Mobiliya is now a QuEST Global Company

FLEET MANAGEMENT SOLUTION

Deployment Guide

Revision History	
Version No.	6.0
Authorized By	Sagar Shah

Contents

1. Introduction

1.1. About this Guide

The purpose of this deployment guide is to assist developers in understanding and setting up the Mobiliya Fleet Management solution. It is a step-by-step walkthrough of the setup process of the solution.

1.2. About Mobiliya Fleet Management Solution

The Fleet Management solution consists of following components:

- Vehicle (Truck/Car)
- OBD/J1939 Dongle
- Mobile Application
- Azure Cloud Application

The solution supports commercial vehicles and cars supporting J1939 and OBD protocols respectively.

A user can connect a Dongle to a vehicle which can retrieve vehicle diagnostic information and forward this information to a mobile application over Bluetooth. The mobile application will further forward this information to the cloud. The cloud application then performs detailed analysis of a given data and provides different reports to different stakeholders/users.

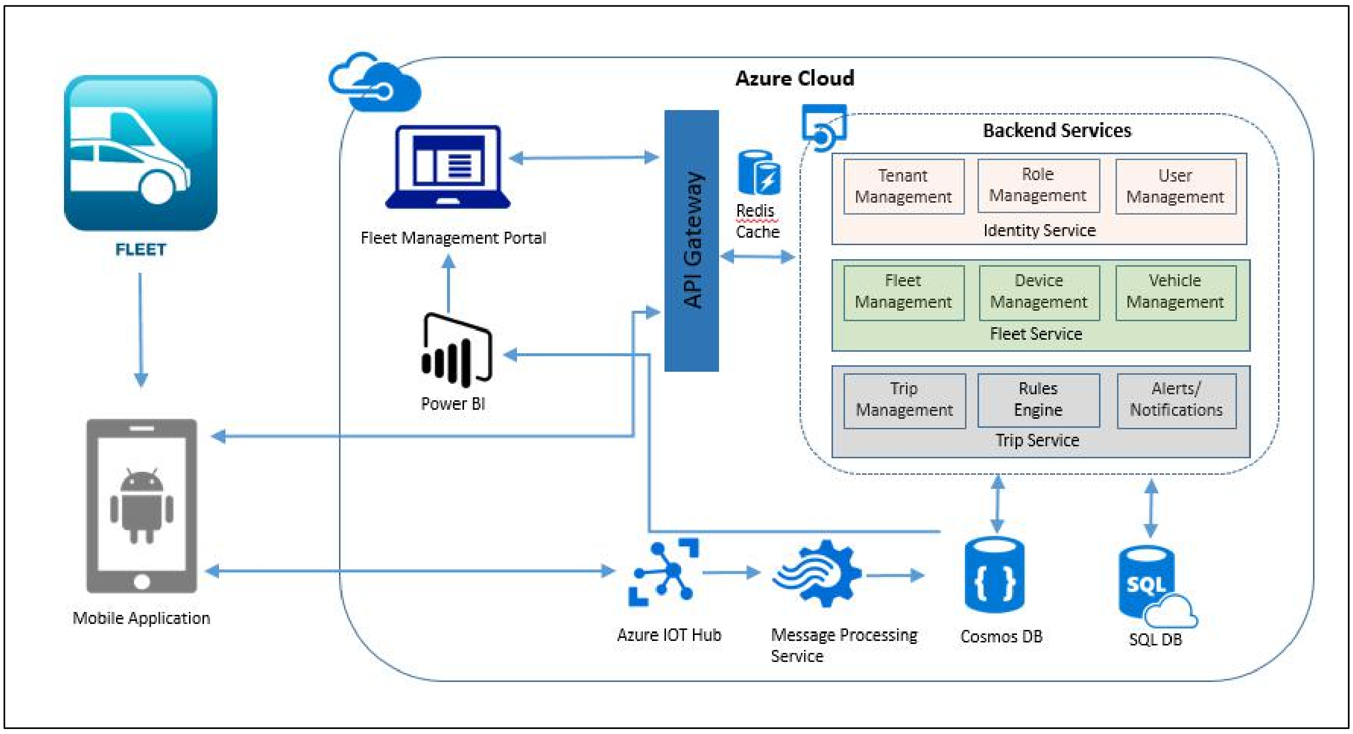
There are 3 types of user actively involved in this system.

