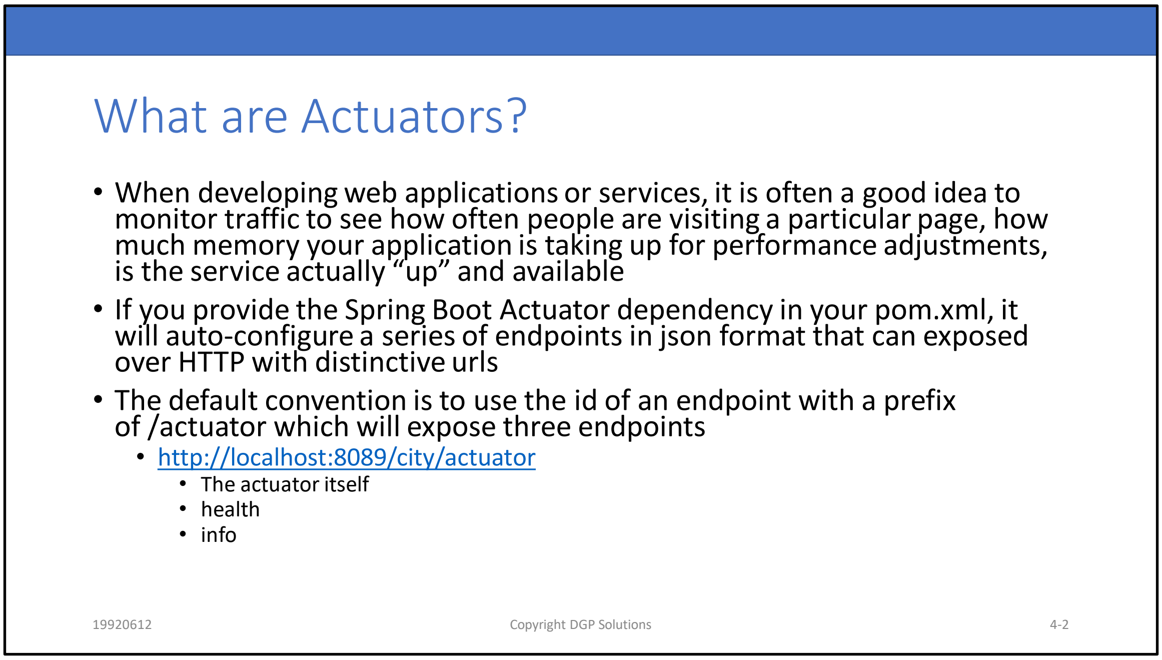
4. Actuators

DGP Solutions

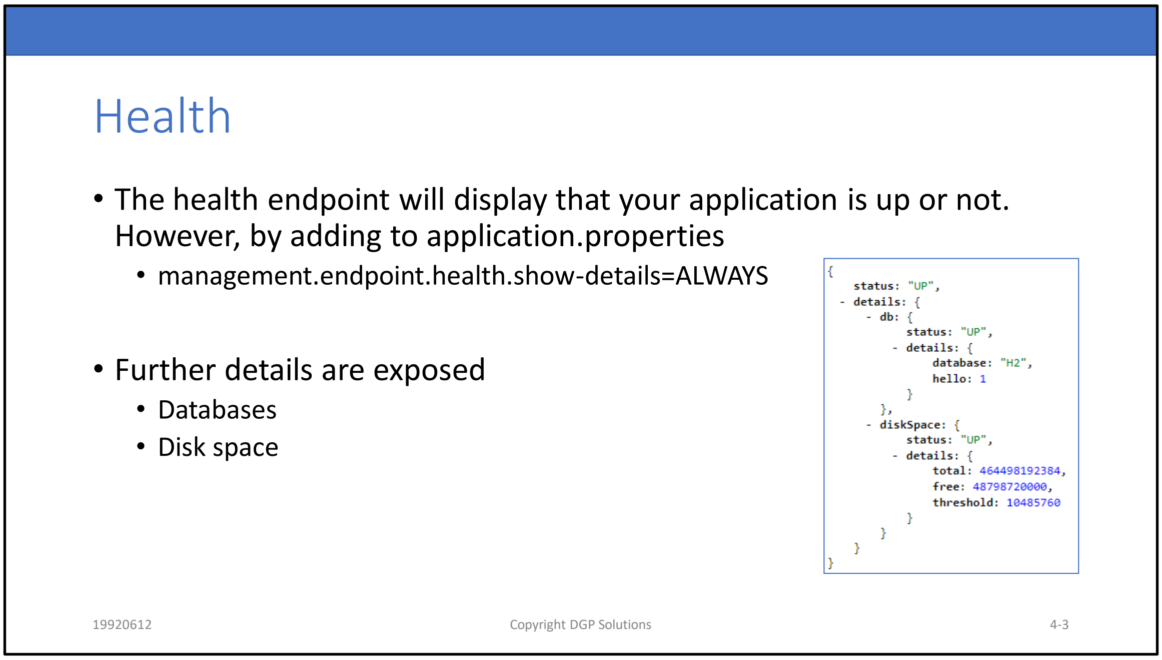


What are Actuators?

