Deploy Machine Learning Models as a REST API

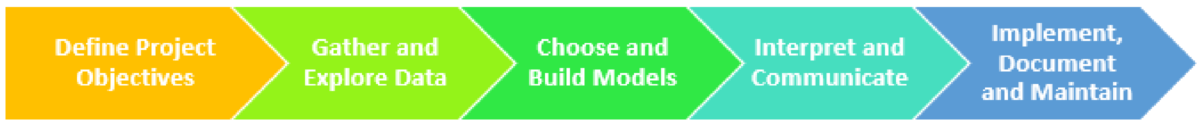
This blog is intended to explain **"how a machine learning model"** can be exposed as a **REST API endpoint** to serve ML models.

Before getting into machine learning model deployment, let us try to understand the different steps/processes involved in Machine Learning Life Cycle. Implementing/Deploying the machine learning models is one of the steps in the entire life cycle.

Note: Building a machine learning model is not discussed in this blog.

The Machine Learning Project Life Cycle
The machine learning project life cycle defines the processes/steps (specifically for data science related projects) that an organization should follow to take advantage of machine learning (ML) and artificial intelligence (AI) to derive practical business values.

There are five major steps:


Define Project Objectives
Gather and Explore Data
Choose and Build Models
Interpret and Communicate
Implement, Document and Maintain

(Figure 1: Machine Learning Project Life Cycle)

Step 1 : Define Business/Project Objectives

The first step of the life cycle is to identify an opportunity to tangibly improve operations, increase customer satisfaction, or otherwise create value to the business.

Every individual project is initiated with an objective. Project objectives in project management are the outcomes and deliverables that define the success of your project. Basically, it describes the "what" of your project.

Building machine learning models are also starting with "defining project objectives". Every project stakeholder should clearly understand the below before stepping on to the next steps:

- What specific task should our model be automating?
- How does the user interact with the model?
- What information should we expose to the user?

The primary purpose of machine learning is to discover patterns in the data and make predictions based on these and intricate patterns for answering business questions and solving business problems.

Step 2: Collect and Explore Data
The next step is to collect and prepare all the relevant data for the machine learning models. This step is also called as Data Preparation. The project team will closely with domain experts and customers to determine what data might be relevant for machine learning models. In this project life cycle, the project team will perform the below activities:

- Data Collection
- Data Preparation
  - Munge data and prepare it for training
  - Data Cleaning – Remove duplicates, deal with missing values, normalization, data type conversions, etc.
  - Conduct Exploratory Data Analysis

- Split into training and evaluation sets

According to Forbes, the data preparation accounts for about 80% of the work of data scientists.

Step 3: Choose, Build and Evaluate Machine Learning Models

In this step, the project team involve building different machine models (based on the project requirements), measure each model outcomes, comparing the models to one another, and investigate the errors from each model. The best performing models should be shortlisted, which can then be fine-tuned afterwards. The chosen model will also be evaluated for its performance against the unseen data. If needed, the models will undergo "parameter tuning" for performance improvements.

Step 4: Interpret and Communicate

One of the most difficult tasks of machine learning projects is explaining a model's outcomes to those without any data science background. Traditionally, machine learning has been thought of as a "black box" because it is difficult to interpret insights and communicate the value of those insights to stakeholders and regulatory bodies. The more interpretable your model, the easier it will be to meet requirements and communicate its value to management and other key stakeholders.

Step 5: Implement, Document and Maintain

The final step is to implement, document, and maintain the data science project so that the business can continue to leverage and improve upon.

Implementation

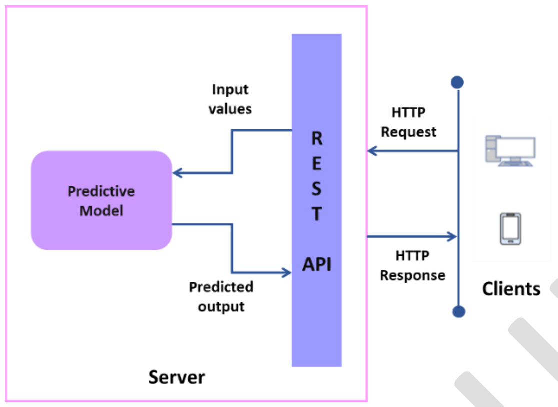
As mentioned earlier, this blog focuses more on **implementing the machine models** as a REST API so that the model can be consumed from outside world. There are a few implementation approaches. In this blog, the below are going to be discussed:

- Expose the model as a web service
- Machine learning model as FaaS (Function as a Service)

Option 1 : Expose the model as a Web Service

Exposing the model as a web service can be done by creating a **Python Flask** application with an endpoint that can take a **JSON** body of features and return a prediction based on those features.