- Tenant Admin
- Fleet Admin
- Driver

2. System Architecture

The architecture comprises of two main components:

- Mobile Application
- Cloud Application



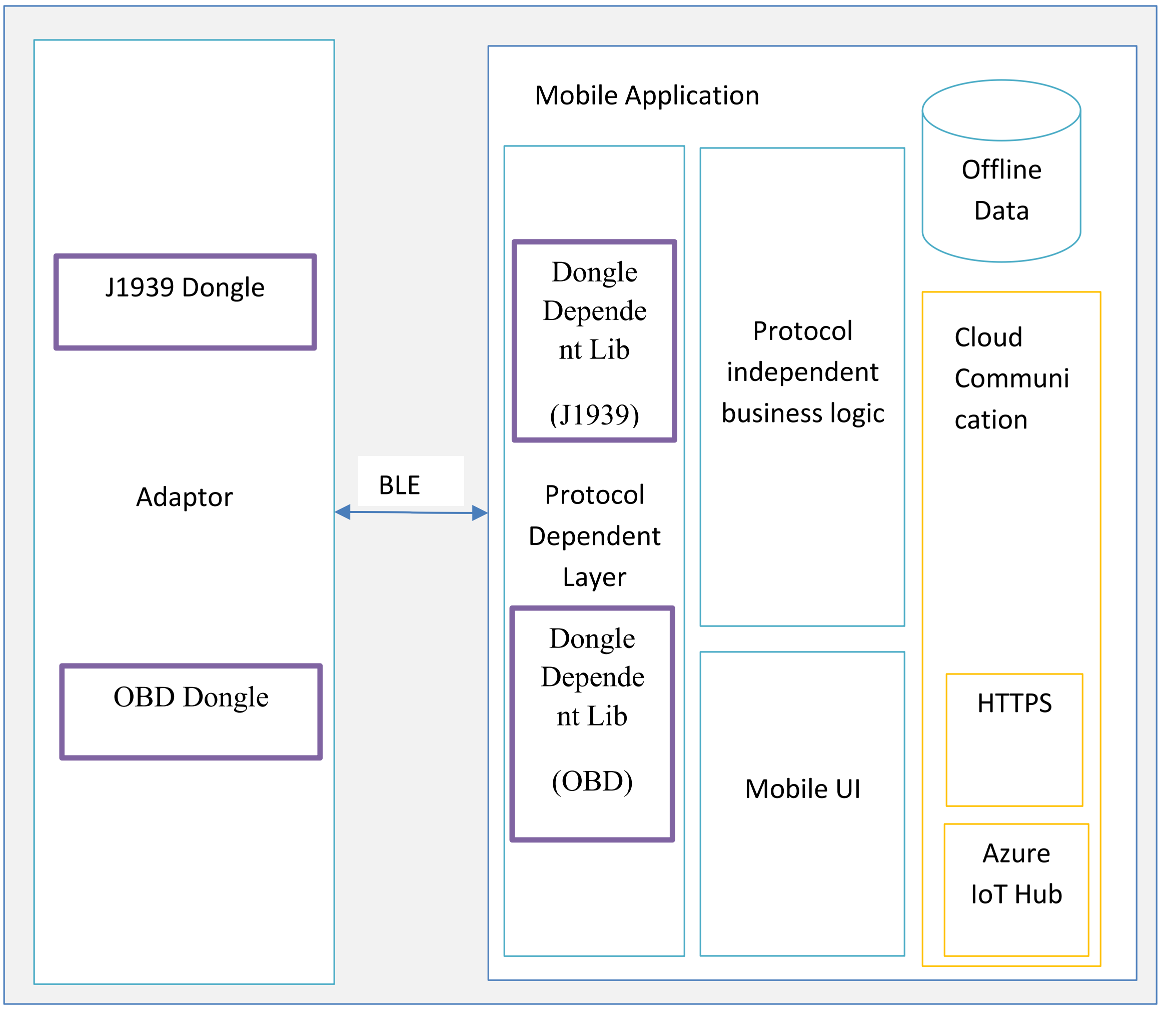
2.1. Mobile App Components

The mobile application has been developed using Android Native Platform. It will support multi-layer architecture for easy protocol integration.

