Assignment 1

Oussama Zouhadi
26453392
COEN 424, Concordia University
Montréal, Canada
Oussamus_84@hotmail.com

Nirusan Nadarajah
29600094
COEN 424, Concordia University
Montréal, Canada
shanrajah@hotmail.com

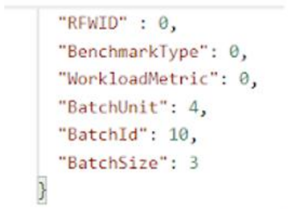
***Abstract*— This report explains the details of Assignment 1 including steps on how to run the applications, design of the data models, methods used for serialization/de-serialization, the data communication of the models, libraries/software packages used to data serialize and screenshots of the results of the assignment task.**

***Keywords*—RESTful API, gPRC, Protobuf, HTTP, HTTP/2, Server, Client**

I. RUNNING THE APPLICATION

Running the application requires a few installations such as .NET Core 3.1, Visual Studio 2019, Postman, Windows PC, and a functional internet connection.

The first method uses the text-based serialization/deserialization method. This is under WorkloadProject. By running the WorkloadProject.sln, the browser shows the localhost address with all the data of DvdTesting as default. To test for different .csv files with different inputs from the Client, use Postman software. The address http://localhost:5000/server or https://workloadproject20201102080146.azurewebsites.net/server (azure cloud hosted link) needs to be set, *GET* should be chosen and in parameters, choose *Body*, *raw*, and then *JSON* as format. Then enter the parameters needed as below. Note that the parameters are just an example.



```
    "RFWID" : 0,
    "BenchmarkType": 0,
    "WorkloadMetric": 0,
    "BatchUnit": 4,
    "BatchId": 10,
    "BatchSize": 3
}
```

Figure 0: Example of client's inputs

After entering the desired parameters, send the request. The output should display on Postman with the correct results based on your input.

In order to test the second method that uses the binary serialization/deserialization, we need to understand that there are two folders GrpcServer and GrpcClient which have the GrpcServer.dll and GrpcClient.dll files that are needed to run this project. Start by opening command prompt and change directory to the location of GrpcServer. Now type *dotnet GrpcServer.dll*. The application should run and display the url and port. Now do the same for GrpcClient by opening a fresh command prompt and change directory to GrpcClient location. Then type *dotnet GrpcClient.dll* to run the application. You should see the application launch. Type https://localhost:5001 and fill in the remaining inputs as per your preference. The output should display the results correctly based off your inputs.

II. DESIGN OF THE DATA MODEL

The data is communicated between the server and the client in two different ways. Protocol buffer method is used for binary serialization/deserialization, JSON files are used as a text-based method.

Under WorkloadProject, The Program.cs file saves the data as objects from the csv files using the *ListOfWorkload.GetWorkloads()* function.

The Models are composed of Batch, ClientRFW, LisfOfWorkload, ServerRFD, and Workload.

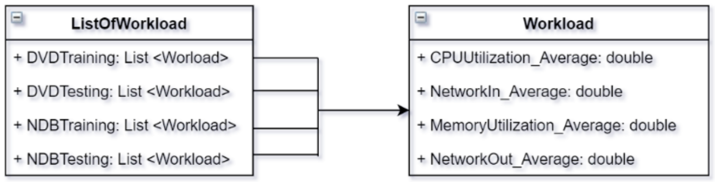

ListOfWorkload
+ DVDTraining: List <Worload>
+ DVDTesting: List <Workload>
+ NDBTraining: List <Workload>
+ NDBTesting: List <Workload>
Workload
+ CPUUtilization_Average: double
+ NetworkIn_Average: double
+ MemoryUtilization_Average: double
+ NetworkOut_Average: double