- When developing web applications or services, it is often a good idea to monitor traffic to see how often people are visiting a particular page, how much memory your application is taking up for performance adjustments, is the service actually "up" and available
- If you provide the Spring Boot Actuator dependency in your pom.xml, it will auto-configure a series of endpoints in json format that can exposed over HTTP with distinctive urls
- The default convention is to use the id of an endpoint with a prefix of /actuator which will expose three endpoints
  - http://localhost:8089/city/actuator
    - The actuator itself
    - health
    - info

19920612 Copyright DGP Solutions 4-2

- By default, all endpoints are enabled except for shutdown. This can be changed;
  - management.endpoint.info.enabled=false, means that the info endpoint is no longer enabled and not exposed
  - management.endpoint.shutdown.enabled=true, means we enabled shutdown
- However, also by default, only health and info are exposed. To change this, in application properties you can selectively expose enabled endpoints over HTTP
  - management.endpoints.web.exposure.include=beans,health, means that apart from the actuator endpoint, the only other endpoints exposed over HTTP are beans and health
- Or expose ALL the enabled endpoints and "hide" others
  - management.endpoints.web.exposure.include=*
  - management.endpoints.web.exposure.exclude=beans

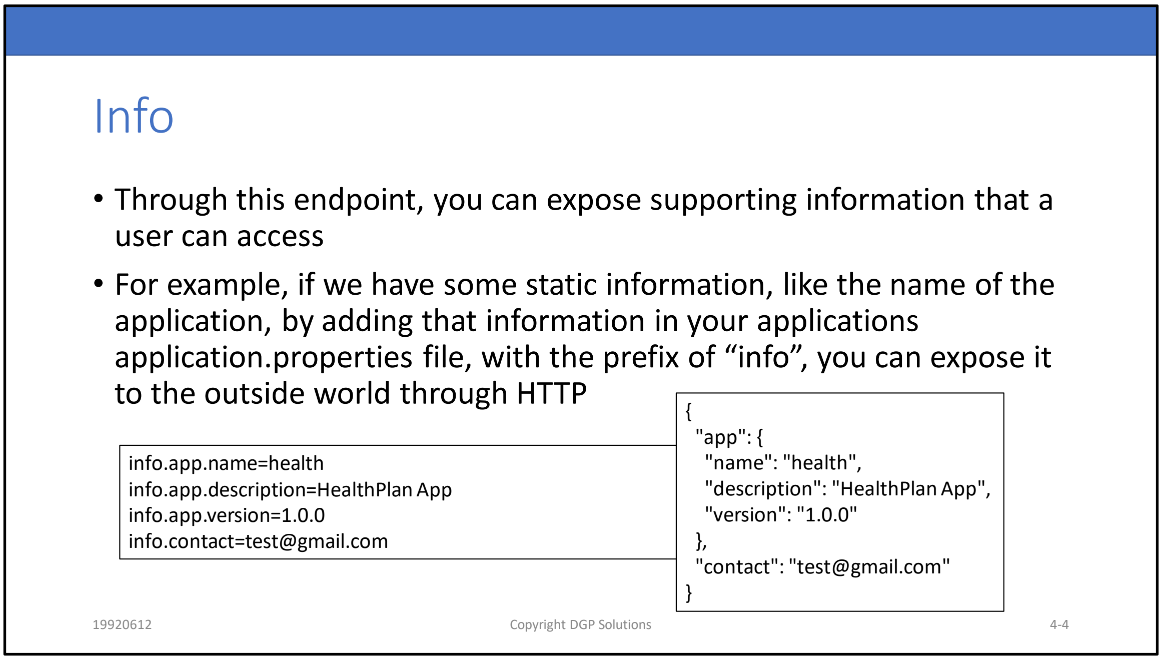


Health

- The health endpoint will display that your application is up or not. However, by adding to application.properties
  - management.endpoint.health.show-details=ALWAYS
- Further details are exposed
  - Databases
  - Disk space

```
{
    status: "UP",
  - details: {
      - db: {
            status: "UP",
          - details: {
                database: "H2",
                hello: 1
            }
        },
      - diskSpace: {
            status: "UP",
          - details: {
                total: 464498192384,
                free: 48798720000,
                threshold: 10485760
            }
        }
    }
}
```

