Geo-Context aware routing using Serverless Computing - Configuration Manual

MSc Research Project
MSc in Cloud Computing

Akash Verma
Student ID: X21128863

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	…….…………… Akash Verma ……………………………………………….
Student ID:	………………………X21128863………………………………………………
Programme:	…………………MSc in Cloud Computing… **Year:** 2022
Module:	………………………Research Project……………………….…………
Supervisor:	……………………Vikas Sahni…………………………………….………
Submission Due Date:	…………………… Dec 15, 2022………………………………………………
Project Title:	Geo-Context aware routing using Serverless Computing.
Word Count:	………………1687…………………… **Page Count**………5……………..……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: [illegible]………………………………………………………………………………………

Date: ………………… Dec 12, 2022……………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Geo-Context aware routing using Serverless Computing - Configuration Manual

Akash Verma
Student ID: X21128863

1 Tools Required

1. SUMO GUI, 1.14.1[1]
2. Open Steet Map, V2.23.2 or later based on current release deployed [2]
3. Polyline tool, latest available on URL[3]
4. dotnet core runtime, 6.0.11[4]
5. Azure subscription (for cloud TPU, geolocator service, and cosmos DB) [5]
6. Visual Studio 2022 community edition (optional, used for development and docker build)[6]
7. Docker 4.6.1 or later

2 Basic configurations

2.1 On Board Unit:

The OBU in the vehicle's is supposed to do all the data transfer related job, an azure function is programmed to act as an OBU. The two configurations that an OBU needs are explained as follows:

2.1.1 local.settings.json

The local.settings.json file is the metadata about the vehicle and is tied to the telemetry service provider. To configure OBU, application settings are needed to be supplied in local.settings.json file, located in "~\ProjectArtifact\OBU\OBU<i>", where 'i' is the OBU number, there can be multiple OBU in the system, which corresponds to the vehicle. Below is the parameter sample:

```
 "VehicleId": "1",
"VehicleType": "car",
"serviceUrl": "http://xyz.com/"
```

Vehicle Id: It is the identity of the vehicle, which can be any value, that will be referenced in the SUMO trace file used by the vehicle for the simulated run (2.1.2, point 5).

[1] https://www.eclipse.org/sumo
[2] https://www.openstreetmap.org/#map=9/53.6162/-6.2814, *OpenStreetMap*. (n.d.). OpenStreetMap. Retrieved December 12, 2022, from https://www.openstreetmap.org/copyright, https://github.com/openstreetmap/iD/releases
[3] https://www.keene.edu/campus/maps/tool/
[4] https://download.visualstudio.microsoft.com/download/pr/05dbe2dd-429c-4e4a-8080-9fe3027cd31b/76461b8691ada05d098efd15933bcbbd/aspnetcore-runtime-6.0.11-win-x64.zip
[5] https://portal.azure.com
[6] https://visualstudio.microsoft.com/vs/community/

serviceUrl: This is the URL for the Geo-Locator service hosted on azure (section 2.3 covers the details).

VehicleType: It is an optional configuration, can be left it as it is.

2.1.2 The Sumo Trace File

The sumo trace file contains the random trip generated using SUMO tool, the vehicle is simulated to run on the path based on this file.

1. Export OSM file
    1. Browse to the Open street map in the preferred browser.
    2. Click on export from the top menu.
    3. Click on "manually select different area" from the left section of the page and select the area using the selection polygon. Click export.
2. Create test.net.xml file for route simulation as explained below
    a. *C:\Program Files (x86)\Eclipse\Sumo\bin>netconvert --osm-files <"path to .osm file"> -o <"path to output test.net.xml">*

        Example:

        C:\Program Files (x86)\Eclipse\Sumo\bin>netconvert --osm-files "F:\NCI_Student\Sem 3\Research\MAPVIEW\map.osm" -o "F:\NCI_Student\Sem 3\Research\MAPVIEW\test.net.xml"

    b. Go to root folder:

        Example: cd F:\NCI_Student\Sem 3\Research\MAPVIEW

    c. Using Random trip, create route files and the trips.