The mobile application communicates with the Dongle using Bluetooth. The mobile application logic is divided in following layers:

- Protocol Dependent Layer
- Protocol independent layer i.e. Business Logic Layer
- Cloud Communication Layer
- Mobile UI
- Internal Offline Storage

The illustration given below provides internal details of the architecture:



- The Dongle will communicate with Protocol dependent layer. This will be a configurable interface to support more than 1 dongle. Protocol dependent layer shall encapsulate protocol/Dongle specific implementation details.

- Protocol Independent business logic is an intermediate layer which will communicate with Protocol dependent layer, Local Database, Cloud API and Mobile UI.

- Maintaining local database to track trips and vehicle information. This data will be synced with server on Azure.

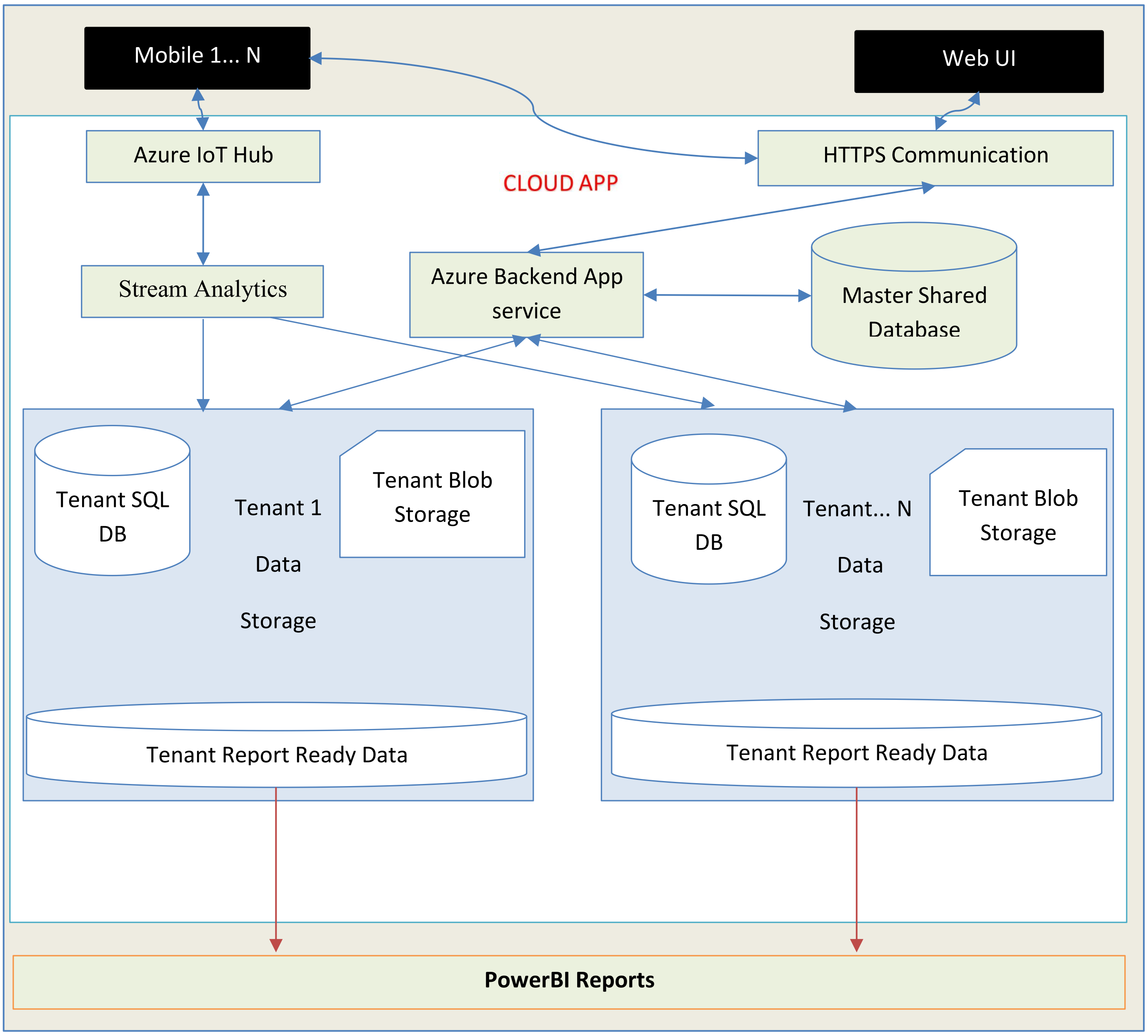
Mobile application has been developed using:

- Android Native Platform
- SQLite Database
- Azure IoT client

2.2. Cloud Application Components

The cloud application uses shared multi-tenant architecture; where tenant details and authorization details (roles and privileges) are stored in a master database while tenant- specific information is stored inside separate tenant-specific databases.

The illustration given below provides a detailed view of the different components used in the cloud application:



There are two active users of cloud application

1. Web application interface
2. Mobile application

The web application uses HTTPS based communication channels and invokes different REST APIs provided by Azure App Service (backend services). The Azure App Service controls the backend business logic.

The web application (Fleet Management Portal) is also deployed as Azure App Service and has been implemented using AngularJS and Bootstrap technologies. It is a responsive web application and can be viewed on different desktop/laptop resolutions.

The Mobile application communicates with Backend Azure Service using two different ways:

1. Azure IoT Hub
2. HTTPS REST API

The Azure IoT hub is used to receive dongle/device data along with Mobile GPS location and forward it to backend Azure App service after every one minute ("Send Duration" can be configured). The Azure backend service internally detects a tenant and connects with it and stores the tenant specific data in a tenant database.

The tenant-specific report-ready Azure SQL/Cosmos data is processed by PowerBI reports.

3. Deploying Resources using ARMTemplate on Azure

The following instructions would help the user to deploy bare minimum Azure resources required to get the solution up and running.

Pre-requisites

The following pre-requisites are essential to get the Mobiliya Fleet Management System up and running on a personal account:

1. Create packages for back end services using Visual Studio with support for Angular and Node.js

- Clone the source code for(Identity/Fleet/Trip service) from github .
- In Visual Studio Solution Explorer, right-click the project and choose Publish.
- In the publish target dialog box, choose Custom Option.
- Enter a profile name.
- In the Web Deploy Package box, choose Web Deploy Package option and enter package location. Visual Studio builds the project and publishes it to the specified package location.
- Select Publish Option and wait for some time.
- Once you get a notification of successful creation of packages from Visual Studio, please verify that the packages have been created in the specified folder.

2. Create package for front end(accelerator portal)

- Clone the source code for (accelerator portal) from github.
- Open command prompt/Terminal in the source directory, you have cloned earlier.
- Install required dependencies and packages for angular project to run.
    1. To install dependancies run ***npm install*** in command prompt.
    2. You also need to install angular-cli using command ***npm install –g angular-cli.***
    3. Install latest angular cli dependency using command ***npm install –g @angular/cli@latest.***
- Once angular cli installed successfully, build the source code using command ***ng build.*** When application builds successfully, ***"dist"*** folder will be generated in the source directory.
- Copy *web.config* file to *dist* folder.
- Go to *dist* folder, select all the files. And then compress all the files into *.zipped* file.
- Change archieve name to user understandable name.

3. Upload packages (which are created using Visual Studio) to a blob container in an Azure Storage account which would be used to deploy the source on Azure portal as a web application

- If the storage account is already created, use existing storage account or Create a new Storage account by clicking > Storage > Storage account - blob, file, table, queue on Azure portal.

- Once storage account is created, navigate to the newly created storage account, and create a new Blob container. Under Blob Service, select Containers, and then + Container.
- Enter a name for container and select public access level and click on OK.
- In Azure Storage Explorer, select the newly created container, and then select Upload > Upload Files to upload the app source package that you created in Step 1.
- Once packages are uploaded successfully,use that Package URL for further deployment.