19920612 Copyright DGP Solutions 4-3



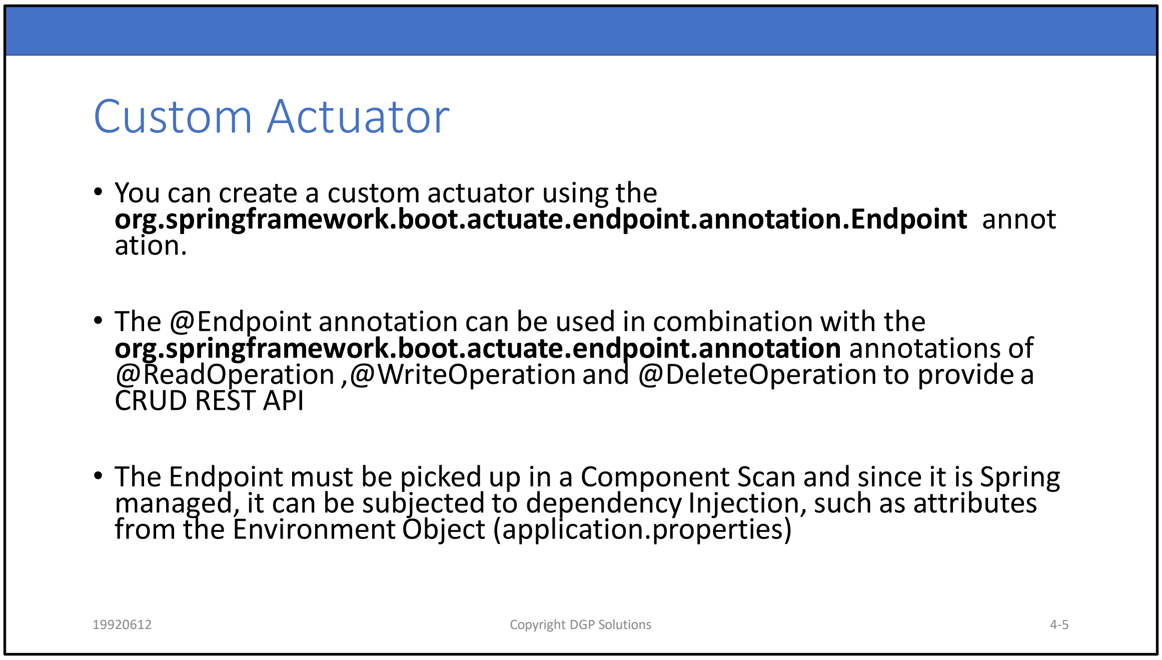
Info

- Through this endpoint, you can expose supporting information that a user can access
- For example, if we have some static information, like the name of the application, by adding that information in your applications application.properties file, with the prefix of “info”, you can expose it to the outside world through HTTP

```
info.app.name=health
info.app.description=HealthPlan App
info.app.version=1.0.0
info.contact=test@gmail.com
```

```
{
 "app": {
  "name": "health",
  "description": "HealthPlan App",
  "version": "1.0.0"
 },
 "contact": "test@gmail.com"
}
```

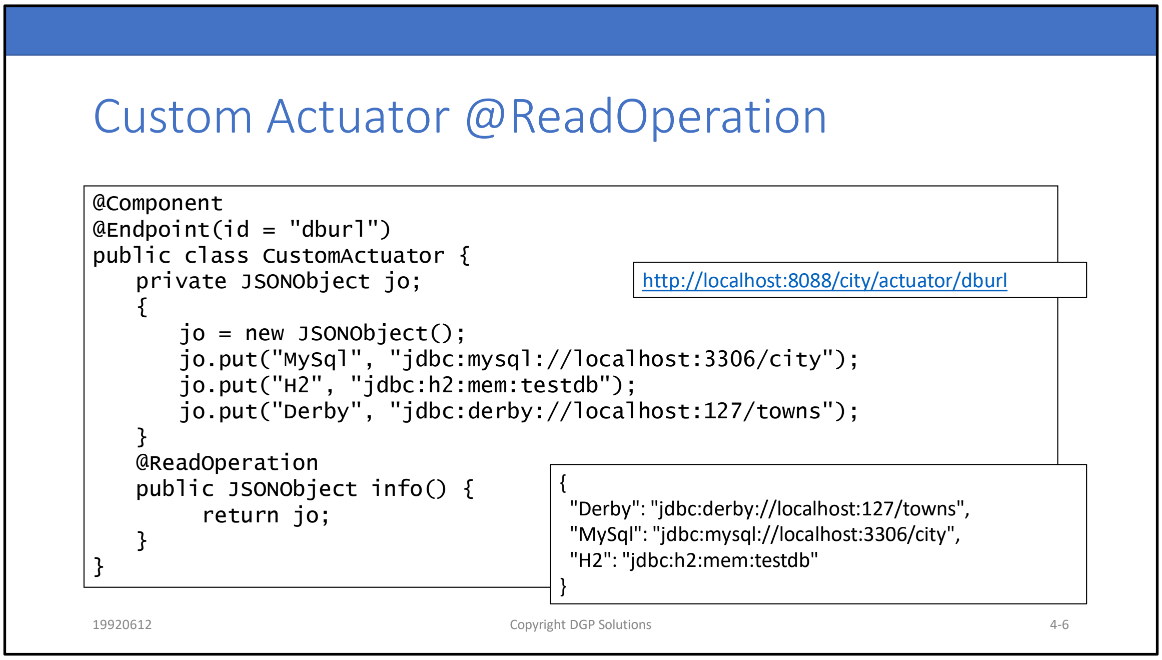
19920612 Copyright DGP Solutions 4-4



Custom Actuator

- You can create a custom actuator using the **org.springframework.boot.actuate.endpoint.annotation.Endpoint** annot ation.
- The @Endpoint annotation can be used in combination with the **org.springframework.boot.actuate.endpoint.annotation** annotations of @ReadOperation ,@WriteOperation and @DeleteOperation to provide a CRUD REST API
- The Endpoint must be picked up in a Component Scan and since it is Spring managed, it can be subjected to dependency Injection, such as attributes from the Environment Object (application.properties)

19920612 Copyright DGP Solutions 4-5



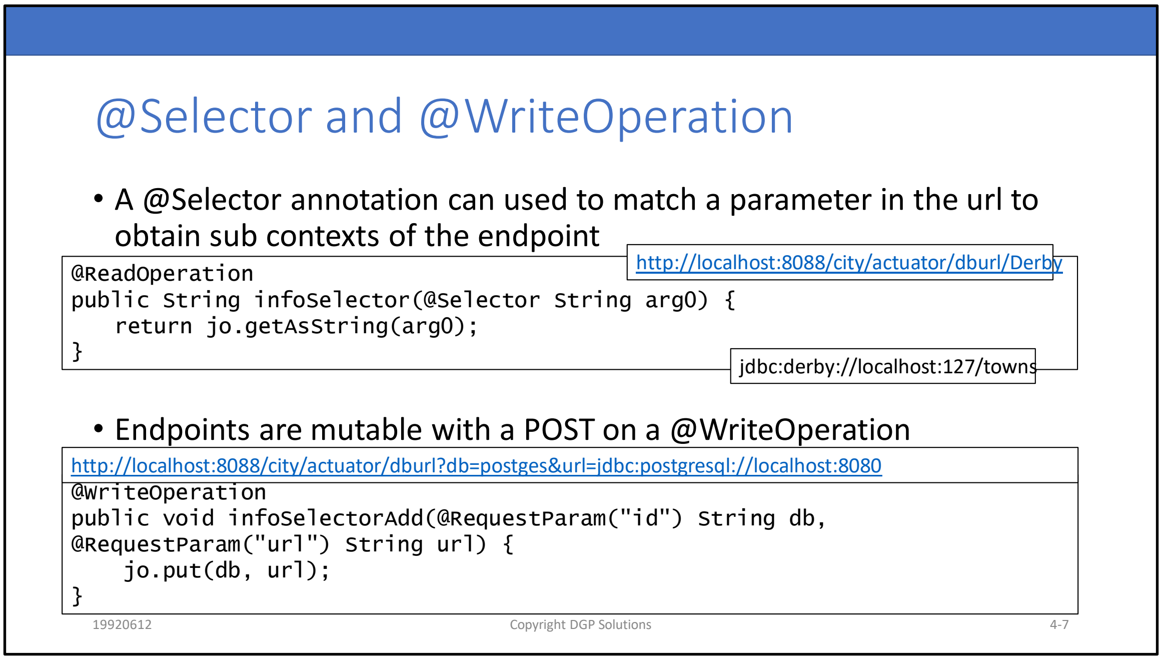
Custom Actuator @ReadOperation

```
@Component
@Endpoint(id = "dburl")
public class CustomActuator {
    private JSONObject jo;
    {
        jo = new JSONObject();
        jo.put("MySql", "jdbc:mysql://localhost:3306/city");
        jo.put("H2", "jdbc:h2:mem:testdb");
        jo.put("Derby", "jdbc:derby://localhost:127/towns");
    }
    @ReadOperation
    public JSONObject info() {
          return jo;
    }
}
```

