Spring Boot — Why you should always use the RestTemplateBuilder to create a RestTemplate instance?

* step-1-rest-template-instance
* step-2-rest-template-instance-as-bean
* step-3-rest-template-using-rest-template-builder
* You are a Spring Boot developer…
* You are using [Micrometer.io](https://micrometer.io/) for your application metrics…
* You are using [Spring Cloud Sleuth](https://spring.io/projects/spring-cloud-sleuth) for distributed tracing…
* You are using Spring’s [RestTemplate](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html" \t "_blank) to do REST API calls (between your services), and you create a RestTemplate instance yourself?

**Read on and learn why you should always use the RestTemplateBuilder instead of creating a RestTemplate instance yourself!**

**Example project**

Before we start, let me explain the example project I use in this blog post. You can find the complete codebase on [Github](https://github.com/stockgeeks/spring-boot-rest-template-build-example" \t "_blank). Clone the Git repo and run the examples step by step while reading this blog post.

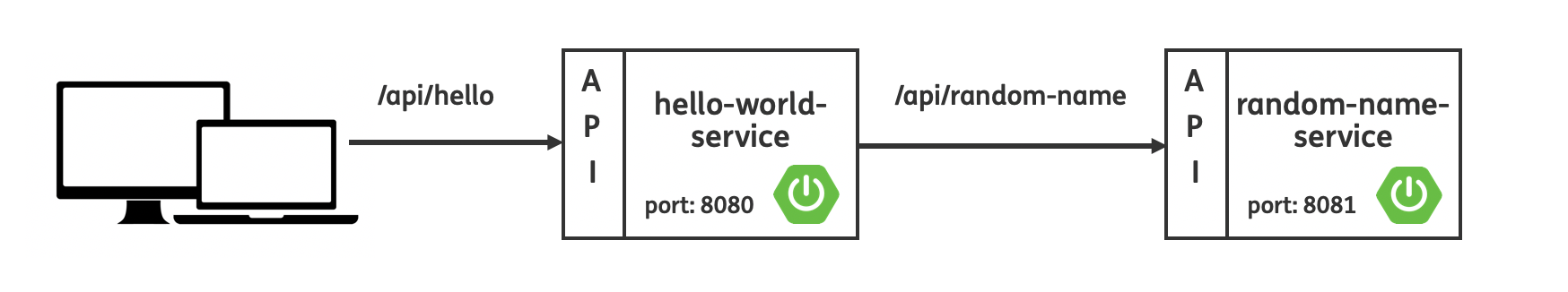
The example project contains two straightforward Spring Boot 2 applications:

**Hello World service:**

* Exposes a Hello World greeting Rest API endpoint/api/hello
* Calls the Random name service endpoint using a RestTemplate
* Appends a random name to the greeting. An example response to the client looks like: “Hello World: Josh Long”.
* Runs on port 8080
* Application in the codebase: hello-world-service

**Random name service:**

* Exposes a Rest API endpoint that returns a random name /api/random-name
* Is a downstream service that is called by the Hello World Service
* Runs on port 8081
* Application in the codebase: random-name-service



The ‘Hello World service’ will call the downstream ‘Random name service’ using a RestTemplate.

[*Spring.io Guide — Consuming a RESTful Web Service*](https://spring.io/guides/gs/consuming-rest/)*: Spring provides you with a convenient template class called*[RestTemplate](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html" \t "_blank)*. RestTemplate makes interacting with most RESTful services a one-line incantation. And it can even bind that data to custom domain types.*

In the HelloWorldController from the hello-world-service (see branch: [step-1-rest-template-instance](https://github.com/stockgeeks/spring-boot-rest-template-build-example/tree/step-1-rest-template-instance)), I created a RestTemplate instance myself. Using this RestTemplate the controller will call the random-name-service:

Created a RestTemplate instance myself in the controller.

Check out branch: [step-1-rest-template-instance](https://github.com/stockgeeks/spring-boot-rest-template-build-example/tree/step-1-rest-template-instance) and compile the project:

./mvnw clean package

Start the hello-world-service:

./mvnw spring-boot:run -pl hello-world-service

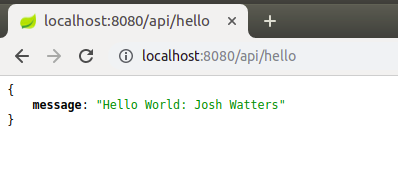
Start the random-name-service:

./mvnw spring-boot:run -pl random-name-service

Call the Hello World endpoint from the browser:

<http://localhost:8080/api/hello>

https://miro.medium.com/max/60/1*NqyhKd2ozB3y2nTaNa_B8g.png?q=20



Call the hello endpoint on the hello-world-service from the browser.

