrabber.getFrameRate(), output);  
 **converterService**.toAnimatedGif(frameGrabber, gifEncoder, start, end, speed);  
  
 ***log***.info(**"Saved generated gif to {}"**, output.toString());  
  
 **return** output.getFileName().toString();  
 }  
}

…

curl -F file=@cat.mp4 -F start=0 -F speed=1 -F repeated=0 localhost:8080/upload

…

**package** schultz.dustin.io.services;  
  
**import** com.madgag.gif.fmsware.AnimatedGifEncoder;  
**import** org.bytedeco.javacv.FFmpegFrameGrabber;  
**import** org.bytedeco.javacv.FrameGrabber;  
**import** org.bytedeco.javacv.Java2DFrameConverter;  
**import** org.springframework.stereotype.Service;  
  
**import** java.awt.image.BufferedImage;  
  
@Service  
**public class** ConverterService {  
  
 **public void** toAnimatedGif(FFmpegFrameGrabber frameGrabber, AnimatedGifEncoder  
 gifEncoder, **int** start, **int** end, **int** speed) **throws** FrameGrabber.Exception {  
 **long** startFrame = Math.*round*(start \* frameGrabber.getFrameRate());  
 **long** endFrame = Math.*round*(end \* frameGrabber.getFrameRate());  
  
 Java2DFrameConverter frameConverter = **new** Java2DFrameConverter();  
  
 **for** (**long** i = startFrame; i < endFrame; i++) {  
  
 **if** (i % speed == 0) {  
  
 *// Bug if frameNumber is set to 0* **if** (i > 0) {  
 frameGrabber.setFrameNumber((**int**) i);  
 }  
  
 BufferedImage bufferedImage = frameConverter  
 .convert(frameGrabber.grabImage());  
 gifEncoder.addFrame(bufferedImage);  
 }  
  
 }  
  
 frameGrabber.stop();  
 gifEncoder.finish();  
 }  
}

…

**package** schultz.dustin.io.services;  
  
**import** com.madgag.gif.fmsware.AnimatedGifEncoder;  
**import** org.springframework.stereotype.Service;  
  
**import** java.nio.file.Path;  
  
@Service  
**public class** GifEncoderService {  
  
 **public** AnimatedGifEncoder getGifEncoder(**boolean** repeat, **float** frameRate, Path  
 output) {  
 AnimatedGifEncoder gifEncoder = **new** AnimatedGifEncoder();  
  
 **if** (repeat) {  
 gifEncoder.setRepeat(0);  
 }  
  
 gifEncoder.setFrameRate(frameRate);  
 gifEncoder.start(output.toString());  
 **return** gifEncoder;  
 }  
}

…

**package** schultz.dustin.io.services;  
  
**import** org.bytedeco.javacv.FFmpegFrameGrabber;  
**import** org.bytedeco.javacv.FrameGrabber;  
**import** org.springframework.stereotype.Service;  
  
**import** java.io.File;  
  
@Service  
**public class** VideoDecoderService {  
  
 **public** FFmpegFrameGrabber read(File video) **throws** FrameGrabber.Exception {  
 FFmpegFrameGrabber frameGrabber = **new** FFmpegFrameGrabber(video);  
 frameGrabber.start();  
 **return** frameGrabber;  
 }  
}