http://localhost:8088/city/actuator/dburl

```
{
  "Derby": "jdbc:derby://localhost:127/towns",
  "MySql": "jdbc:mysql://localhost:3306/city",
  "H2": "jdbc:h2:mem:testdb"
}
```

19920612 Copyright DGP Solutions 4-6



@Selector and @WriteOperation

- A @Selector annotation can used to match a parameter in the url to obtain sub contexts of the endpoint

http://localhost:8088/city/actuator/dburl/Derby

```
@ReadOperation
public String infoSelector(@Selector String arg0) {
   return jo.getAsString(arg0);
}
```

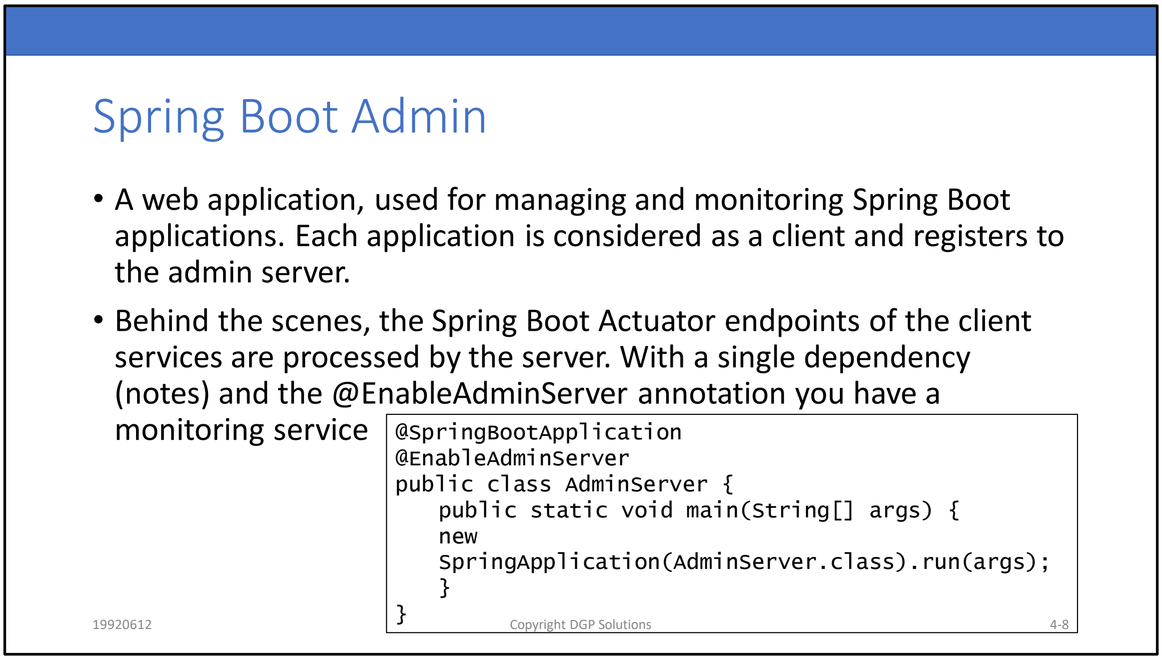
jdbc:derby://localhost:127/towns

- Endpoints are mutable with a POST on a @WriteOperation

http://localhost:8088/city/actuator/dburl?db=postges&url=jdbc:postgresql://localhost:8080

```
@WriteOperation
public void infoSelectorAdd(@RequestParam("id") String db,
@RequestParam("url") String url) {
    jo.put(db, url);
}
```

19920612 Copyright DGP Solutions 4-7



Spring Boot Admin

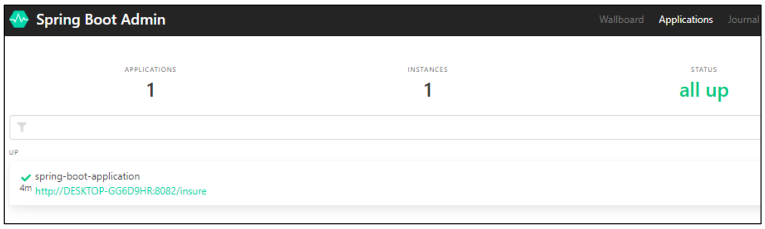
- A web application, used for managing and monitoring Spring Boot applications. Each application is considered as a client and registers to the admin server.
- Behind the scenes, the Spring Boot Actuator endpoints of the client services are processed by the server. With a single dependency (notes) and the @EnableAdminServer annotation you have a monitoring service

```
@SpringBootApplication
@EnableAdminServer
public class AdminServer {
    public static void main(String[] args) {
    new
    SpringApplication(AdminServer.class).run(args);
    }
}
```