        Python "C:\Program Files (x86)\Eclipse\Sumo\tools\randomTrips.py" -n <path to test.netfile> -r <output route file name> -e <no of vehicle> -l

        Example: *Python "C:\Program Files (x86)\Eclipse\Sumo\tools\randomTrips.py" -n test.net.xml -r test.rou.xml -e 1 -l*

3. Create SUMO config file as "test.sumo.cfg" below:

```
<configuration>
<input>
<net-file value="test.net.xml"/>
<route-files value="test.rou.xml"/>
</input>
<time>
<begin value = "0"/>
<end value = "20000"/>
</time>
</configuration>
```

4. Transform coordinate-based file to geolocation file:
    sumo -c test.sumo.cfg --fcd-output trace1.xml --fcd-output.geo
5. Update the OBU local.setting.json with the vehicle id generated in sumo trace file.
6. While using multiple vehicle simulations, create copies of the OBU folder, and copy the sumo file obtained by setting number of vehicle (-e) in step 2c above to each of the OBU. Update only the ID in the settings file with the VehicleId (step 2.1.1) of that specific value, which is played for that OBU.

Or

Simply use the existing trace and setting files in the artifact for OBU "~\Artifact\OBU\OBU0\sumoTrace.xml"

2.2 Telemetry Processing Unit

There are two types of TPUs, local and cloud. For local TPU, local.settings.json file is used for configuration, and azure configuration for cloud.
The cloud TPU can be deployed using the visual studio[7], and the configuration is set under configurations (deployed function > configuration (under settings tab)) in azure. Below are the config settings that are supplied to the TPU.

```
"device_location": "-6.258603 53.359184",
"tpuserviceUrl": "http://localhost:82/",
 "served_location": "{'ServedPolygon':[[[-6.2845917,53.4290679],[-6.1767884,53.4229273],[-6.1692353,53.4536215],[-6.2804718,53.4618029],[-6.2845917,53.4290679]]],'TelemetryEndPoint':'sl1', 'deviceLocation':{'longitude':0,'latitude':0}, 'deviceId':'sl1', 'type':'fog'}"
```

1. device_location: It is the location where the TPU is placed, this is an optional parameter.
2. tpuserviceurl: The base URL for the geolocator service.
3. served location: This is the coverage of the TPU. It needs to be set for each TPU separately. The polyline tool is used to set up the polygon following the below steps:
   a. Open the polyline tool on the browser. (https://www.keene.edu/campus/maps/tool/)
   b. Navigate in the map to area under study (research use m1 motorway).
   c. Figure out the polygons in the route simulated using SUMO (2.1.2).
   d. A polygon can be created in the polyline tool using a right-click on the map.
   e. Copy the coordinates and substitute json in the 'ServedPolygon' section of the served location setting.
   f. Note: The cloud TPU has empty array list as served polygon ({'ServedPolygon':[[]],'TelemetryEndPoint':'cl', 'deviceLocation':{'longitude':0,'latitude':0},'deviceId':'cl','type':'cloud' }), which denotes all area.
4. served_location.deviceId : Identifier for TPU
5. served_location.type: Use 'cloud' for cloud TPU, 'fog' for local TPU.

Or

Simply use the existing TPU configuration and setting files in the artifact for TPUs from "Artifact.zip\Artifact\TPU\sl<i>"
The serving polygon used for the setup of local TPU is depicted in table 1.

Table 1: serving polygon for TPU

TPU Name	**Served area(figure1)**	**Served Polygon**
TPU1	S1	[-6.2845917,53.4290679],[-6.1767884,53.4229273],[-6.1692353,53.4536215],[-6.2804718,53.4618029],[-6.2845917,53.4290679]

[7] https://learn.microsoft.com/en-us/azure/azure-functions/functions-develop-vs?tabs=in-process

TPU2	S2	[-6.1761017,53.4752987],[-6.1822815,53.4916515],[-6.2372132,53.4875639],[-6.2337799,53.4708006],[-6.1761017,53.4752987]
TPU3	S3	[-6.1740418,53.5227045],[-6.1781616,53.5667933],[-6.2447663,53.5598565],[-6.2344666,53.5182114],[-6.1740418,53.5227045]
TPU4	S4	[-6.1733551,53.4544397],[-6.1761017,53.4781609],[-6.2365265,53.4757076],[-6.2310334,53.4528033],[-6.1733551,53.4544397]
TPU5	S5	[-6.2461396,53.5598565],[-6.1905213,53.5623049],[-6.2063141,53.6071677],[-6.2523194,53.6026836],[-6.2461396,53.5598565]
TPU6	S6	[-6.2042542,53.6026836],[-6.2145539,53.6332479],[-6.2687989,53.6279516],[-6.2550659,53.5977913],[-6.2042542,53.6026836]