…

application.properties

# configure auto-configured MultipartConfigElement

multipart.maxFileSize=50MB

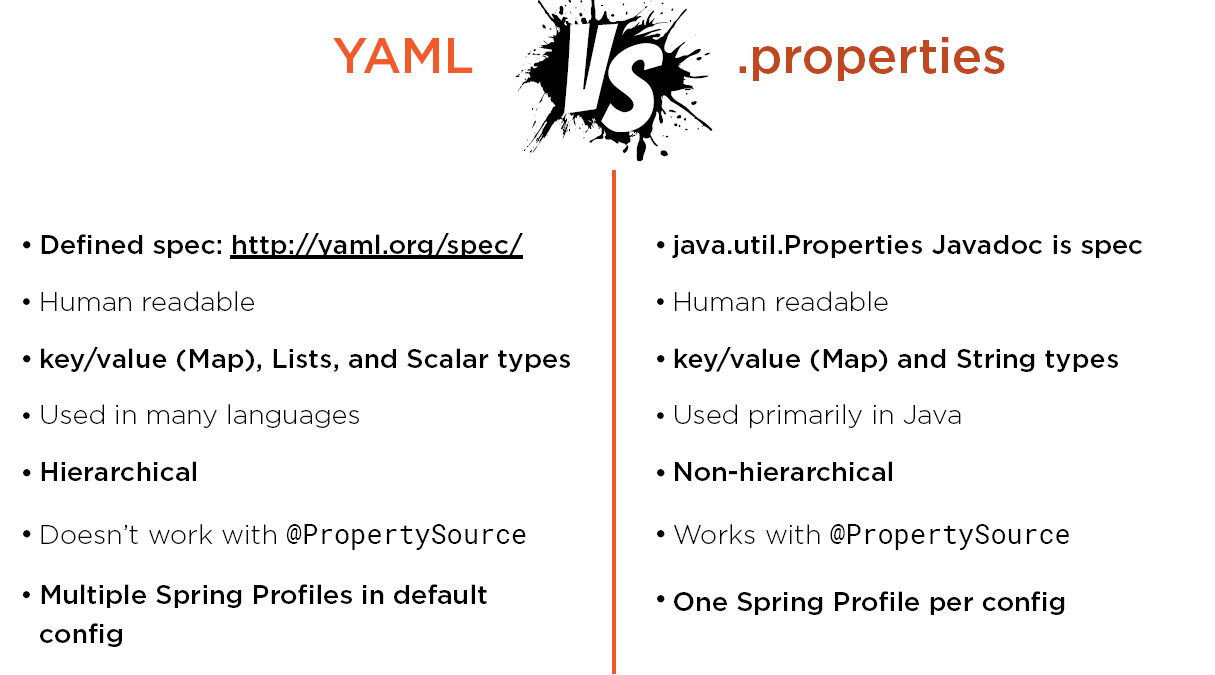
multipart.maxRequestSize=50MB

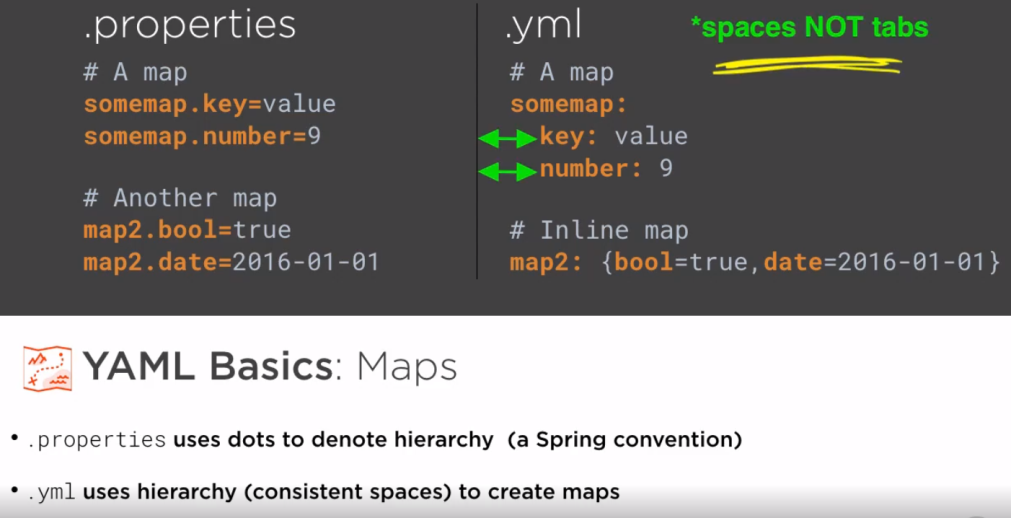
