1 Georouting API Docker containers

The Georouting API Docker containers can be accessed from

- the public docker repository: https://registry.dpsin.dpdgroup.com/
  - user: georouting
  - password: jhz5s3bzKsAj

1.1 Features

To be completed

1.2 Differences in images

There are four separate images distributed currently. The functionalities of the images are the same, but there are some differences during startup.

1.2.1 geor_docker_fulldb

This image has already an up to date Geordb file imported from RDB sftp folder. If it is started, it is ready to be used within 1 minute. The drawback is that this image is bigger and needs to be updated frequently (monthly) from sftp. The RDB is loaded for RDB sftp folder

1.2.2 geor_docker_w_imp_doall

This image is smaller, because it is shipped with an empty Db. The drawback is the image needs 0.5-2 hour to import and expand a database, before it can answer routing requests. This image doesn't need to be updated from Sftp. The RDB is loaded for RDB sftp folder

1.2.3 GeoRouting Docker tags

There are 3 different tags:

- **Master**
  - Published GR API version
- **Stable**
  - Validated GR API version, on DPD IT Preprod env
- **Develop**
  - On testing GR API version

The tag to use is latest-master, It permits to have the latest published version of the GR API

2 Installation and starting Georouting Docker API

2.1 Requirements

2.1.1 Hardware requirement

2.1.1.1 Hardware requirement of the Docker Host

To run this container you need minimum the following hardware on the host:

- x86_64 CPU
- 4 GB ram
- 100 GB Disk
- Internet access

The optimal hardware is:

- x86_64 CPU, with 2 cores
- 8 GB ram
- 100 GB SSD Disk
- Internet access

2.1.1.2 Resource requirement for the container

The container needs the minimum the following resources:

- 1 CPU core
- 4 GB ram
- 30 GB Disk
- Internet access

The optimal resources are

- 2 or more CPU core
- 8 GB ram
- 30 GB Disk
- Internet access

2.1.2 Software requirement

This is a linux based Docker Container, so it needs either a linux machine, with Docker, or a windows machine, installed with Docker for Windows in Linux container mode.

The minimum required Docker version is 17.03.2-ce.

2.2 Starting Docker Georouting API

Currently there is no Docker repository for these images: they are distributed as zip files. The following steps needs to be made on Docker host, before the container can be started.

2.2.1 Importing docker Image

The distributed file needs to be extracted first you can step forward.

2.2.1.1 Importing in a standalone server

Before to load the image, you must login into the nexus repository:

```
docker login -u georouting -p jhz5s3bzKsAj https://registry.dpsin.dpdgroup.com/
```

The image can be imported with a load command, by giving a new image id for it.

Image can be imported with the following command:

```
docker pull registry.dpsin.dpdgroup.com/geor_docker_fulldb:latest-master
```

If the command ended successfully, then it will show the imported image name. This could be used in next command to start the Image.

2.2.1.2 Starting in a managed environment or in cloud

If there is a docker swarm installation, or if the container needs to be started in the cloud, then first
you need to import it to a private registry. Please consult with your manual how to do this. Please note that this image is a property of Geopost, so it cannot be uploaded to a public registry.

2.2.2 Starting docker image

The docker image can be started by the following command:

```
docker run -P <image name>
```

2.2.3 Keeping the docker image up to date

To ensure you are using the latest version of the Docker image, run the following command to pull the most recent image:

```
docker pull registry.dpsin.dpdgroup.com/geor_docker_fulldb:latest-master
```

This command downloads the latest version of the image, ensuring that any updates or changes are applied. We recommend running this command periodically to stay up to date e.g., the 10th of every month.

After pulling the latest image, you need to restart your container to apply the changes

To ensure you are using the latest version of the Docker image, run the following command to pull the most recent image:

```
docker pull registry.dpsin.dpdgroup.com/geor_docker_fulldb:latest-master
```

This command downloads the latest version of the image, ensuring that any updates or changes are applied. We recommend running this command periodically to stay up to date e.g., the 10th of every month.