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

The Client Service

- The client application must register with the running monitoring service. If Spring Boot Admin was running on port 9082, then a single entry in the client service application properties file and one maven dependency will suffice
  - **spring.boot.admin.client.url=http://localhost:9082**



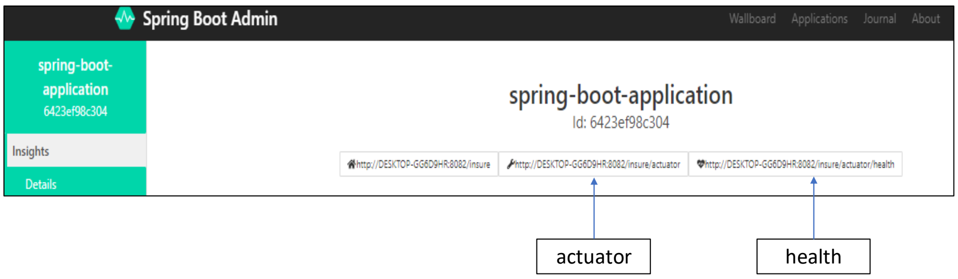
APPLICATIONS 1
INSTANCES 1
STATUS all up

UP
✓ spring-boot-application
4m http://DESKTOP-GG6D9HR:8082/insure

```
<dependency>
<groupId>de.codecentric</groupId>
<artifactId>spring-boot-admin-starter-client</artifactId>
<version>2.2.0</version>
</dependency>
```

Display Service Actuators

- Upon selecting the “tick” icon next to your application, actuator display information from the client application actuators

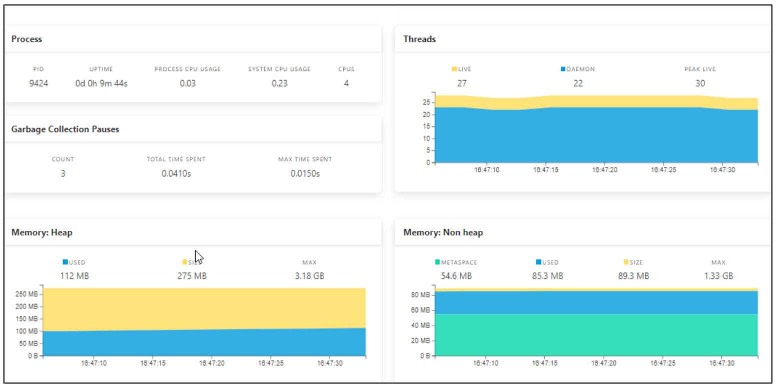


spring-boot-application
Id: 6423ef98c304

http://DESKTOP-GG6D9HR:8082/insure
http://DESKTOP-GG6D9HR:8082/insure/actuator
http://DESKTOP-GG6D9HR:8082/insure/actuator/health