or your favorite command-line tool (like curl):

curl <http://localhost:8080/api/hello>  
{"message":"Hello World: Josh Watters"}

# Everything works fine right?

Yes and no. The hello-world-service is calling the random-name-service and returning the Hello World greeting plus the random name back to the client. Let’s deploy to production, right? Wait… Let take a closer look at:

* RestTemplate metrics
* Distributed tracing

**A closer look — RestTemplate Metrics**

Micrometer.io is the metrics collection facility included in Spring Boot 2’s Actuator. Micrometer.io manages the instrumentation of your RestTemplate and supports metrics out of the box, and that is available with the name: http.client.requests. The metrics are exposed by the Spring Boot’s actuator metrics endpoint:

<http://localhost:8080/actuator/metrics/http.client.requests>

But when we try to access the metrics, we will see the metrics are not available:

curl <http://localhost:8080/actuator/metrics/http.client.requests>  
{"timestamp":"2019-07-08T19:51:07.016+0000","status":404,"error":"Not Found","message":"No message available","path":"/actuator/http.client.requests"}

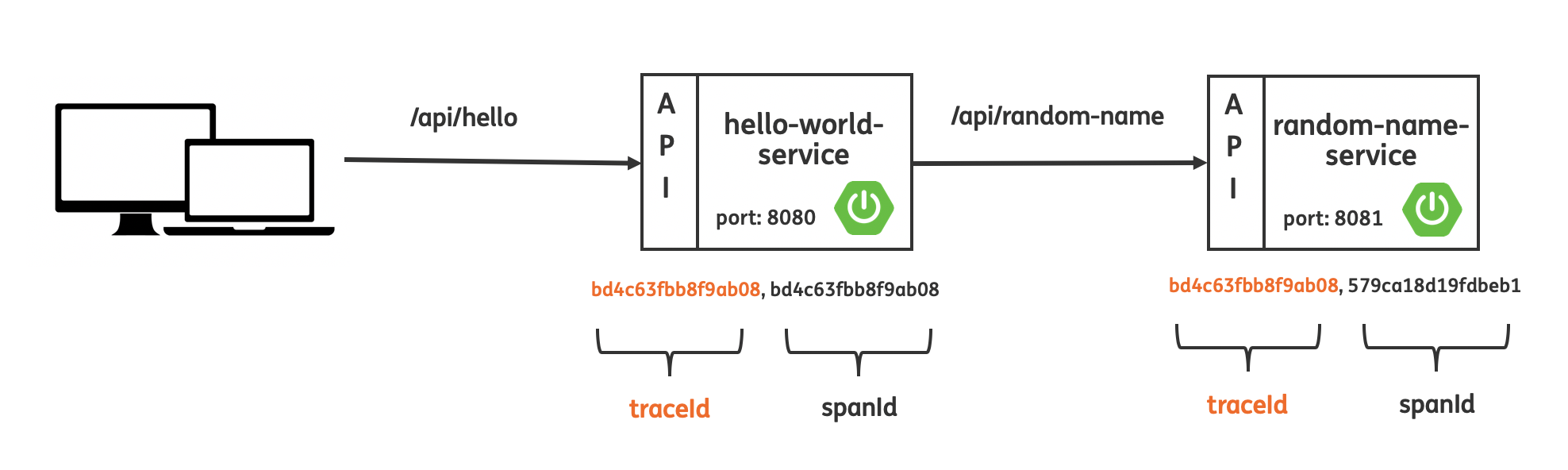
That’s not what we want! Why are my RestTemplate metrics broken 🤔?

Since I created the RestTemplate instance myself, Micrometer.io is not able to instrument the RestTemplate. A side effect is no RestTemplate metrics are exposed. Before I explain how to fix this issue, let’s take a look at what else is broken.

**A closer look — Distributed tracing**

In a distributed system, many services (which could even be deployed on different hosts) can be involved in creating a response to a single request initiated by a client. To be able to debug and trace the path of such a request through all involved services, we need distributed tracing.

In my example, two services are involved in creating the Hello World greeting back to the client.



The expected flow of API calls, including proper traceIds and spanIds.

Let’s see step by step what happens:

* A client requests /api/hello on the hello-world-service
* The incoming request has no trace information attached
* Spring Cloud Sleuth will generate a random traceId and spanId. Please note by default any application flow will start with the same traceId and spanId!
* The hello-world-service calls out to /api/random-name on the downstream random-name-service
* a new spanId is created as a child of the former span. It is identified by the same trace id, a new spanId, and the parent id is set to the spanId of the previous span.