After pulling the latest image, you need to restart your container to apply the changes

Use the following commands:

1. Stop the running container:

   ```
   docker stop <container_name>
   ```

2. Remove the stopped container (optional):

   ```
   docker rm <container_name>
   ```

3. Start a new container with the latest image:

   ```
   docker run -P registry.dpsin.dpdgroup.com/geor_docker_fulldb:latest-master
   ```

However, there are several commands to configure the image,

2.3 Docker parameters

2.3.1 Publish ports

Although the port 8080 is exposed by the dockerfile, by default it is assigning the ports randomly.

With the following commands the rest service port and admin port could be published

Name	Format	Example
docker publish rest service endpoint	[IP]:[PORT]:[PORT] <public ip for serviec>:8080:8080	-p 8080:8080
docker publish port for admin frontend	[IP]:[PORT]:[PORT] <protected ip>:8081:8081	-p 127.0.0.1:8081:8081

Please note, that admin fronted port should not be accessed by the client of this rest service, it should be either opened only for localhost, (and may use ssh port forward to access it remotely), or use firewall / proxy rules to restrict the access to it.

2.3.2 SSL config

The Tomcat server inside the API support SSL configuration, just correct certificate and key needs to be provided. Please note that the CNAME field should be equal to the outside visible hostname.

Name	Format	Example
Add Tomcat SSL Certificate File	-v [.cert filePath]:/tmp/geortool/ssl.cert	-v C:\ \Dockerfile\server.cert:/tmp/geortool/ssl.cert
Add Tomcat SSL Certificate Key File	-v [.key filePath]:/tmp/geortool/ssl.key	-v C: \Dockerfile\server.key:/tmp/geortool/ssl.key

2.3.3 Configuring the API and API Tools

Name	Format	Example
Override the config file with an external one	-v /<path>/georouting_toolsPostgres.properties: /usr/geortool/georouting_toolsPostgres.properties	-v /config/georouting_toolsPostgres.properties: /usr/geortool/georouting_toolsPostgres.properties
override the config via parameters: this way the config can be overridden as a java parameters. It could be a complete config, or it could be only a few paramters	-e "GEORPROPERTIESCONFIG=[tools configurations]" (String type)	-e "GEORPROPERTIESCONFIG=-Dcom.geopostgroup.GeoRouting.debuglevel=1 -Dcom.geopostgroup.GeoRouting.SqlLiteDir=db/ -Dcom.geopostgroup.GeoRouting.workdir=db/temp/ -Dcom.geopostgroup.GeoRouting.JdbcStr=jdbc:postgresql://localhost:5432/postgres?user=postgres&password=postgres -Dcom.geopostgroup.GeoRouting.UpdateDownload=4 -Dcom.geopostgroup.GeoRouting.UpdateType=sftp -Dcom.geopostgroup.GeoRouting.sftphost=ftp-geodata.geopostgroup.com -Dcom.geopostgroup.GeoRouting.sftpuser=sftpuser -Dcom.geopostgroup.GeoRouting.sftppass=sftppass -Dcom.geopostgroup.GeoRouting.sftpdir=/OUT/tmp"

2.3.4 Set secret code for rest api

By this command, a secret code could be set, which needs to be added as a get parameter for all API call:

Name	Format	Example
Add rest secret code for docker	-e "REST_SERVICE_SECRET=[secretcode for backend]" (String type)	-e "REST_SERVICE_SECRET=secretCode"

2.3.5 Have a volume mapping for Postgresql database

Currently the API uses an inner volume for the container, defined during build.

Please do not use volume mapping for geor_docker_fulldb, because the container would start with empty directory, and needs to import data, as geor_docker_w_imp_doall.

The volume could be mapped by a command, and in this way the container does not need to start import data from beginning, if it was started before. Please note that you have to mount separate directory, if more than on container is started on the same host.