Memory and Threading

- Upon scrolling down we see a dashboard of thread and memory information



Process

PID	UPTIME	PROCESS CPU USAGE	SYSTEM CPU USAGE	CPUS
9424	0d 0h 9m 44s	0.03	0.23	4

Garbage Collection Pauses

COUNT	TOTAL TIME SPENT	MAX TIME SPENT
3	0.0410s	0.0150s

Threads

🟨 LIVE	🟦 DAEMON	PEAK LIVE
27	22	30

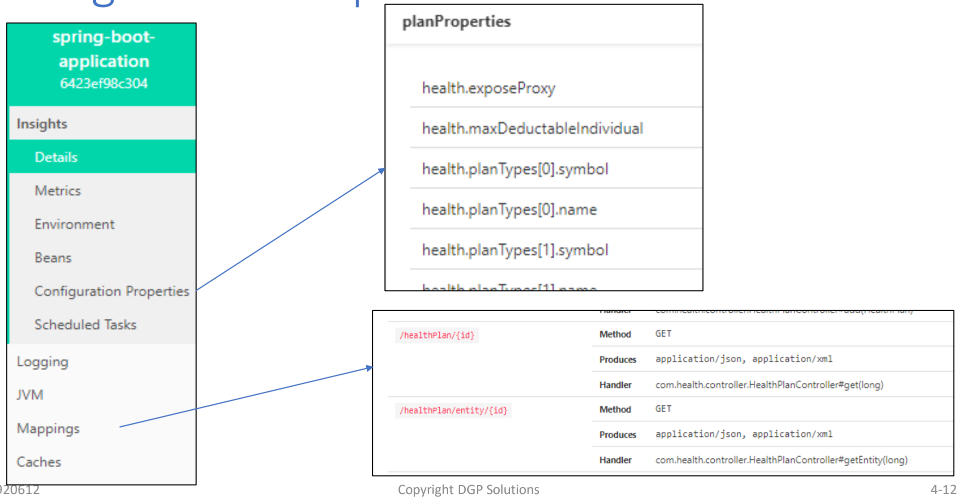
Memory: Heap

🟦 USED	🟨 SI	MAX
112 MB	275 MB	3.18 GB

Memory: Non heap

🟩 METASPACE	🟦 USED	🟨 SIZE	MAX
54.6 MB	85.3 MB	89.3 MB	1.33 GB

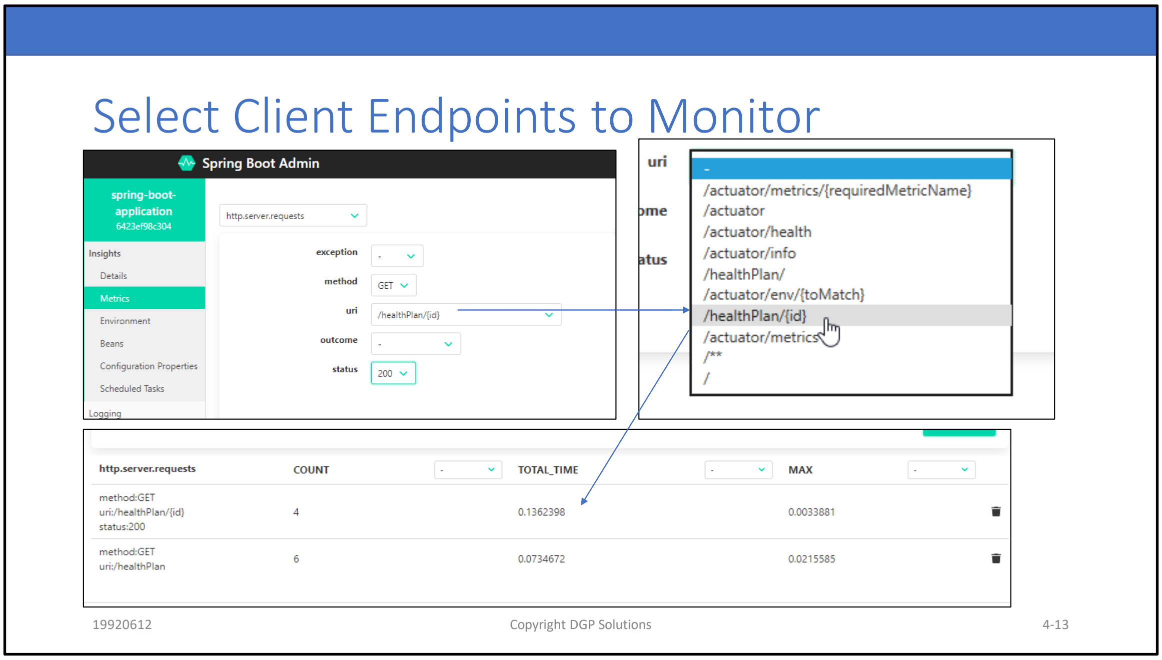
Configuration Properties



planProperties

health.exposeProxy
health.maxDeductableIndividual
health.planTypes[0].symbol
health.planTypes[0].name
health.planTypes[1].symbol

/healthPlan/{id}	Method	GET
	Produces	application/json, application/xml
	Handler	com.health.controller.HealthPlanController#get(long)
/healthPlan/entity/{id}	Method	GET
	Produces	application/json, application/xml
	Handler	com.health.controller.HealthPlanController#getEntity(long)



Select Client Endpoints to Monitor

http.server.requests

exception: -
method: GET
uri: /healthPlan/{id}
outcome: -
status: 200

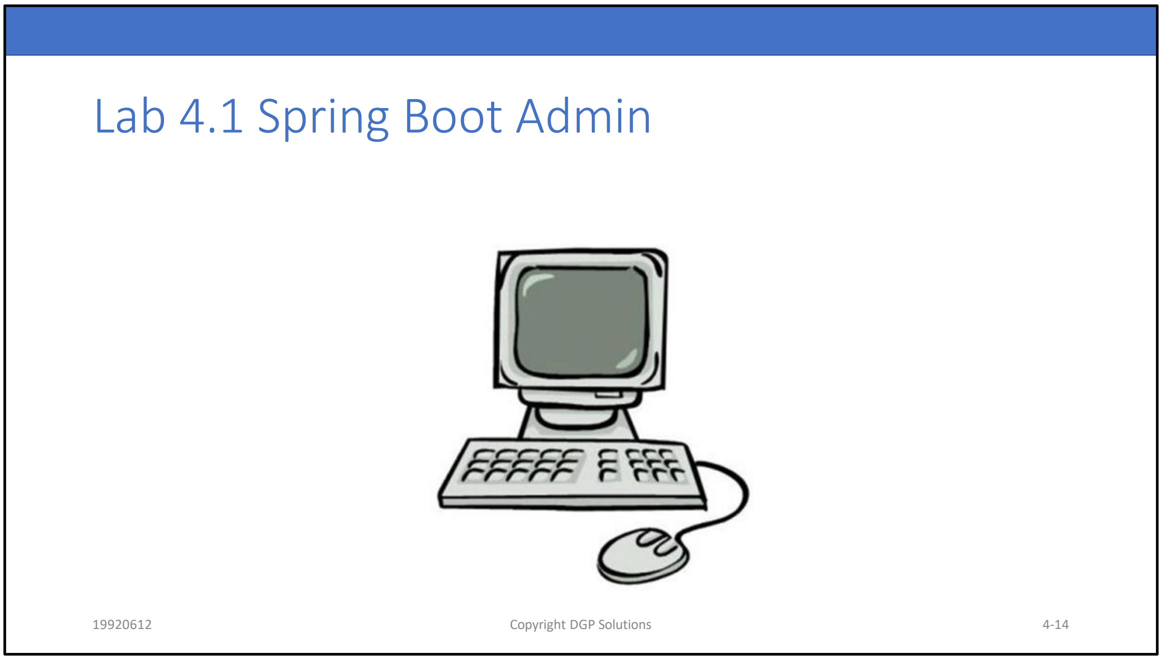
uri

- -
- /actuator/metrics/{requiredMetricName}
- /actuator
- /actuator/health
- /actuator/info
- /healthPlan/
- /actuator/env/{toMatch}
- /healthPlan/{id}
- /actuator/metrics
- /**
- /