4. Active subscription

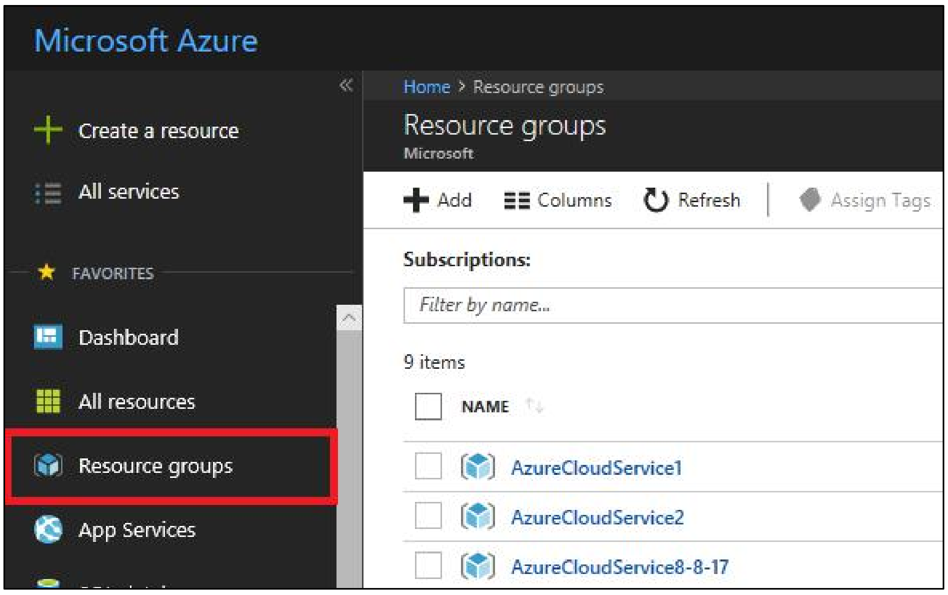
5. **Link to the ARM** (Azure Resource Manager) template that would be used as a script to install the Services
https://portal.azure.com/#create/Microsoft.Template/uri/https%3A%2F%2Fmfleetpackagestorage.blob.core.windows.net%2Foneclickdeployment%2Fazuredeploy.json

6. Follow the **Azure Resource Manager Naming Convention**, (https://docs.microsoft.com/en-us/azure/architecture/best-practices/naming-conventions)

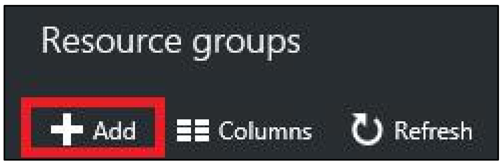
3.1. Step 1: Create Resource Group (Refer only if not created)

Once the pre-requisites are in place, the next step would be the installation of the ARM script. The steps for installing the ARM script are given below:

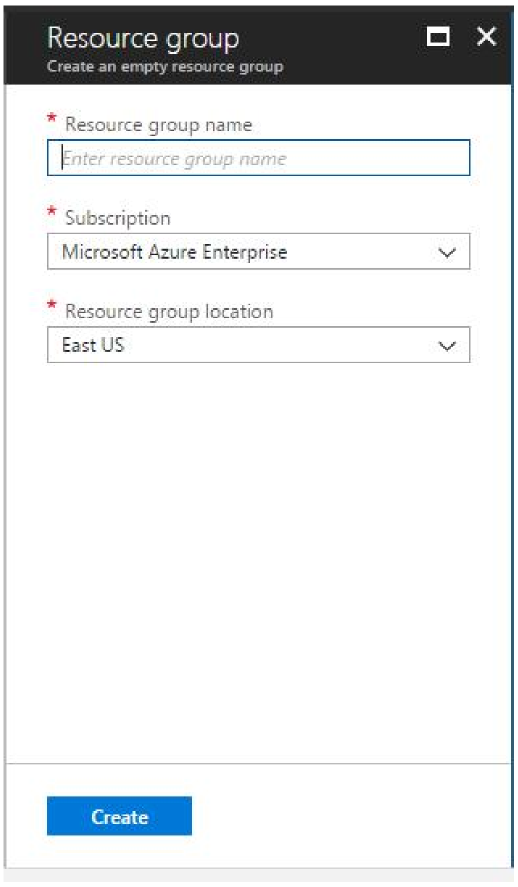
- If the resource group is already created, use existing resource group and skip to next step.
- Login to Azure portal (https://portal.azure.com) and select the appropriate subscription if it is not selected by default. Create a resource group that serves as the container for the deployed resources.
- Click on **Resource Groups** from the left menu to create a resource group.