In some cases, for example Postgresql is upgraded in the image, the volume might need to be re-created, because newer minor version of DB uses different file format. Although upgrading the Postgresql backend will not happen frequently (maybe every 2-3 years once) instructions would be given on release notes, if such step needs to be made.

Please note that the referenced directory needs to be exists, and it should not be a mounted folder (or it should support proper fsync methods) to avoid data corruption.

The directory needs to be empty, othervise the initialization will not be done, and Postgres cannot start successfully.

Name	Format	Example
Use a blind volume mapping for postgres	-v /<path-to-folder for postgres>/:/var/lib/postgresql/data	-v /var/lib/postgresql/data/:/var/lib/postgresql/data

2.3.6 Change the password for Admin page

The default username & password for admin page is user / pass.

The username and password are configurable for admin page, with in 2 ways:

2.3.6.1 Set admin page user & pass via environment variable

The default password can be overridden by a simple environment variable. However this simple solution is not the safest way to provide passwords.

Name	Format	Example
username & password for admin page	-e loginuser=<username> -e loginpass=<password>	-e loginuser=secretadmin -e loginpass= `WTXvv7WnCVm0vGq5`

2.3.6.2 Set admin page user & pass via property file (safest)

In this way, the username and password could be hidden,

- Java property file needs to be added to the container, by volume mapping for example (preferably it can come from Docker secret), with the following keys:
  - loginuser
  - loginpass

  For example file content could be:

  ```
  loginuser=geopostuser
  loginpass=WTXvv7WnCVm0vGq5
  ```
- Filename must be set with the user.properties environment variable

Name	Format	Example
file for admin page username & secret	-e user.properties=/path/to/file/inContainer	-e user.properties=/etc/dockerAdminPass.porperties

2.3.7 Caching

The webapp is able to cache request result to speed up responses. The following Rest endpoints are cached:

- /productSelection
- /productSelectionList
- /route

Cache stores most frequently used results, and it can answer request from the cache as long as no new Geordb data needs to be used for that request.

Caching is disabled by default.

2.3.7.1 Enable caching

To enable caching, a spring profile needs to be activated through env variables, with the following command line parameter.

Name	Format	Example
Enable cache	-e "SPRING_PROFILES_ACTIVE=cachingEnabled"	-e "SPRING_PROFILES_ACTIVE=cachingEnabled"

2.3.7.2 Cache size

There is 2 separate cache for:

- Route
- ProductSelection

By default the cache size is configured to 1M, which means 1 MB of Heap memory will be used for both cache.

This setting is a safety setting (without this config, or 0M the application cannot be started), will provide only a relatively medium improvement: one cache entry is about 100-200 byte, so this amount of memory may be enough to host 5000 route result.

Cache size can be set by the following commands:

Name	Format	Example
Route cache size	-e "georouting_routeResultCacheSize=<mem>"	-e "georouting_routeResultCacheSize=100M"
ProductSelection cache size	-e “georouting_productSelectionResultCacheSize=<mem>”	-e “georouting_productSelectionResultCacheSize=100M”
Tomcat heap size	-e “JAVA_OPTS=-Xmx< route cache size+ productSeleciton cache size + 1G>”	-e “JAVA_OPTS=-Xmx1200M”

Route cache size set to 1G would result ~ 5 million routing result to be stored. If the container used with a single origin, then this amount of cache could be enough to store all frequently used route result (one country with 5 digit postcode, and ~ 40 000 postcode may result 100k-500k route entry, depending on how many different service is used)

IMPORTANT! If you increase the cache size, don’t forget to set the heap size as well! If webapp runs out of heap, might stop working! Set the heap size + 1G of the full cache size.

It is also important, to not assign all memory to Tomcat, because Postgresql needs some shared memory (by default it is set to 1.5GB)

2.4 HealthCheck

The docker container health status could be monitored by the following request:

- Request type: POST / GET
- URL: http://<address of container or 127.0.0.1 if it is inside a container>:8080/GeoRouting-WS-R/route-for-check-healthy

Request will return http status 200 if it is a success, and a result of a routing (this does not need to be validated).

Because the healthcheck executes a routing, it might take up to 1 hour to report a first healthly state (if db is just updated. for a full image this should be relatively short max 3 minute).

For command based checks, there is a dedicated shell script: /healthcheck.sh.

3 Start the api

3.1 Rest api

After the docker image is started the Rest service will become ready (if 5.2.2 geor_dowcker_w_imp_doall is used, it could take some time, up to 1 hour) , can be called, or the Swagger UI can be opened in url:

http://127.0.0.1:8080/GeoRouting-WS-R/swagger-ui.html

Please change the port in the URL, if you published it for a different port.
For more information, please consult with Georouting_rest_api.doc

3.2 Admin page

Admin page is accessible in the following url:

http://127.0.0.1:8080/GeoRouting-WS-R-WEB/

The functions of the admin page is described in document GEOROUTING_Docker_API_Admin.docx

4 Getting started

4.1 Install Docker

4.1.1 Docker download page

- Go on https://docs.docker.com/get-docker/

4.1.2 Windows installation

- Click on **Docker Desktop for Windows**

- Click on **Download from Docker Hub**

- Click on **Get Docker**
- Execute the downloaded file **“Docker Desktop Installer.exe**”
- Click on **Yes**
- Click on **OK**

- Click on **Close**
- Docker installation is **finished**
- Restart your computer if it’s requested
- In case of issue during the Docker installation refer to the docker documentation and to your IT Help

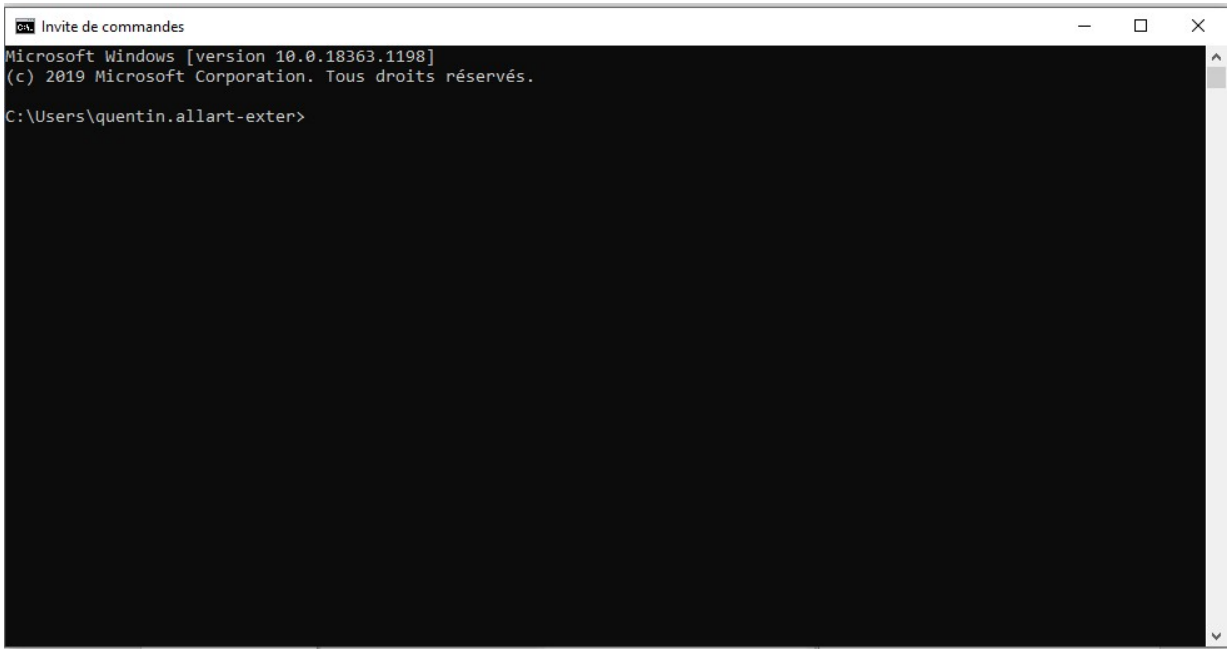
4.2 Run the Docker GeoRouting API container