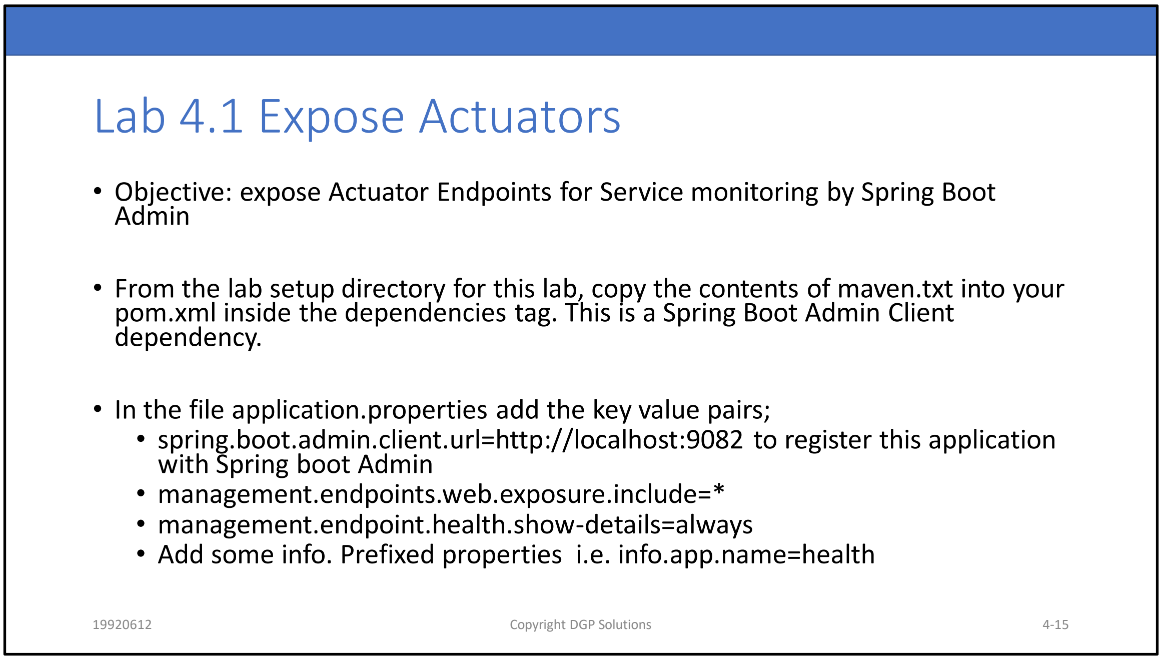
http.server.requests	COUNT	TOTAL_TIME	MAX
method:GET uri:/healthPlan/{id} status:200	4	0.1362398	0.0033881
method:GET uri:/healthPlan	6	0.0734672	0.0215585

19920612 Copyright DGP Solutions 4-13

- Their maybe some latency between running you client app and the endpoint being registered. Run your services a few times and your url will appear in the drop down menu of uri's

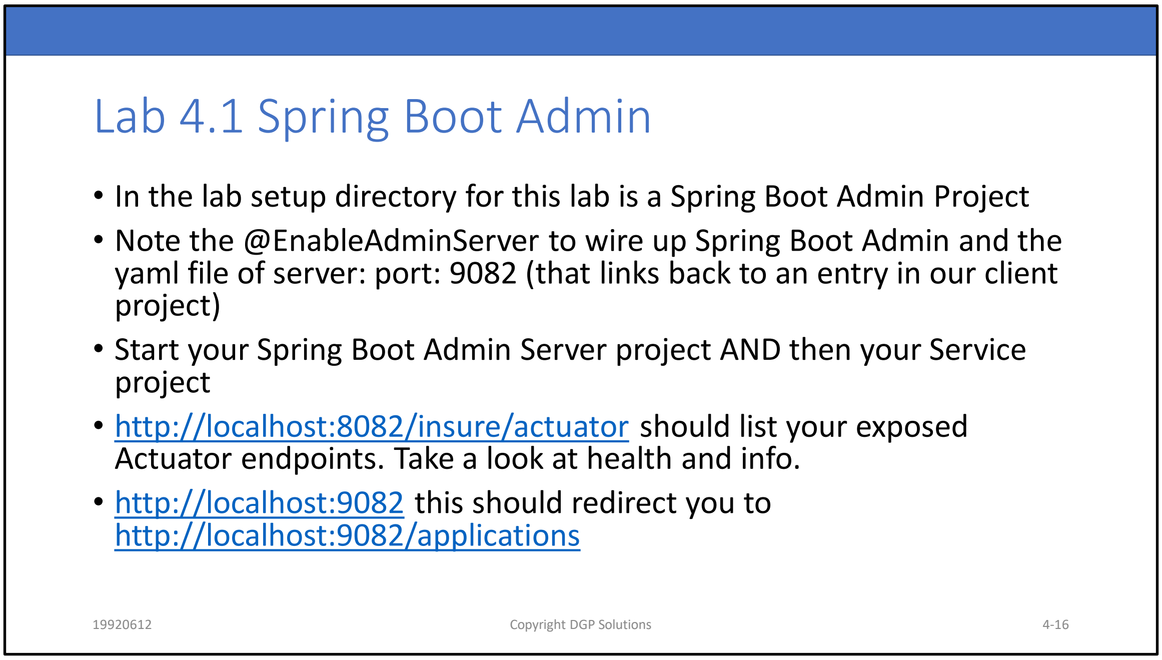


Lab 4.1 Spring Boot Admin



Lab 4.1 Expose Actuators

- Objective: expose Actuator Endpoints for Service monitoring by Spring Boot Admin
- From the lab setup directory for this lab, copy the contents of maven.txt into your pom.xml inside the dependencies tag. This is a Spring Boot Admin Client dependency.
- In the file application.properties add the key value pairs;
  - spring.boot.admin.client.url=http://localhost:9082 to register this application with Spring boot Admin
  - management.endpoints.web.exposure.include=*
  - management.endpoint.health.show-details=always
  - Add some info. Prefixed properties i.e. info.app.name=health