Figure 1: Data model for ListOfWorkload and Workload

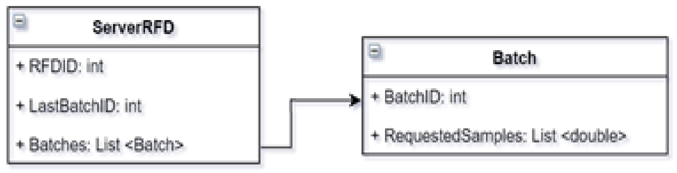


Figure 2: Data model for ServerRFD and Batch

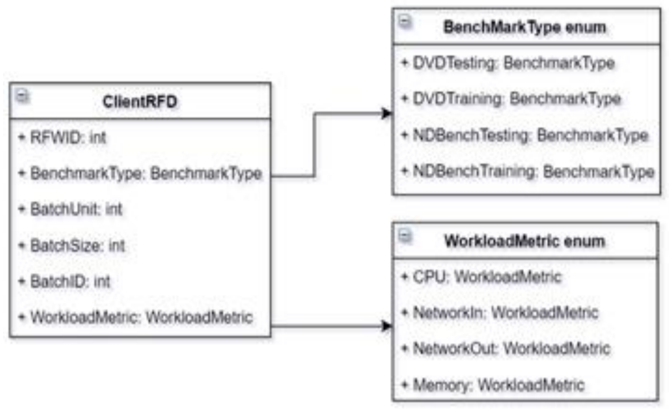

BenchMarkType enum
+ DVDTesting: BenchmarkType
+ DVDTraining: BenchmarkType
+ NDBenchTesting: BenchmarkType
+ NDBenchTraining: BenchmarkType
ClientRFD
+ RFWID: int
+ BenchmarkType: BenchmarkType
+ BatchUnit: int
+ BatchSize: int
+ BatchID: int
+ WorkloadMetric: WorkloadMetric
WorkloadMetric enum
+ CPU: WorkloadMetric
+ NetworkIn: WorkloadMetric
+ NetworkOut: WorkloadMetric
+ Memory: WorkloadMetric

Figure 3: Data model for ClientRFD, BenchmarkType, and WorkloadMetric

Batch: BatchId is the id of the batch. RequestSamples are all the values of the requested samples based of the client inputs.

ClientRFW: RFW Id, BenchmarkType (where it uses enumeration to choose between DVDTesting, DVDTraining, NDBenchTesting , and NDBenchTraining), WorkloadMetric (where it uses enumeration to choose between CPU, NetworkIn, NetworkOut, Memory), BatchUnit, BatchId, BatchSize.

ListOfWorkload: That gets the list of all the Dell DVD testing and training data, and the list of all the NDBench testing and training data.

ServerRFD: RFDID (Response For Data), LastBatchID, and list of Batches.

Workload: It's composed of column names: PUUtilization_Average, NetworkIn_Average, NetworkOut_Average, and MemoryUtilization_Average.

III. Methods Used To Serialisation/DE-Serialisation Data

These data serialization and deserialization methods have been done in two ways: in a text based Json and in binary (gRPC).

The first method uses HTTP and RESTful API along with ASP.NET Core 3.1. HTTP GET is coded in ServerController.cs where all the logic is implemented using variables functions to achieve the task.

The second method uses gRPC and HTTP/2 for binary serialization/deserialization using framework .NET Core 3.1. The protocol buffer is used in this method. gRPC is a high performance Remote Procedure Call (RPC) framework that is defined and structured already in this framework. The WorkloadService.cs contains the logic for processing the client inputs that it received through the built in framework and to reply back with server response. In this case the models are in the work.proto file which similar to the JSON test based method described earlier but uses a proto type model instead. The architecture of model is overall the same.

IV. Data Communication

The project has two methods of communication, text based JSON supported by RESTful API running over HTTP protocol, and binary gRPC protocol using HTTP/2 protocol. The first method is under WorkloadProject where the core server logics are handled at ServerControllers.cs file in the Controller folder. The second method is handled by the GrpcServer and the GrpcClient programs.

T he ServerController.cs file, under WorkloadProject/Controllers, is responsible for managing the data calls and defining the logics of mapping rules of the application. It uses the build-in framework automatically to get the "HTTPGET" from the APIs.

The workload list (*List<Models.Workload> workloadList*) object stores the four columns data fetched from the model *ListOfWorkload.cs*. It reads the benchmarkType the client had chosen.

The method *GetAllWorkloadColumnValues* uses these workload lists and takes all the column values list (*List<double> allColumnValuesList*). It uses the workloadMetric that comes from the client.