[*Spring Cloud Sleuth*](https://spring.io/projects/spring-cloud-sleuth)*implements a distributed tracing solution for Spring Boot application. For most users Sleuth should be invisible, and all your interactions with external systems should be instrumented automatically. You can capture data simply in logs, or by sending it to a remote collector service.*

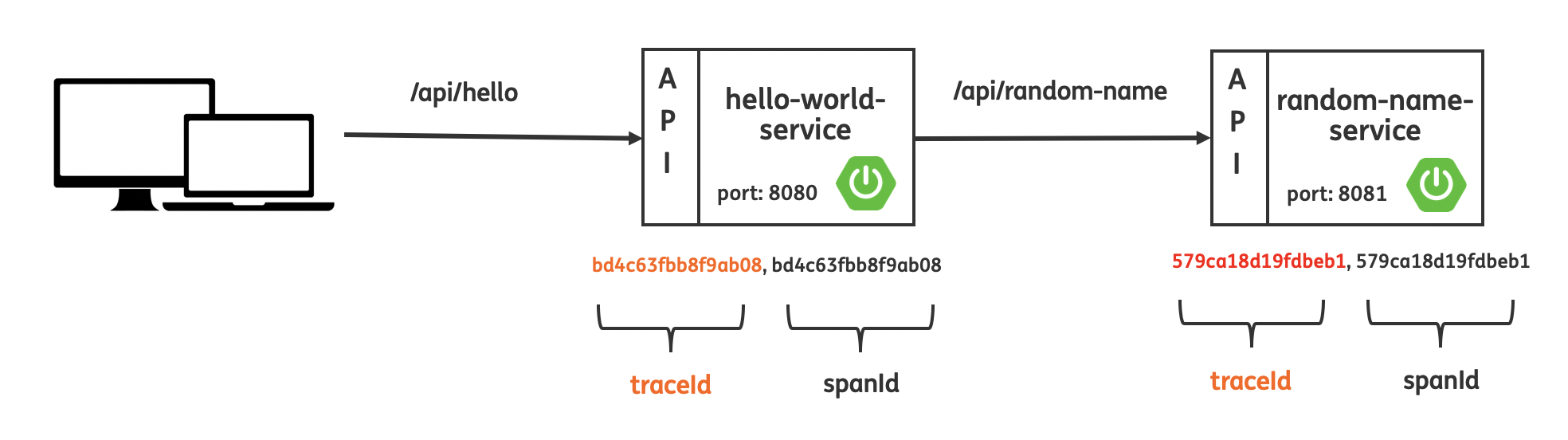
Let’s now take a close look at the log lines of our hello-world-service. We can see both traceId and spanId for the incoming call /api/hello.

INFO [hello-world-service,**bd4c63fbb8f9ab08,bd4c63fbb8f9ab08**,false] 10525 --- [nio-8080-exec-1] i.s.h.world.api.HelloWorldController : Received call on /api/hello. And will call /api/random-name on the random-name-service!

Now let’s switch to the log of the random-name-service.  
We clearly see a new traceId and spanId are created for the incoming /api-random-name call on the random-name-service.

INFO [random-name-service,**579ca18d19fdbeb1,579ca18d19fdbeb1**,false] 10491 --- [nio-8081-exec-2] i.s.r.name.api.RandomNameController : Received call on /api/random-name

This is not what we expected. So also distributed tracing is broken! But why 🤔? The same problem occurs here because Spring Cloud Sleuth is not able to instrument automatically our RestTemplate instance we created in the controller.



Distributed is tracing broken! New traceId and spanId create for the call on the random-name-service.

This is bad! We have to fix the issues before we deploy to production.

**Let’s fix it! Attempt 1**

Because of creating the RestTemplate instance myself in the controller caused the issues of the broken metrics and distributed tracing.

Branch: [step-2-rest-template-instance-as-bean](https://github.com/stockgeeks/spring-boot-rest-template-build-example/tree/step-2-rest-template-instance-as-bean)

Let’s make the RestTemplate a Spring Bean in the RestTemplate configuration.

and autowire the RestTemplate bean in the HelloWorldController:

Autowire the RestTemplate in the controller using constructor injection.

Now rebuild and restart the hello-world-service.

./mvnw spring-boot:run -pl hello-world-service

And do a call to the hello endpoint again:

curl <http://localhost:8080/api/hello>

Let’s take a look at the log files of both applications again. We now see the proper traceIds and spanIds from the log files.

hello-world-service logs:

INFO [hello-world-service,**fc1d5e47723fd38d**,**fc1d5e47723fd38d**,false] 24555 --- [nio-8080-exec-6] i.s.h.world.api.HelloWorldController : Received call on /api/hello. And will call /api/random-name on the random-name-service!

random-name-service logs:

INFO [random-name-service,**fc1d5e47723fd38d**,**825d8abf6d385c02**,false] 24148 --- [nio-8081-exec-3] i.s.r.name.api.RandomNameController : Received call on /api/random-name

So distributed tracing is working again 😀! But unfortunately, the RestTemplate metrics are still not working 😤.

curl <http://localhost:8080/actuator/metrics/http.client.requests>  
{"timestamp":"2019-07-08T20:40:53.621+0000","status":404,"error":"Not Found","message":"No message available","path":"/actuator/http.client.requests"}

Why is distributed tracing working now but the are metrics not? To understand this you need to dive into the codebases of Spring Cloud Sleuth and Micrometer.io to check out how it exactly works.

But long story short:

* Spring Cloud Sleuth uses a [BeanPostProcessor](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/config/BeanPostProcessor.html" \t "_blank) and [RestTemplateCustomizer](https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/web/client/RestTemplateCustomizer.html" \t "_blank) to register it’s interceptor to be able to have the traceId and spanId in the log. See: TraceRestTemplateBeanPostProcessor and TraceRestTemplateBeanPostProcessor in the [TraceWebClientAutoConfiguration](https://github.com/spring-cloud/spring-cloud-sleuth/blob/master/spring-cloud-sleuth-core/src/main/java/org/springframework/cloud/sleuth/instrument/web/client/TraceWebClientAutoConfiguration.java" \t "_blank).
* Micrometer.io is only using a [RestTemplateCustomizer](https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/web/client/RestTemplateCustomizer.html" \t "_blank) and no [BeanPostProcessor](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/config/BeanPostProcessor.html" \t "_blank) to configure the RestTemplate to record request metrics.

Since only the BeanProcessor of Sleuth is able to instrument the RestTemplate bean I created distributed tracing works, but metrics are still broken.  
It’s unacceptable to have broken RestTemplate metrics in production so let’s fix it!

**Let’s fix it! Attempt 2**

In this section we will make the final fix to also make the metrics work again by using the [RestTemplateBuilder](https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/web/client/RestTemplateBuilder.html" \t "_blank) to create the RestTemplate!

Branch: [step-3-rest-template-using-rest-template-builder](https://github.com/stockgeeks/spring-boot-rest-template-build-example/tree/step-3-rest-template-using-rest-template-builder)

Build the RestTemplate using the RestTemplateBuilder.

Once again restart the hello-world-service

./mvnw spring-boot:run -pl hello-world-service

And call the hello endpoint on the hello-world-service:

curl <http://localhost:8080/api/hello>

Now when we request the RestTemplate metrics. We see finally have working metrics!

curl <http://localhost:8080/actuator/metrics/http.client.requests>{  
 name: "http.client.requests",  
 description: "Timer of RestTemplate operation",  
 baseUnit: "seconds",  
 measurements: [  
 {  
 statistic: "COUNT",  
 value: 1  
 },  
 {  
 statistic: "TOTAL\_TIME",  
 value: 0.395653157  
 },  
 {  
 statistic: "MAX",  
 value: 0  
 }  
 ],  
 availableTags: [  
 {  
 tag: "method",  
 values: [  
 "GET"  
 ]  
 },  
 {  
 tag: "clientName",  
 values: [  
 "localhost"  
 ]  
 },  
 {  
 tag: "uri",  
 values: [  
 "/api/random-name"  
 ]  
 },  
 {  
 tag: "status",  
 values: [  
 "200"  
 ]  
 }  
 ]  
}

**Conclusion**

The main takeaways of my blogpost:

* Always use a [RestTemplateBuilder](https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/web/client/RestTemplateBuilder.html" \t "_blank) to create an instance of your RestTemplate!
* Using a builder is not specific to a RestTemplate only. Many other Spring / Spring Boot classes like the WebClient have a [builder](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/reactive/function/client/WebClient.Builder.html).
* For the most up to date information about RestTemplate and the RestTemplateBuilder check out the [official Spring Boot documentation](https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-resttemplate.html).
* Don’t take answers given on Stackoverflow like these:  
  - <https://stackoverflow.com/a/36151777>  
  - <https://stackoverflow.com/a/40339656>  
  for granted they are outdated!
* Do a deep dive yourself into the Spring Framework / Spring Boot / Spring Cloud codebases to understand how BeanPostProcessor’s and customizers (like the RestTemplateCustomizer) work and how libraries like Sleuth and Micrometer.io implement their own BeanPostProcessors and or customizers so they can to customize a Spring Bean.