4.2.1 Open the CMD

- Open Windows command line **CMD**

4.2.2 Login into the registry

- Enter in the CMD the following command:

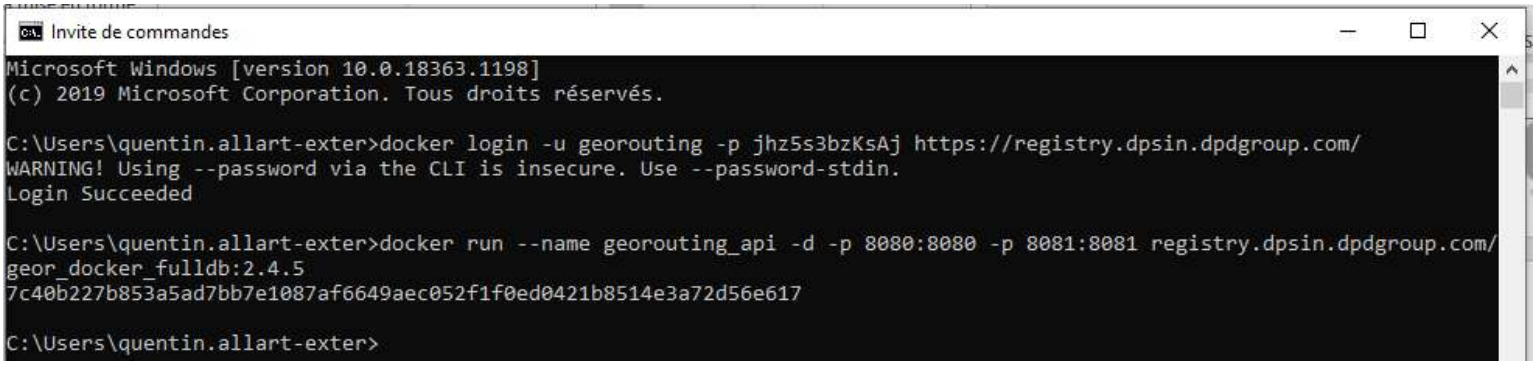


```
docker login -u georouting -p jhz5s3bzKsAj https://registry.dpsin.dpdgroup.com/
```

4.2.3 Run the container

- Enter in the CMD the following command:

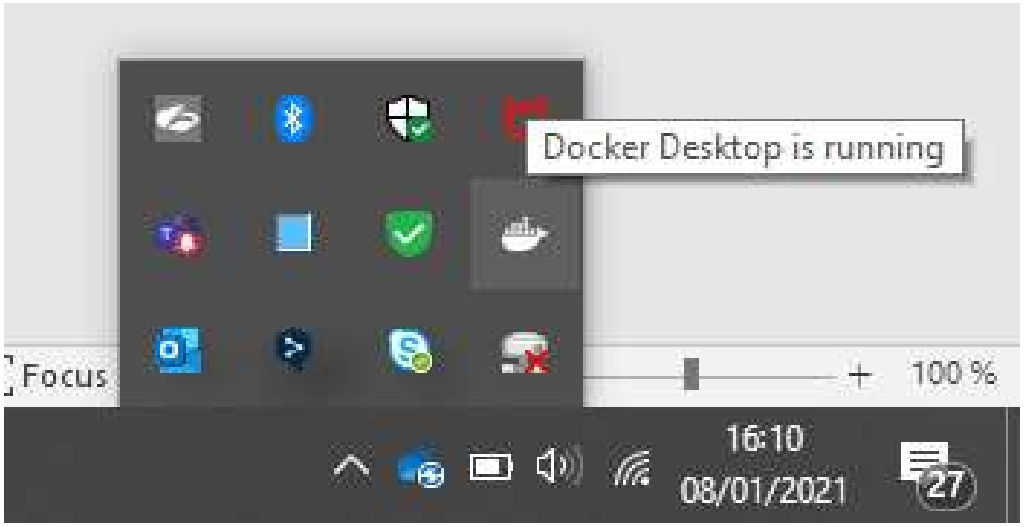
```
docker run --name georouting_api -d -p 8080:8080 -p 8081:8081
registry.dpsin.dpdgroup.com/geor_docker_fulldb:latest-master
```