multipart.location=${java.io.tmpdir}

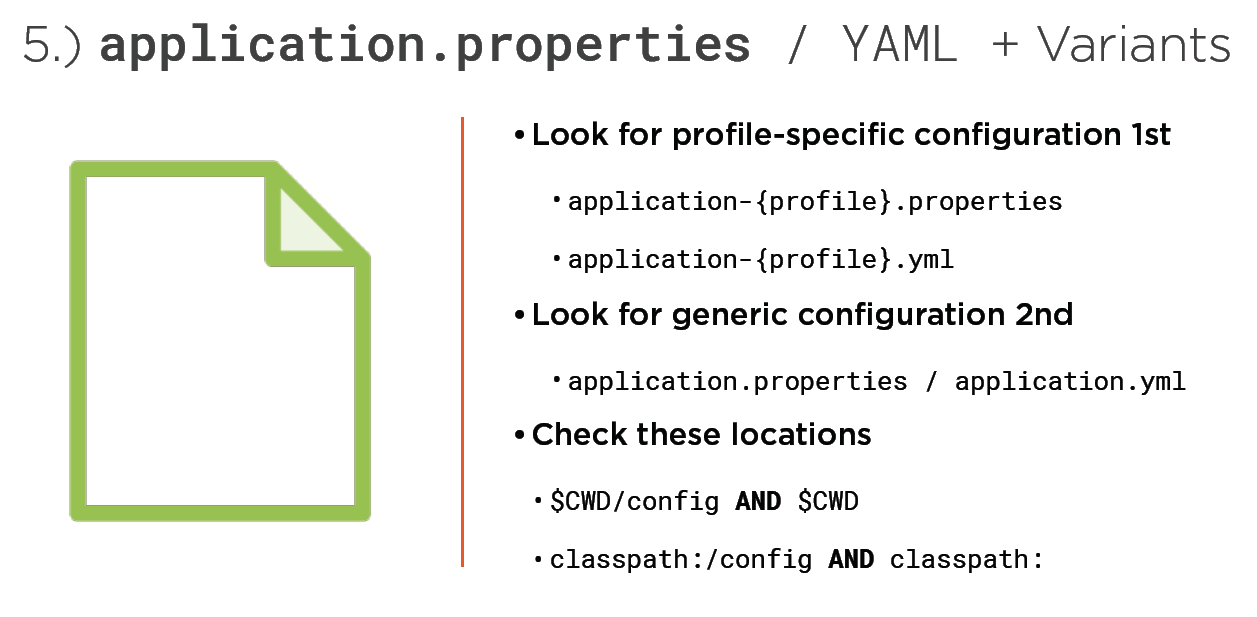
==========================

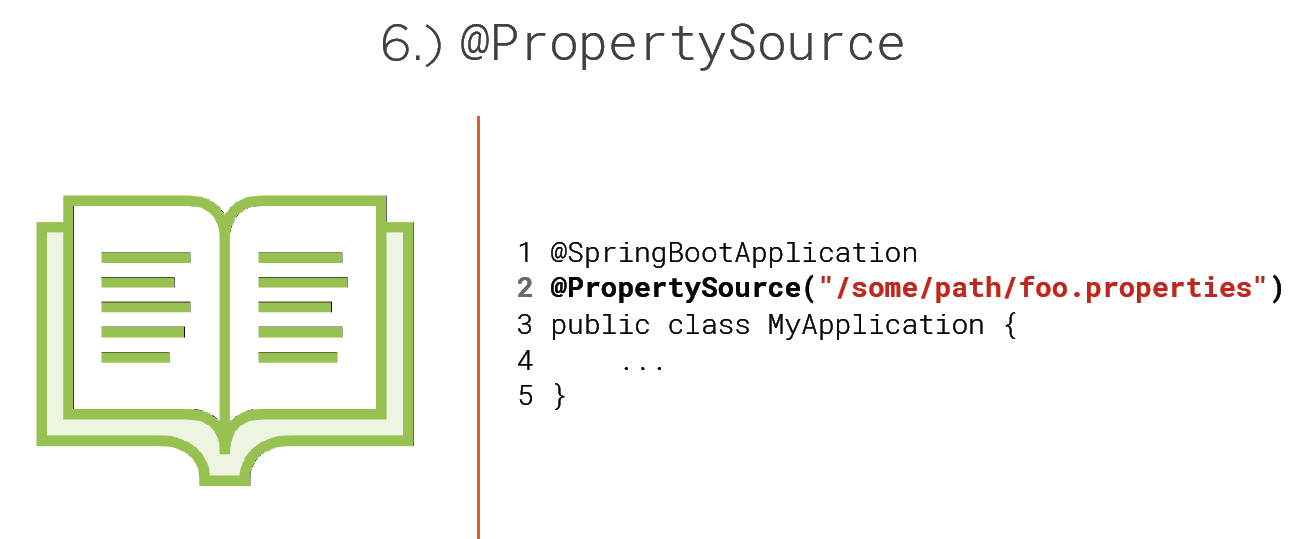
Anado las devtool para refrescar (con optional en true):