The figure 2 describes the indicative architecture of implementing ML model as a Service



(Figure 2: ML model - Architecture)

The below steps to be implemented to expose the ML model as a Web Service

Step 1: Create a python file "<name>.py", add the following code to import the necessary packages.

```
from flask import Flask, request, jsonify
app = Flask(__name__)
from sklearn.externals import joblib
```

Flask will be used to create the web server. The packages request and jsonify will be used for processing the POST requests and responses.

This step will create a Web Server listening at http://127.0.0.1:5000

Step 2: Import the model file (pickle file) created for the ML model. The code shown below loads the pickle file created for "Logistic Regression" model.

```
reg = joblib.load("./logistic_regression_model.pkl")
```

In Python, Pickle is used for serializing and de-serializing a Python object structure. Any object in python can be pickled so that it can be saved on disk. Creating a pickle file for a Python object involves two steps:

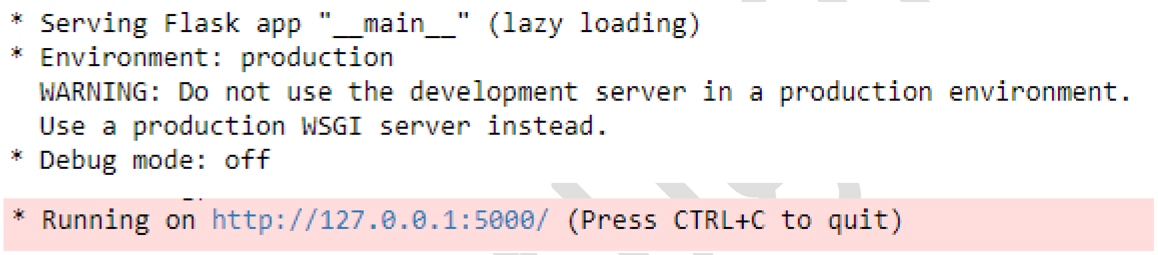
- ✓ First one is dump, which dumps an object to a file object
- ✓ Second one is load, which loads an object from a file object

Step 3: Add a route that will allow you to send a JSON body of features and will return a prediction

```
@app.route('/predictDiabetes',methods=['POST'])
def predict_diabetes():
    print(request.json)
    inp = request.json['input']
    global reg
    return jsonify({"value":int(reg.predict([inp])[0])})

if __name__ == '__main__':
    app.run(debug=False)
```

This step will add the endpoint "predictDiabetes" in the Web Server created in Step 1. The service will be consumed by the URL http://127.0.0.1:5000/predictDiabetes