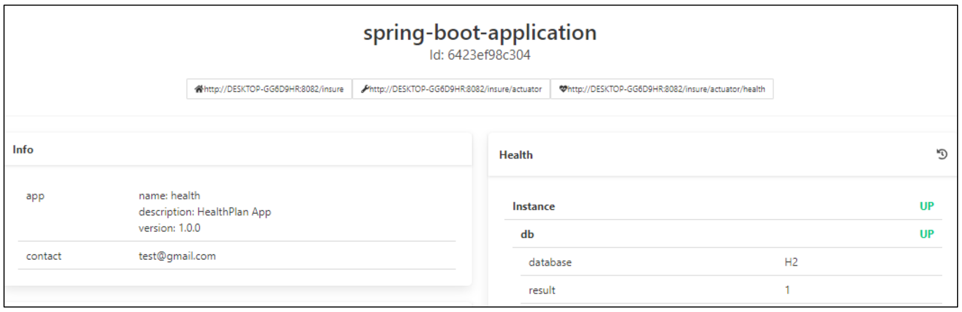


Lab 4.1 Spring Boot Admin

- In the lab setup directory for this lab is a Spring Boot Admin Project
- Note the @EnableAdminServer to wire up Spring Boot Admin and the yaml file of server: port: 9082 (that links back to an entry in our client project)
- Start your Spring Boot Admin Server project AND then your Service project
- http://localhost:8082/insure/actuator should list your exposed Actuator endpoints. Take a look at health and info.
- http://localhost:9082 this should redirect you to http://localhost:9082/applications

Lab 4.1 Monitor Requests

- Click on the tick mark of your application (only one listed). You will see your info and health actuator endpoint listed n



spring-boot-application
Id: 6423ef98c304

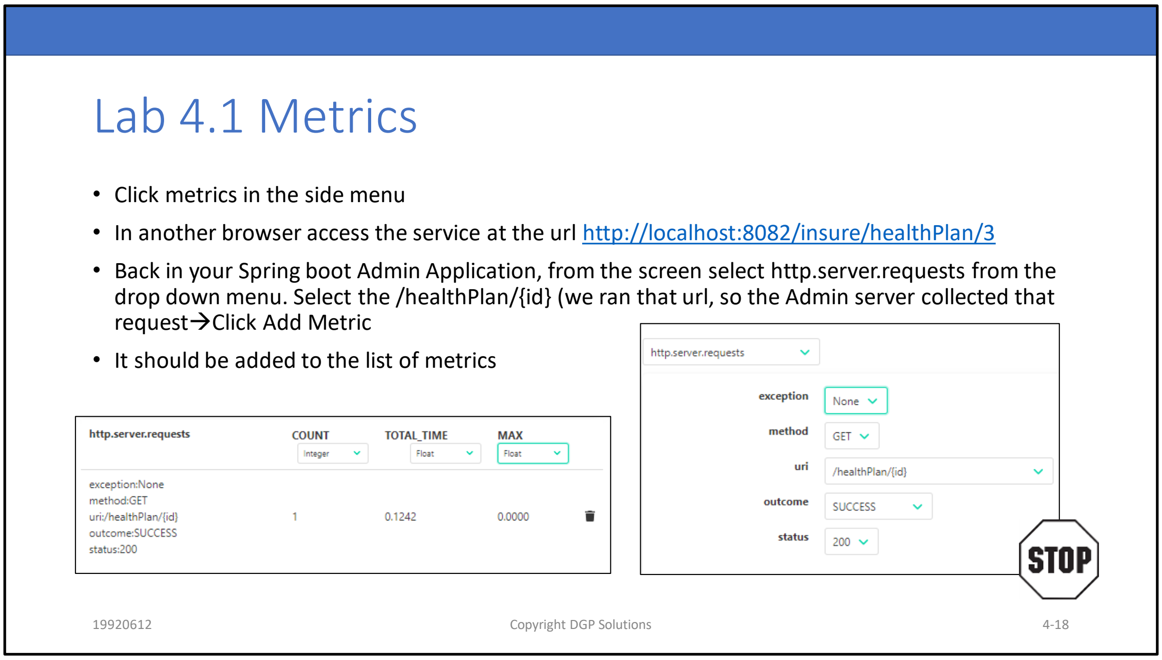
http://DESKTOP-GG6D9HR:8082/insure | http://DESKTOP-GG6D9HR:8082/insure/actuator | http://DESKTOP-GG6D9HR:8082/insure/actuator/health

Info

app	name: health description: HealthPlan App version: 1.0.0
contact	test@gmail.com

Health

Instance		UP
db		UP
database	H2	
result	1	



Lab 4.1 Metrics

- Click metrics in the side menu
- In another browser access the service at the url http://localhost:8082/insure/healthPlan/3
- Back in your Spring boot Admin Application, from the screen select http.server.requests from the drop down menu. Select the /healthPlan/{id} (we ran that url, so the Admin server collected that request→Click Add Metric
- It should be added to the list of metrics

http.server.requests	COUNT	TOTAL_TIME	MAX
	Integer	Float	Float
exception:None method:GET uri:/healthPlan/{id} outcome:SUCCESS status:200	1	0.1242	0.0000

http.server.requests

exception	None
method	GET
uri	/healthPlan/{id}
outcome	SUCCESS
status	200

STOP

19920612 Copyright DGP Solutions 4-18

- As you repeatedly access http://localhost:8082/insure/healthPlan/3, the metrics change for that request

- Add another metric for http://localhost:8082/insure/healthPlan. Make sure you RUN the service at least once before setting up the metric