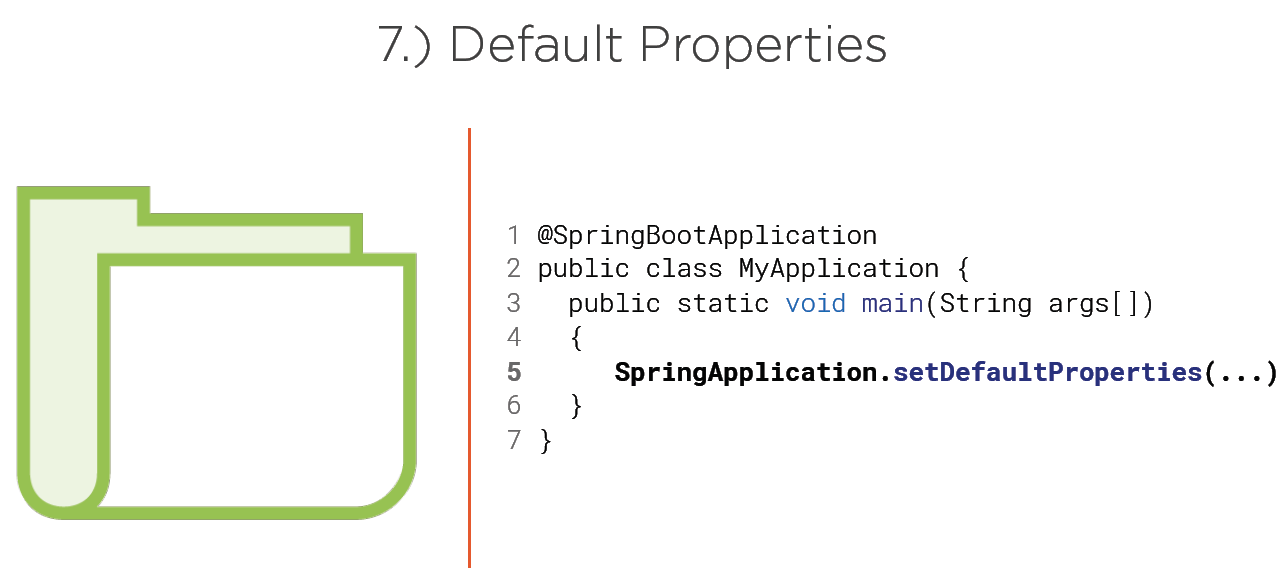
<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-devtools</**artifactId**>  
 <**optional**>true</**optional**>  
</**dependency**>