- Click on **+Add** under **Resource Groups**



Provide a name and location for the new resource group. Click on **Create**.



* Resource group name
Enter resource group name

* Subscription
Microsoft Azure Enterprise

* Resource group location
East US

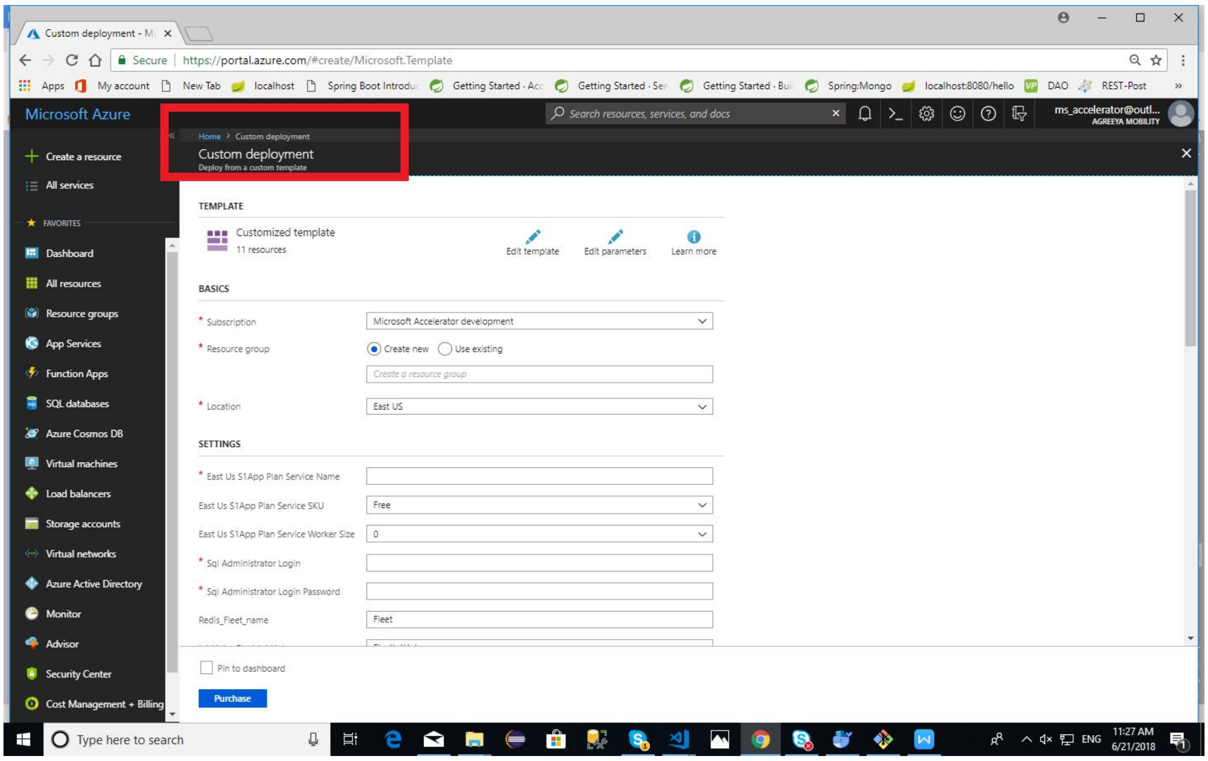
3.2. Step 2: Deploy Resources with Resource Manager Template

To deploy the template that defines the resources to the resource group:

- Visit the below link to the ARM(Azure Resource Manager) Template.
https://portal.azure.com/#create/Microsoft.Template/uri/https%3A%2F%2Fmfleetpackagestorage.blob.core.windows.net%2Foneclickdeployment%2Fazuredeploy.json

- Once the link is loaded. A new form appears which accepts some values before the actual deployment.