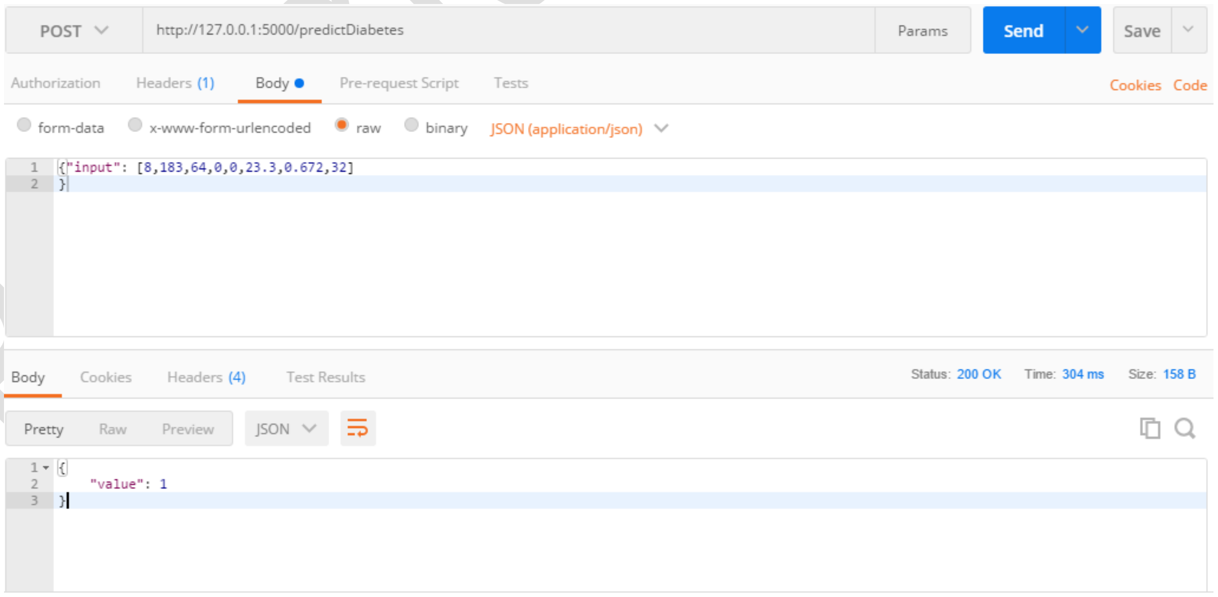


```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off

 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Step 4: Invoke the ML model REST API from Postman through the endpoint served by Flask. For the input data (features) passed, the ML model returns with a response either '1' – Yes or '2' – No with status "200 OK".

Request Body : `{"input": [8,183,64,0,0,23.3,0.672,32]}`



POST http://127.0.0.1:5000/predictDiabetes

```
{"input": [8,183,64,0,0,23.3,0.672,32]
}
```

Status: 200 OK Time: 304 ms Size: 158 B

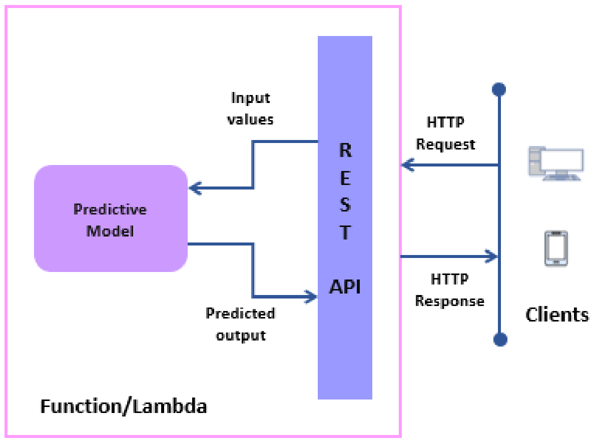
```
{
    "value": 1
}
```

Response : `{"value" : 1}`

Option 2 : Machine learning model as FaaS (Function as a Service)

Exposing the model as a Function (in AWS, GCP or Azure) can also be done by creating a **Python Flask** application with an endpoint that can take a **JSON** body of features and return a prediction based on those features.

The figure 3 describes the indicative architecture of implementing ML model as a Function



(Figure 3: ML model - Architecture)

In this blog, the ML model is deployed as a Function in Google Cloud Platform (GCP)

Exposing a ML model as a Function requires three components. They are

- main.py – to load the pickle file ("./logistic_regression_model.pkl"), include the packages request and jsonify for processing the POST requests and send back responses.
- requirements.txt – include the dependency packages with required versions.
- The dumped pickle file - "logistic_regression_model.pkl"

Note: All three components/files should be zipped as a single file before uploading on to GCP.

The Python module "main.py" should contain the below code:

```
def predict_diabetes(request):
    """HTTP Cloud Function.
    Args:
        request (flask.Request): The request object.
        <http://flask.pocoo.org/docs/1.0/api/#flask.Request>
    Returns:
        The response text, or any set of values that can be turned into a
        Response object using `make_response`
        <http://flask.pocoo.org/docs/1.0/api/#flask.Flask.make_response>.
    """
    from flask import request, jsonify
    from sklearn.externals import joblib
    reg = joblib.load("./logistic_regression_model.pkl")
    print(request.json)
    inp = request.json['input']
    return jsonify({"value":int(reg.predict([inp])[0])})
```

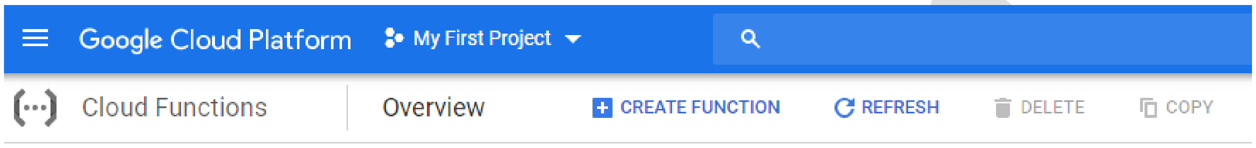
The statement “joblib.load” loads the pickle file. The input data is obtained by using “request.json” and “reg.predict” uses the input data, predicts either ‘1’ – Yes or ‘0’ – No

The “requirements.txt” should have

```
google-cloud-error-reporting==0.30.0scikit-learn==0.20.2
```

The below steps to be implemented to expose the ML model as a Function

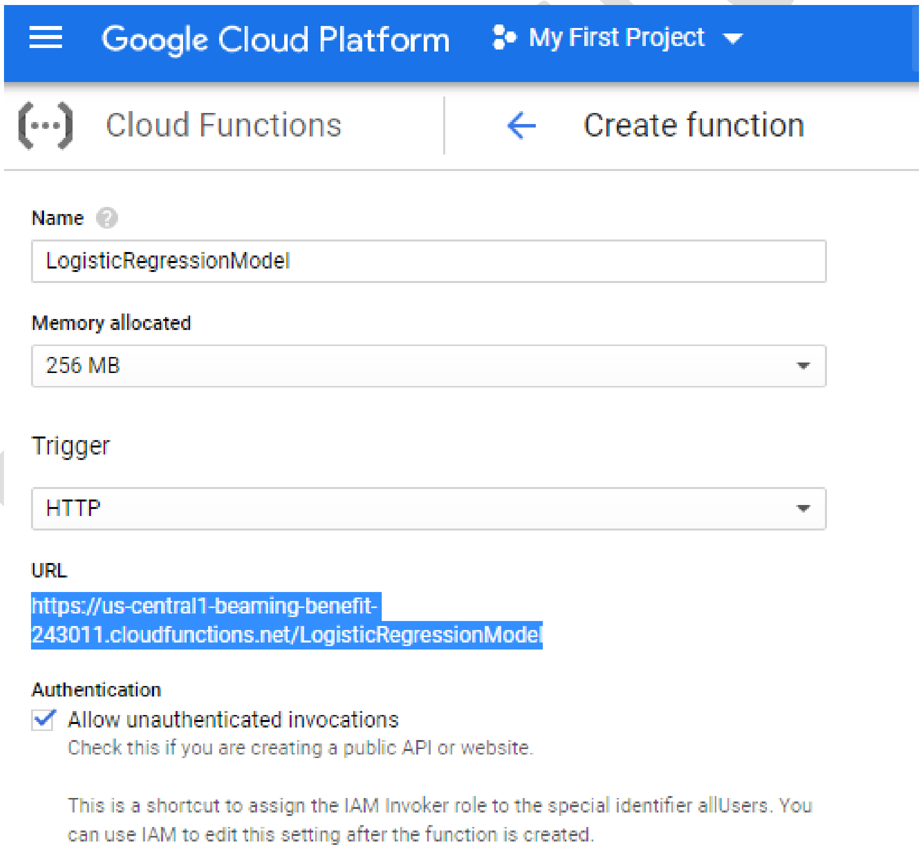
Step 1: Create a new Function under “Cloud Functions”.



Step 2: Provide a name for the function. Ex: LogisticRegressionModel

The new function can be consumed by accessing the below URL:

https://us-central1-beaming-benefit-243011.cloudfunctions.net/LogisticRegressionModel



Name

LogisticRegressionModel

Memory allocated

256 MB

Trigger

HTTP

URL

https://us-central1-beaming-benefit-243011.cloudfunctions.net/LogisticRegressionModel

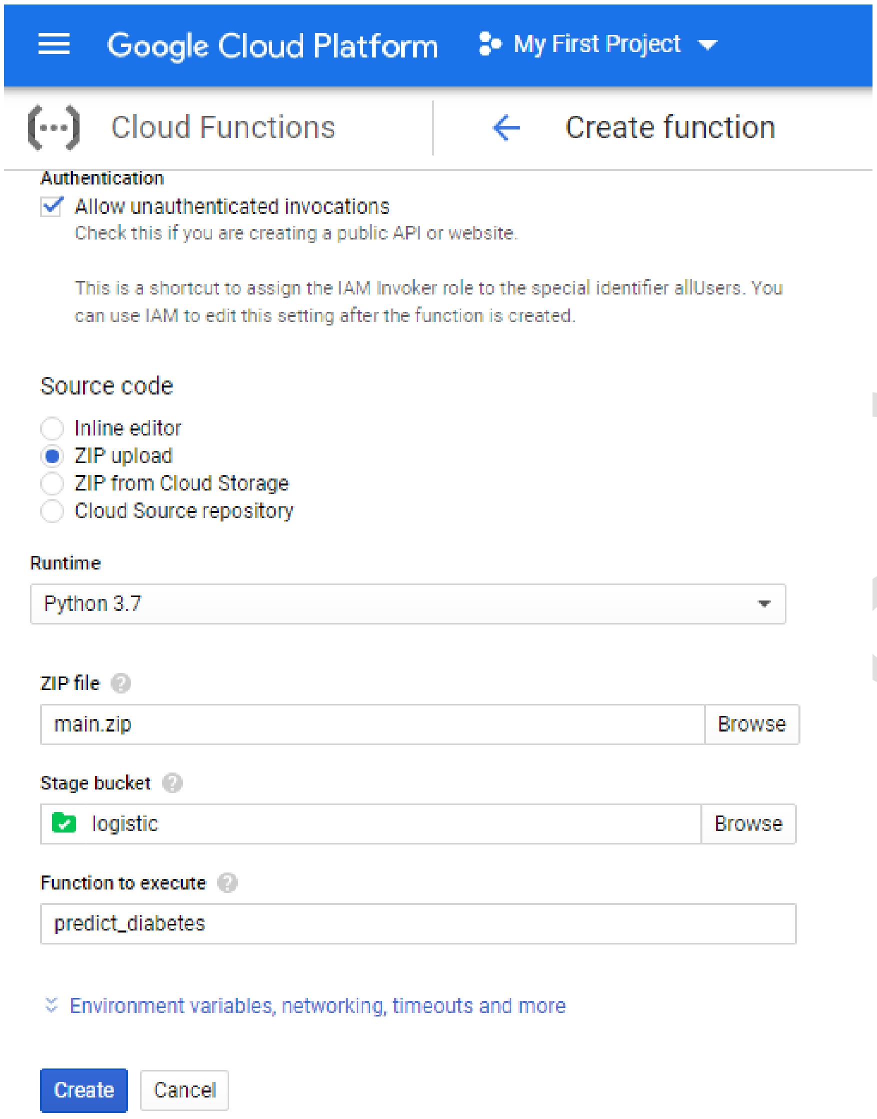
Authentication

☑ Allow unauthenticated invocations

Check this if you are creating a public API or website.

This is a shortcut to assign the IAM Invoker role to the special identifier allUsers. You can use IAM to edit this setting after the function is created.

Step 3: Select the option "ZIP upload", Runtime as "Python 3.7" and upload the zip file. The function to execute should have "predict_diabetes"



Authentication

[✓] Allow unauthenticated invocations

Check this if you are creating a public API or website.