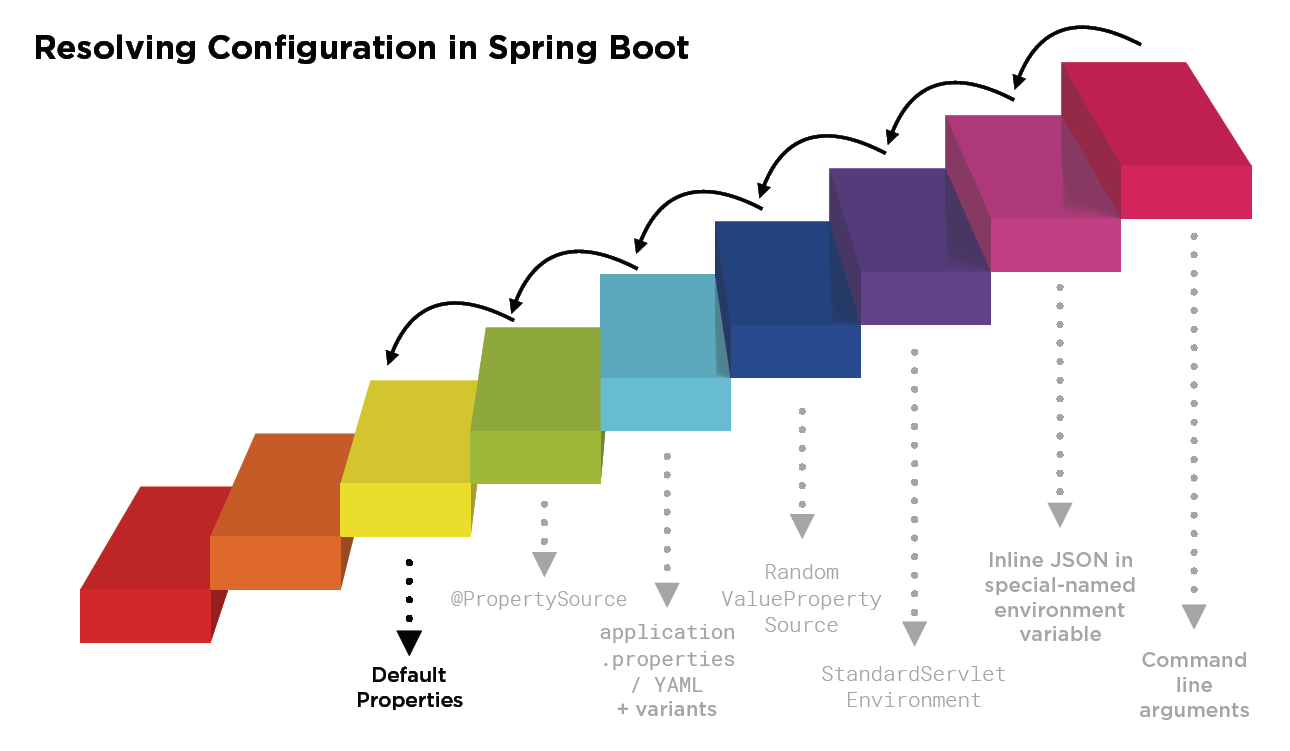












\*\*\* Para meter el actuator tiene que ir security tambien

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-actuator</**artifactId**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-security</**artifactId**>  
</**dependency**>

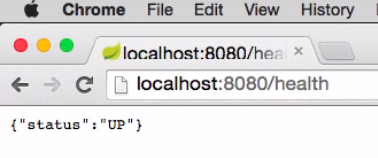
Y creo esta clase en otro paquete

**package** schultz.dustin.io.health;  
  
**import** com.justgifit.JustGifItProperties;  
**import** org.springframework.boot.actuate.health.Health;  
**import** org.springframework.boot.actuate.health.HealthIndicator;  
**import** org.springframework.stereotype.Component;  
  
**import** javax.inject.Inject;  
  
@Component  
**public class** JustGifItHealthIndicator **implements** HealthIndicator {  
  
 @Inject  
 **private** JustGifItProperties **properties**;  
  
 @Override  
 **public** Health health() {  
 **if** (!**properties**.getGifLocation().canWrite()) {  
 **return** Health.*down*().build();  
 }  
  
 **return** Health.*up*().build();  
 }  
}

…

application.yml

*# configure auto-configured MultipartConfigElement***multipart**:  
 **maxFileSize**: 50MB  
 **maxRequestSize**: 50MB  
 **location**: ${java.io.tmpdir}  
  
**spring**:  
 **mvc**:  
 **favicon**:  
 **enabled**: false  
  
**com**:  
 **justgifit**:  
 **optimize**: true  
 **gif-location**: ${java.io.tmpdir}/gif



Y como le he metido “security” entonces le tengo que ejecutar con user y password que la pw me la imprimen en los logs cuando corro la applicacion con CURL