Fill in all the required details using the following guidelines:
* Use of small case characters is preferred.
* Use of special symbols is not allowed
* Use the **"I" image symbol** as a guidance for filling data
* Give special preference to fields marked with **"Has to be unique"**



Home > Custom deployment

Custom deployment
Deploy from a custom template

TEMPLATE

Customized template
11 resources

BASICS

* Subscription	Microsoft Accelerator development
* Resource group	[✓ Create new] Use existing
	Create a resource group
* Location	East US

SETTINGS

* East Us S1App Plan Service Name	
East Us S1App Plan Service SKU	Free
East Us S1App Plan Service Worker Size	0
* Sql Administrator Login	
* Sql Administrator Login Password	
Redis_Fleet_name	Fleet

Pin to dashboard

Notes:

1. For "PowerBIUserName" and "PowerBIPassword": use own PowerBI credentials.For "PowerBIClientId" and "PowerBIClientSecret": Please keep it as is, if you don't have powerbi setup ready.

2. For identityServicePackageURL ,fleetServicePackageURL, tripServicePackageURL, acceleratorPoratlPackageURL parameters use the Package URL from blob container in an Azure Storage account , where packages have been uploaded as explained in(prerequisite step2)

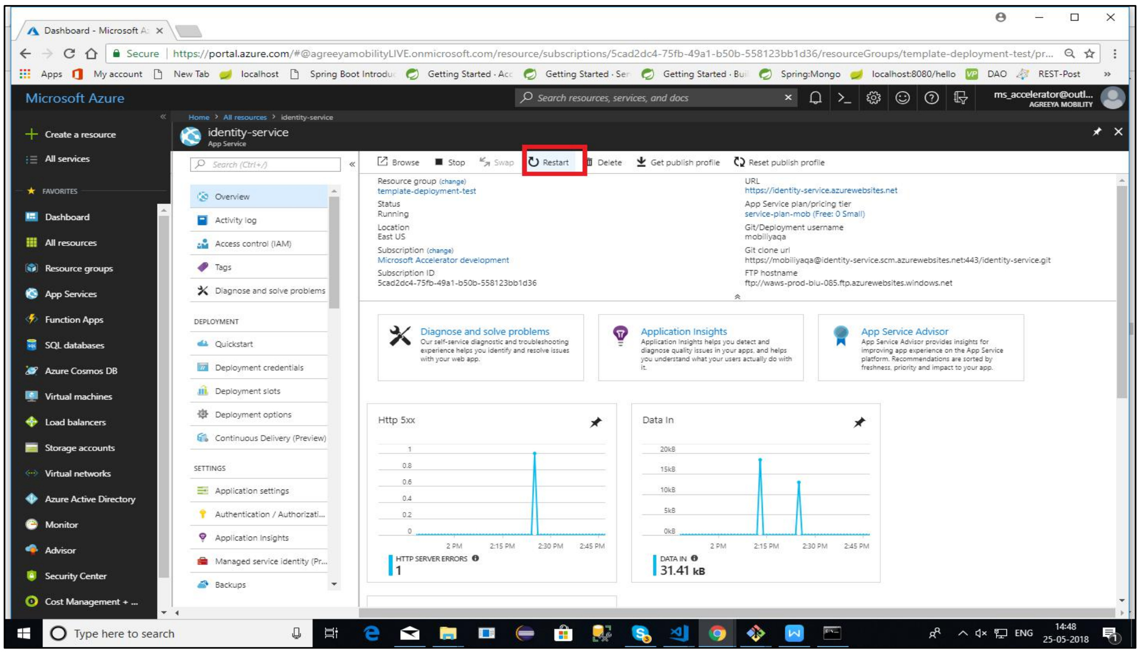
* Once, all the fields have been filled, accept the licensing terms and click on **Purchase**
* The deployment would take some time; you can have a cup of coffee till then
* Once you get a notification of successful deployment of resources from Azure, please verify that the resource types mentioned below have been created in the Azure portal:

1. App Services (app services for identity/fleet/trip/accelerator-portal).
2. Azure cosmos db account.
3. Redis cache
4. IoT Hub
5. SQL server
6. SQL database
7. Azure Blob Storage

- Names of the resources will be those that users have provided in the form.

3.3. Step 3: Restart Application

- On Azure portal,after successful deployment of resources restart the web application (app services for identity/fleet/trip/accelerator-portal).