This is a shortcut to assign the IAM Invoker role to the special identifier allUsers. You can use IAM to edit this setting after the function is created.

Source code

- Inline editor
- [✓ ZIP upload]
- ZIP from Cloud Storage
- Cloud Source repository

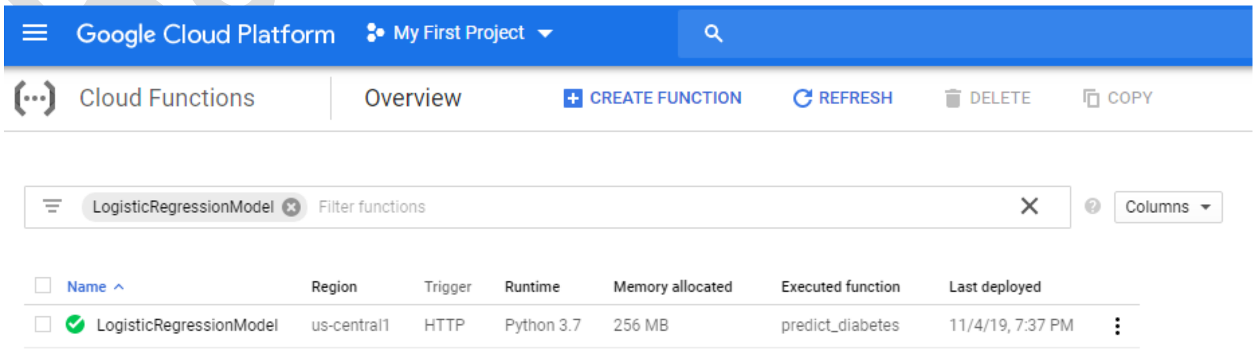
Runtime: Python 3.7

ZIP file: main.zip

Stage bucket: logistic

Function to execute: predict_diabetes

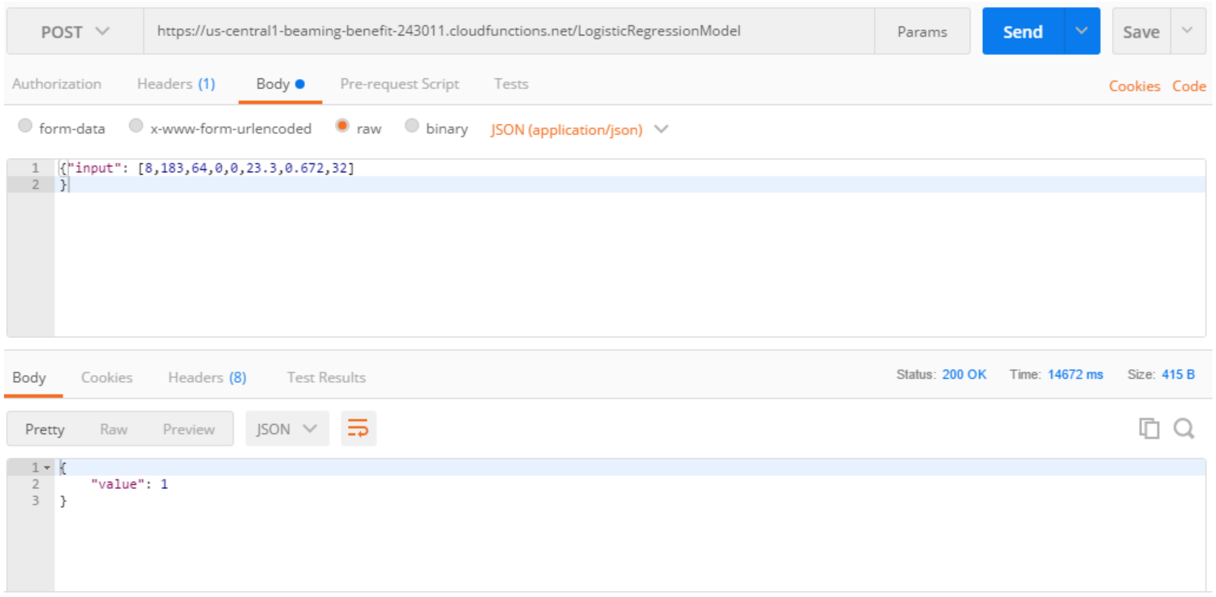
Step 4: Ensure that the function is created successfully



Name	Region	Trigger	Runtime	Memory allocated	Executed function	Last deployed
✓ LogisticRegressionModel	us-central1	HTTP	Python 3.7	256 MB	predict_diabetes	11/4/19, 7:37 PM

Step 5: Invoke the ML model function using Postman. For the input data (features) passed, the ML model returns with a response either '1' – Yes or '2' – No with status "200 OK".

Request Body : `{"input": [8,183,64,0,0,23.3,0.672,32]}`



POST https://us-central1-beaming-benefit-243011.cloudfunctions.net/LogisticRegressionModel

Body: raw, JSON (application/json)

```
{"input": [8,183,64,0,0,23.3,0.672,32]
}
```

Status: 200 OK Time: 14672 ms Size: 415 B

```
{
    "value": 1
}
```

Response : `{"value" : 1}`