2.3 The Geo-Locator Service:

1. Deploy the geolocator service to the cloud from visual studio similarly like cloud TPU and update the settings in the OBU and TPU with the base URL, that is obtained after deploying geo-locator service.
2. Configure the cosmosurl and cosmosKey in the geolocator service app.settings.json file or in azure when deployed about the endpoint of cosmos DB.
   "cosmosuri": "https://xyz.documents.azure.com/",
   "cosmosKey": "cosmos key"

2.4 The Cosmos Database:

The Azure Cosmos database account is needed for setting up the database. The application is programmed to create the containers for the database if they do not exist. Only geolocator service needs to be supplied with the database URL and key.

3 Running the setup

Once all the configuration is done, make sure the geo locator service is up and running after the deployment. Running and testing the setup require two components to be kicked off, the TPUs and OBUs.

3.1 Running local TPUs

There are two ways to run the TPUs, one is using the executables in shared, non-isolated mode, with dot net run time and second one is using docker for visual studio. The artifact folder contains published project. Use the following steps to run the TPUs in non-isolated mode:

1. Make sure dotnet core runtime is installed on the machine.
2. Each TPU can be run independently using the function start command from the root of project placed in "~\Artifact\sl<i>\".
   func start –port <portno>.
   Or
   Use the batch file to fire up all the functions at once using batch file located at "~\Artifact\TPU\start.bat".
3. A new TPU can be added at any point, provided the configuration is given as explained in section 2.2.

4. Now run the function to initiate the registration process of TPU with geolocator service using " ~\Artifact \TPU\run.ps1" .

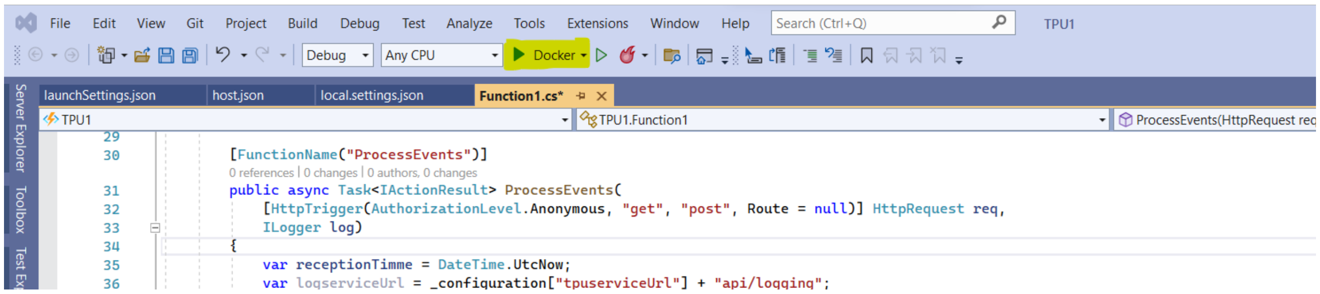
   Or

   simply run *"wget <tpuurl>/api/StartProcessor"*. Note: The URL in the console is written by the dotnet run time after step 2.

The second way is through docker containerization using visual studio. The project is docker ready project, just open each solution and build. It will create a docker image. Create new instances of the same image with a different configuration of TPU and run.

Or

To run in isolated mode, simply use visual studio to run each of the project placed at "~\Artifact\Code\TPUs\TPU<i>" using docker.


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
TPU1
Debug
Any CPU
Docker
Server Explorer
Toolbox
Test Ex
launchSettings.json
host.json
local.settings.json
Function1.cs*
TPU1
TPU1.Function1
ProcessEvents(HttpRequest req
0 references | 0 changes | 0 authors, 0 changes

```
[FunctionName("ProcessEvents")]
public async Task<IActionResult> ProcessEvents(
    [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)] HttpRequest req,
    ILogger log)
{
    var receptionTimme = DateTime.UtcNow;
    var logserviceUrl = _configuration["tpuserviceUrl"] + "api/logging";
```

Figure: using docker to run azure function locally

3.2 Running OBU/OBUs

The final step is running the OBU and observing the results, using following steps:

1. Run the OBU from the root of the OBU (where local.settings.json is placed) using the command:
   func start –port <portno>.

4 References: