* [**Start Here**](https://www.baeldung.com/start-here)
* [**Courses ▼**](javascript:void(0))
* [**Guides ▼**](javascript:void(0))
* [**About ▼**](javascript:void(0))

Test a REST API with curl

5.5M

Last updated: March 9, 2021

Written by:[baeldung](https://www.baeldung.com/author/baeldung)

* [**REST**](https://www.baeldung.com/category/rest)

**Get started with Spring 5 and Spring Boot 2, through the reference *Learn Spring* course:**

[**>> CHECK OUT THE COURSE**](https://www.baeldung.com/ls-course-start)

**1. Overview**

This tutorial gives a brief overview of testing a REST API using *curl.*

***curl* is a command-line tool for transferring data, and it supports about 22 protocols, including HTTP.** This combination makes it a very good ad-hoc tool for testing our REST services.

**Further reading:**

[**Testing Web APIs with Postman Collections**](https://www.baeldung.com/postman-testing-collections)

Learn how to create a Postman Collection that can test a REST API

[**Read more**](https://www.baeldung.com/postman-testing-collections)→

[**A Guide to REST-assured**](https://www.baeldung.com/rest-assured-tutorial)

Explore the basics of REST-assured - a library that simplifies the testing and validation of REST APIs.

[**Read more**](https://www.baeldung.com/rest-assured-tutorial)→

**2. Command-line Options**

***curl* supports over 200 command-line options**. We can have zero or more of them to accompany the URL in the command.

Before we use it for our purposes, let's take a look at two that would make our lives easier.

**2.1. Verbose**

When we're testing, it’s a good idea to set the verbose mode on:

curl -v http://www.example.com/Copy

As a result, the commands provide helpful information such as the resolved IP address, the port we're trying to connect to, and the headers.

**2.2. Output**

By default, *curl* outputs the response body to standard output. Additionally, we can provide the output option to save to a file:

curl -o out.json http://www.example.com/index.htmlCopy

This is especially helpful when the response size is large.

**3. HTTP Methods With *curl***

Every HTTP request contains a method. The most commonly used methods are GET, POST, PUT and DELETE.

**3.1. GET**

This is the default method when making HTTP calls with*curl*. In fact, the examples previously shown were plain GET calls.

While running a local instance of a service at port 8082, we'd use something like this command to make a GET call:

curl -v http://localhost:8082/spring-rest/foos/9Copy

Since we have the verbose mode on, we get a little more information along with the response body:

\* Trying ::1...

\* TCP\_NODELAY set

\* Connected to localhost (::1) port 8082 (#0)

> GET /spring-rest/foos/9 HTTP/1.1

> Host: localhost:8082

> User-Agent: curl/7.60.0

> Accept: \*/\*

>

< HTTP/1.1 200

< X-Application-Context: application:8082

< Content-Type: application/json;charset=UTF-8

< Transfer-Encoding: chunked

< Date: Sun, 15 Jul 2018 11:55:26 GMT

<

{

"id" : 9,

"name" : "TuwJ"

}\* Connection #0 to host localhost left intactCopy

**3.2. POST**

We use this method to send data to a receiving service, which means we use the data option.

The simplest way of doing this is to embed the data in the command:

curl -d 'id=9&name=baeldung' http://localhost:8082/spring-rest/foos/newCopy

Alternatively, we can pass a file containing the request body to the data option like this:

curl -d @request.json -H "Content-Type: application/json"

http://localhost:8082/spring-rest/foos/newCopy

By using the above commands as they are, we may run into error messages like the following one:

{

"timestamp" : "15-07-2018 05:57",

"status" : 415,

"error" : "Unsupported Media Type",

"exception" : "org.springframework.web.HttpMediaTypeNotSupportedException",

"message" : "Content type 'application/x-www-form-urlencoded;charset=UTF-8' not supported",

"path" : "/spring-rest/foos/new"

}Copy

This is because *curl* adds the following default header to all POST requests:

Content-Type: application/x-www-form-urlencodedCopy

This is also what the browsers use in a plain POST. In our usage, we’d usually want to customize the headers depending on our needs.

For instance, if our service expects JSON content-type, then we can use the -H option to modify our original POST request:

curl -d '{"id":9,"name":"baeldung"}' -H 'Content-Type: application/json'

http://localhost:8082/spring-rest/foos/newCopy

Windows command prompt has no support for single quotes like the Unix-like shells.

As a result, we'd need to replace the single quotes with double quotes, though we try to escape them wherever necessary:

curl -d "{\"id\":9,\"name\":\"baeldung\"}" -H "Content-Type: application/json"

http://localhost:8082/spring-rest/foos/newCopy

Besides, when we want to send a somewhat larger amount of data, it is usually a good idea to use a data file.

**3.3. PUT**

This method is very similar to POST, but we use it when we want to send a new version of an existing resource. In order to do this, we use the -X option.