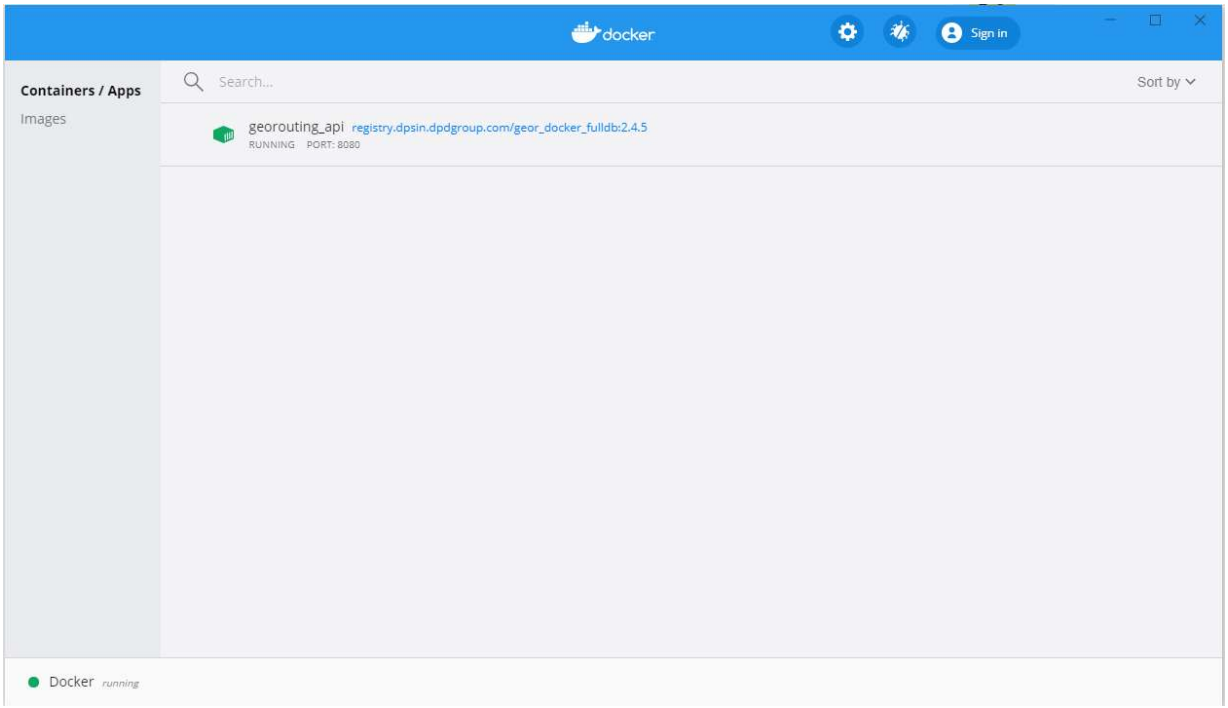
- The Georouting API docker container is now running

4.3 **Check application**

- Go in your running application menu and click on **Docker Desktop**



- In Docker Desktop you can see your **running container**



4.4 Access to Admin page

- http://127.0.0.1:8081/GeoRouting-WS-R-WEB/login

- The default username & password for admin page is user / pass.