The method *GetAllBatches()* calls the list of the column values by verifying the batch units to organize all the batches and returns a *List<Batch> allBatchesList.* For example, if we have 15000 values and the batches unit is 10, the list will have 1500 batches with 10 values in each batch.

Then the program starts a loop to go through all the batches and add them to the *serverRFD* object before returning it. This loop depends on the user input parameters. For instance, if the user wants to start at the *BatchID*: 10, the start index will get the value 10, and ends at the batch size + 10.

In the *GrpcServer*, the Workload service under Services folder, the structure and the logic are quite similar to the workload project except it has a Service instead of a Controller and the data is transmitted as a binary format instead of JSON.

The *gRPC* is an open-source framework made by Google. It allows defining and Requests and Responses for Remote Procedure Calls by using Protocols Buffers (Proto3). Both, the client and the server use the work.proto files for the request and response workloads. The client program.cs asks the user to enter RFWID (request for workload id), the benchmark type, the workload metric, the batch unit, the batch id, and the batch size. These inputs form the workload request object to be sent to the server.

V. LIBRARIES AND PACKAGES

Framework ASP.NET Core using .NET CORE 3.1 with C#. It's an open source used to create server client applications.

gRPC framework is for binary serialization/deserialization. It is built upon HTTP/2 that allows bidirectional communication. It uses protocol buffer for that is less heavy in terms of payload compared to JSON. The bandwidth in this case is optimised. Proto files, for gRPC, have the definition of service using protocol buffers. The *WorkloadService* will write them and allows gRPC to generate the proto code that can be used to build the project. The performance is optimized under this method since the communication is fast and efficient. The binary method is not understandable like JSON and for error handling, it can be hard since it does not use http code.

gRPC framework pros are:
+ Client's and server's CPU utilisation are light.
+ The wall time is short.
+ Machine readable.
+ Language neutral.

The cons are:
- Less browser and languages support.
- Not human readable.
- Error handling.

REST API that uses JSON which is readable, self-contained in a way that everything the consumer needs is within the object. There is no need for synchronization in the JSON for sending or accepting it. Its ability to extend the data within the object is easily processed by the client. Furthermore, inspecting the data is easy. In the other hand, JSON is costly when it comes to serialization and deserialization in a high payload and causes an overhead. It's heavy in terms of data. So, text-based serialization and deserialization can use JSON if the data volume is small, the messages are different among each other.

Pros of REST API framework are:
+ Easy to understand.
+ Availability of frameworks.
+ Built on top of http protocol.
+ Language neutral (Flexibility).

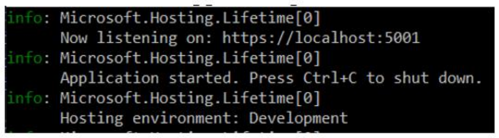
The cons are:
- Expensive in massive data.
- Difficult to maintain state.

VI. RESULTS

This section shows the successful results of running the two methods. WorkloadProject runs on Azure cloud host or localhost and GrpcServer runs on localhost. We test both servers by sending identical requests to see if the responses will be the same.

Binary method:

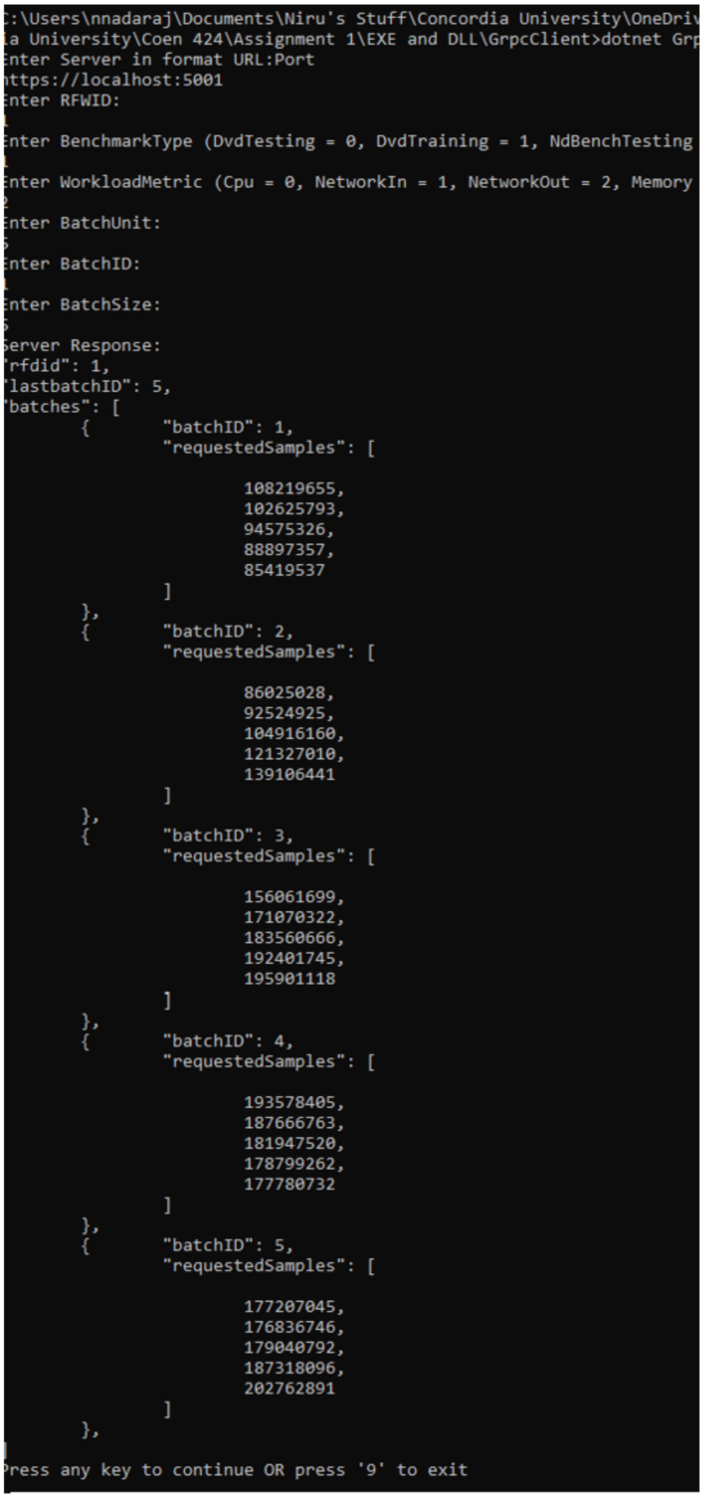
For the binary method that uses gRPC framework, we start by running GrpcServer solution first and then GrpcClient to avoid any complication. The communication happens on port 5001.



```
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
```

Figure 4: The URL and the port for GrpcServer.

In the client side, the user has to enter the server URL along with its port. After that, he/she has to enter the parameters RFWID, BenchmarkType, WorkloadMetric, BatchUnit, BatchId, and BatchSize. The program has some restrictions to prevent the user entering the wrong values.
The following pictures display parameters example and their results:



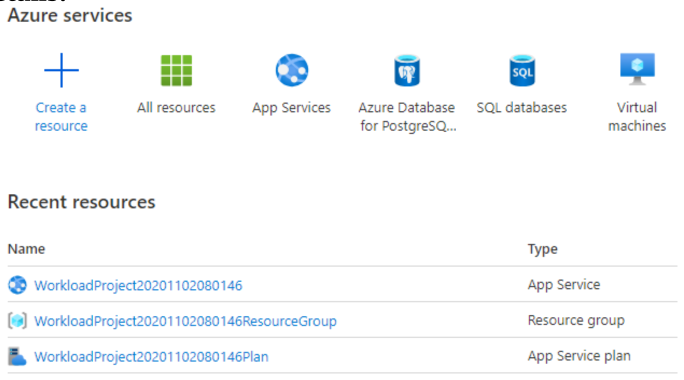
```
:\Users\nnadaraj\Documents\Niru's Stuff\Concordia University\OneDriv
a University\Coen 424\Assignment 1\EXE and DLL\GrpcClient>dotnet Grp
nter Server in format URL:Port
ttps://localhost:5001
nter RFWID:
1
nter BenchmarkType (DvdTesting = 0, DvdTraining = 1, NdBenchTesting
1
nter WorkloadMetric (Cpu = 0, NetworkIn = 1, NetworkOut = 2, Memory
2
nter BatchUnit:
5
nter BatchID:
1
nter BatchSize:
5
erver Response:
rfdid": 1,
lastbatchID": 5,
batches": [
        {       "batchID": 1,
                "requestedSamples": [

                        108219655,
                        102625793,
                        94575326,
                        88897357,
                        85419537
                ]
        },
        {       "batchID": 2,
                "requestedSamples": [

                        86025028,
                        92524925,
                        104916160,
                        121327010,
                        139106441
                ]
        },
        {       "batchID": 3,
                "requestedSamples": [

                        156061699,
                        171070322,
                        183560666,
                        192401745,
                        195901118
                ]
        },
        {       "batchID": 4,
                "requestedSamples": [

                        193578405,
                        187666763,
                        181947520,
                        178799262,
                        177780732
                ]
        },
        {       "batchID": 5,
                "requestedSamples": [

                        177207045,
                        176836746,
                        179040792,
                        187318096,
                        202762891
                ]
        },

ress any key to continue OR press '9' to exit
```