Without any mention of a request method type, *curl* defaults to using GET; therefore, we explicitly mention the method type in the case of PUT:

curl -d @request.json -H 'Content-Type: application/json'

-X PUT http://localhost:8082/spring-rest/foos/9Copy

**3.4. DELETE**

Again, we specify that we want to use DELETE by using the -X option:

curl -X DELETE http://localhost:8082/spring-rest/foos/9Copy

**4. Custom Headers**

We can replace the default headers or add headers of our own.

For instance, to change the Host header, we do this:

curl -H "Host: com.baeldung" http://example.com/Copy

To switch off the User-Agent header, we put in an empty value:

curl -H "User-Agent:" http://example.com/Copy

The most common scenario while testing is changing the Content-Type and Accept header. We just have to prefix each header with the -H option:

curl -d @request.json -H "Content-Type: application/json"

-H "Accept: application/json" http://localhost:8082/spring-rest/foos/newCopy

**5. Authentication**

A [service that requires authentication](https://www.baeldung.com/spring-security-basic-authentication) would send back a 401 – Unauthorized HTTP response code, and an associated WWW-Authenticate header.

For basic authentication, we can **simply embed the username and password combination inside our request using the user option**:

curl --user baeldung:secretPassword http://example.com/Copy

However, if we want to [use OAuth2 for authentication](https://www.baeldung.com/rest-api-spring-oauth2-angularjs), we first need to get the *access\_token* from our authorization service.

The service response would contain the *access\_token:*

{

"access\_token": "b1094abc0-54a4-3eab-7213-877142c33fh3",

"token\_type": "bearer",

"refresh\_token": "253begef-868c-5d48-92e8-448c2ec4bd91",

"expires\_in": 31234

}Copy

Now we can use the token in our Authorization header:

curl -H "Authorization: Bearer b1094abc0-54a4-3eab-7213-877142c33fh3" http://example.com/Copy

**6. Conclusion**

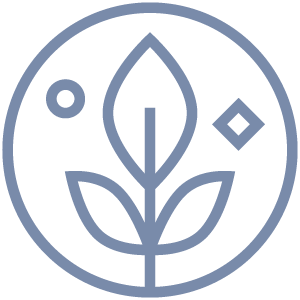
In this article, we demonstrated using the bare minimum functionality of*curl* to test our REST services. Although it can do much more than what was discussed here, for our purposes, this much should suffice.

Feel free to type *curl* -h on the command line to check out all of the available options. The REST service used for the demonstration is available [here on GitHub](https://github.com/eugenp/tutorials/tree/master/spring-web-modules/spring-rest-simple).

**Get started with Spring 5 and Spring Boot 2, through the *Learn Spring* course :**

[**>> CHECK OUT THE COURSE**](https://www.baeldung.com/ls-course-end)





Learning to build your API  
**with Spring**?

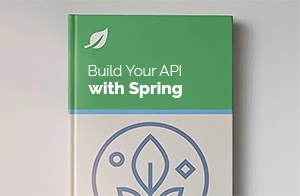
[**Download the E-book**](https://www.baeldung.com/rest-api-spring-guide)

**2 COMMENTS**

Oldest

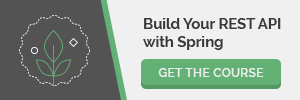
View Comments

Comments are closed on this article!



Building a REST API with Spring 5?

[**Download the E-book**](https://www.baeldung.com/rest-api-spring-guide)

[](https://www.baeldung.com/rws-course)

**COURSES**

* [ALL COURSES](https://www.baeldung.com/all-courses)
* [ALL BULK COURSES](https://www.baeldung.com/all-bulk-courses)
* [ALL BULK TEAM COURSES](https://www.baeldung.com/all-bulk-team-courses)
* [THE COURSES PLATFORM](https://courses.baeldung.com/)

**SERIES**

* [JAVA “BACK TO BASICS” TUTORIAL](https://www.baeldung.com/java-tutorial)
* [JACKSON JSON TUTORIAL](https://www.baeldung.com/jackson)
* [APACHE HTTPCLIENT TUTORIAL](https://www.baeldung.com/httpclient-guide)
* [REST WITH SPRING TUTORIAL](https://www.baeldung.com/rest-with-spring-series)
* [SPRING PERSISTENCE TUTORIAL](https://www.baeldung.com/persistence-with-spring-series)
* [SECURITY WITH SPRING](https://www.baeldung.com/security-spring)
* [SPRING REACTIVE TUTORIALS](https://www.baeldung.com/spring-reactive-guide)

**ABOUT**

* [ABOUT BAELDUNG](https://www.baeldung.com/about)
* [THE FULL ARCHIVE](https://www.baeldung.com/full_archive)
* [EDITORS](https://www.baeldung.com/editors)
* [JOBS](https://www.baeldung.com/tag/active-job/)
* [OUR PARTNERS](https://www.baeldung.com/partners)
* [PARTNER WITH BAELDUNG](https://www.baeldung.com/advertise)
* [TERMS OF SERVICE](https://www.baeldung.com/terms-of-service)

* [PRIVACY POLICY](https://www.baeldung.com/privacy-policy)

* [COMPANY INFO](https://www.baeldung.com/baeldung-company-info)

* [CONTACT](https://www.baeldung.com/contact)