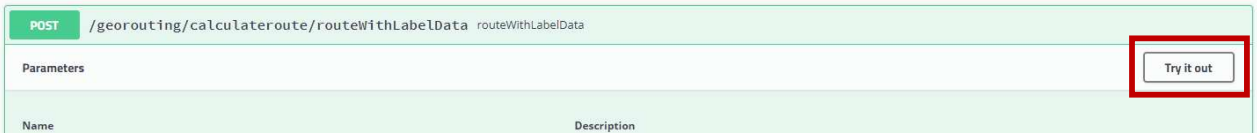
4.5 Access to the API / SWAGGER

- http://127.0.0.1:8080/GeoRouting-WS-R/swagger-ui.html

4.6 Call: /georouting/calculateroute/routeWithLabelData

4.6.1 Via Swagger

- Go to http://127.0.0.1:8080/GeoRouting-WS-R/swagger-ui.html#/calculate-route-controller/routeWithLabelDataUsingPOST
- Click on Try Out

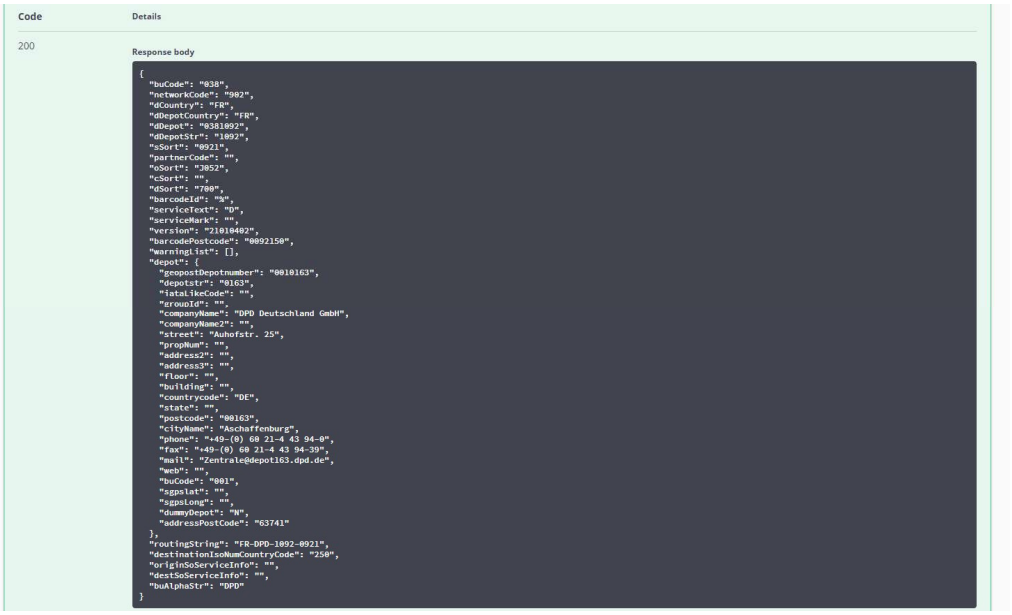


- Enter the data to send to the api

  Example:

```
{
  "date": "2021-01-18",
  "dcountry": "FR",
  "dpostcode": "92150",
  "odepot": "0010163",
  "soCode": "101"
}
```