Figure 5: User inputs and the results.

Text-based method:
The second method uses JSON to communicate data. We start the WorkloadProject.sln at URL:

: https://workloadproject20201102080146.azurewebsites.net

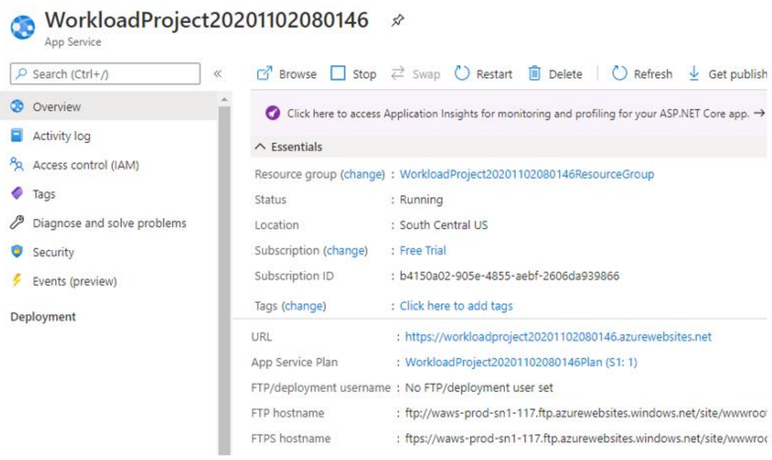
The following screenshots display the cloud host details:



Recent resources

Name	Type
WorkloadProject20201102080146	App Service
WorkloadProject20201102080146ResourceGroup	Resource group
WorkloadProject20201102080146Plan	App Service plan