- Click on execute
- This the result



Code

Details

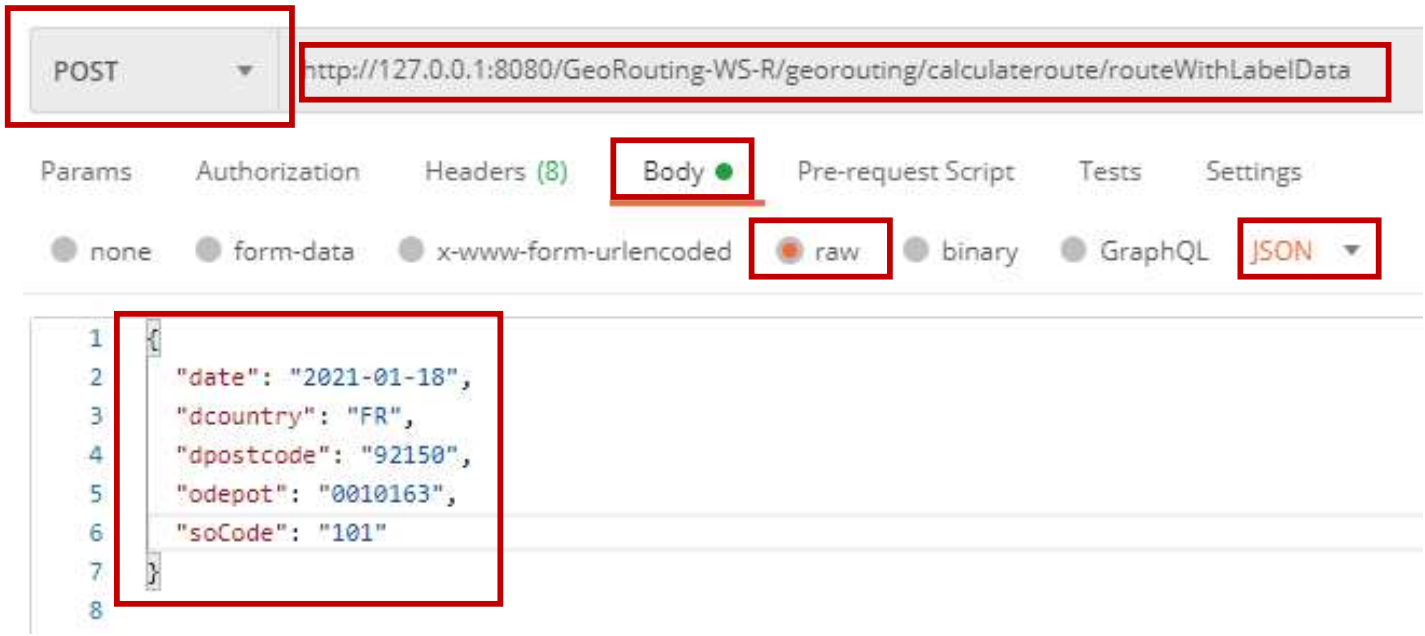
200

Response body


```
{
  "buCode": "038",
  "networkCode": "982",
  "dCountry": "FR",
  "dDepotCountry": "FR",
  "dDepot": "0381092",
  "dDepotStr": "1092",
  "sSort": "0921",
  "partnerCode": "",
  "oSort": "3052",
  "cSort": "",
  "dSort": "709",
  "barcodeId": "M",
  "serviceText": "D",
  "serviceMark": "",
  "version": "21019402",
  "barcodePostcode": "0092150",
  "warningList": [],
  "depot": {
    "geopostDepotnumber": "0010163",
    "depotstr": "0163",
    "iataLikeCode": "",
    "groupId": "",
    "companyName": "DPD Deutschland GmbH",
    "companyName2": "",
    "street": "Auhofstr. 25",
    "propNum": "",
    "address2": "",
    "address3": "",
    "floor": "",
    "building": "",
    "countrycode": "DE",
    "state": "",
    "postcode": "00163",
    "cityName": "Aschaffenburg",
    "phone": "+49-(0) 60 21-4 43 94-0",
    "fax": "+49-(0) 60 21-4 43 94-39",
    "mail": "Zentrale@depot163.dpd.de",
    "web": "",
    "buCode": "001",
    "sgpslat": "",
    "sgpsLong": "",
    "dummyDepot": "N",
    "addressPostCode": "63741"
  },
  "routingString": "FR-DPD-1092-0921",
  "destinationIsoNumCountryCode": "250",
  "originSoServiceInfo": "",
  "destSoServiceInfo": "",
  "buAlphaStr": "DPD"
}
```

4.6.2 Via Postman

- Open Postman
- Create a new request
- Select POST
- Enter the url: http://127.0.0.1:8080/GeoRouting-WS-R/georouting/calculateroute/routeWithLabelData
- Select raw for the request Body and JSON as type



- Click on Send
- The result should be in the response Body as in the above image