Picture 6: Azure Cloud host_1



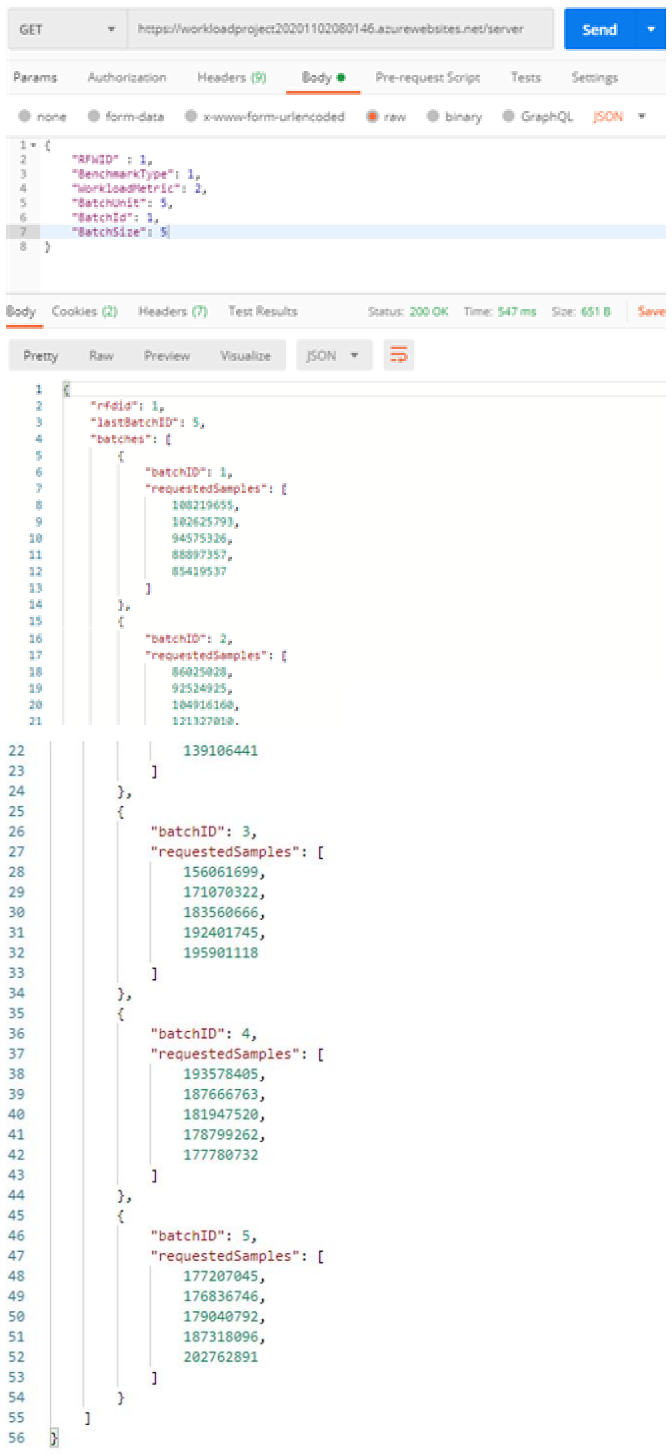
Click here to access Application Insights for monitoring and profiling for your ASP.NET Core app. →

Essentials

Resource group (change)	: WorkloadProject20201102080146ResourceGroup
Status	: Running
Location	: South Central US
Subscription (change)	: Free Trial
Subscription ID	: b4150a02-905e-4855-aebf-2606da939866
Tags (change)	: Click here to add tags
URL	: https://workloadproject20201102080146.azurewebsites.net
App Service Plan	: WorkloadProject20201102080146Plan (S1: 1)
FTP/deployment username	: No FTP/deployment user set
FTP hostname	: ftp://waws-prod-sn1-117.ftp.azurewebsites.windows.net/site/wwwroo
FTPS hostname	: ftps://waws-prod-sn1-117.ftp.azurewebsites.windows.net/site/wwwro

Picture 7: Azure Cloud host_2

Postman software plays the role of the client in this scenario. The GET method sends requests to the server hosted in Azure cloud at the address shown in the following screenshot.



GET https://workloadproject20201102080146.azurewebsites.net/server Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "RFWID" : 1,
    "BenchmarkType": 1,
    "WorkloadMetric": 2,
    "BatchUnit": 5,
    "BatchId": 1,
    "BatchSize": 5
}
```

Body Cookies (2) Headers (7) Test Results Status: 200 OK Time: 547 ms Size: 651 B Save

Pretty Raw Preview Visualize JSON

```
{
    "rfdid": 1,
    "lastBatchID": 5,
    "batches": [
        {
            "batchID": 1,
            "requestedSamples": [
                108219655,
                102625793,
                94575326,
                88897357,
                85419537
            ]
        },
        {
            "batchID": 2,
            "requestedSamples": [
                86025028,
                92524925,
                104916160,
                121327010,
                139106441
            ]
        },
        {
            "batchID": 3,
            "requestedSamples": [
                156061699,
                171070322,
                183560666,
                192401745,
                195901118
            ]
        },
        {
            "batchID": 4,
            "requestedSamples": [
                193578405,
                187666763,
                181947520,
                178799262,
                177780732
            ]
        },
        {
            "batchID": 5,
            "requestedSamples": [
                177207045,
                176836746,
                179040792,
                187318096,
                202762891
            ]
        }
    ]
}
```

Picture 8: The request and the response

In both cases the server returns the same results successfully.