Resource group (change)
template-deployment-test
Status
Running
Location
East US
Subscription (change)
Microsoft Accelerator development
Subscription ID
5cad2dc4-75fb-49a1-b50b-558123bb1d36

URL
https://identity-service.azurewebsites.net
App Service plan/pricing tier
service-plan-mob (Free: 0 Small)
Git/Deployment username
mobiliyaqa
Git clone url
https://mobiliyaqa@identity-service.scm.azurewebsites.net:443/identity-service.git
FTP hostname
ftp://waws-prod-blu-085.ftp.azurewebsites.windows.net

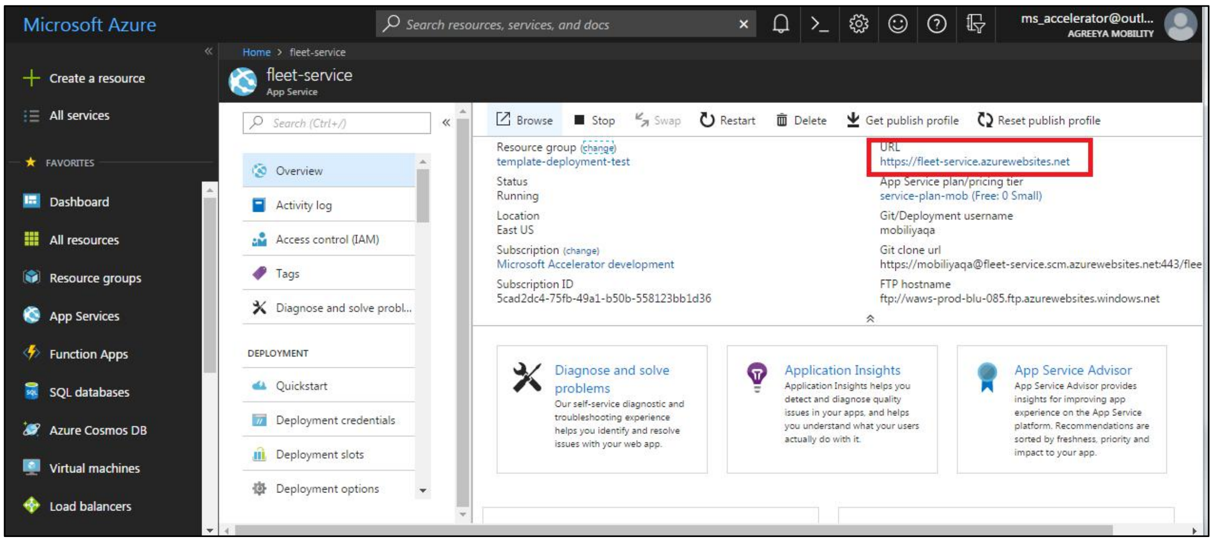
Diagnose and solve problems
Our self-service diagnostic and troubleshooting experience helps you identify and resolve issues with your web app.

Application Insights
Application Insights helps you detect and diagnose quality issues in your apps, and helps you understand what your users actually do with it.

App Service Advisor
App Service Advisor provides insights for improving app experience on the App Service platform. Recommendations are sorted by freshness, priority and impact to your app.

3.4. Step 4: Verify Service is working

To test your application, open **URL**. It will show a message indicating your service is up and running.



Resource group (change)
template-deployment-test
Status
Running
Location
East US
Subscription (change)
Microsoft Accelerator development
Subscription ID
5cad2dc4-75fb-49a1-b50b-558123bb1d36

URL
https://fleet-service.azurewebsites.net
App Service plan/pricing tier
service-plan-mob (Free: 0 Small)
Git/Deployment username
mobiliyaqa
Git clone url
https://mobiliyaqa@fleet-service.scm.azurewebsites.net:443/flee
FTP hostname
ftp://waws-prod-blu-085.ftp.azurewebsites.windows.net

Diagnose and solve problems
Our self-service diagnostic and troubleshooting experience helps you identify and resolve issues with your web app.

Application Insights
Application Insights helps you detect and diagnose quality issues in your apps, and helps you understand what your users actually do with it.

App Service Advisor
App Service Advisor provides insights for improving app experience on the App Service platform. Recommendations are sorted by freshness, priority and impact to your app.

3.5. Powerbi setup

User has to setup a PowerBI environment using the PowerBI Deployment Guide for generating reports which will be shown on dashboard/reports section of front end. (Visit GitHub URL given below for Powerbi Deployment Guide).
(https://github.com/MobiliyaTechnologies/MobiliyaFleetPowerBI)

Follow steps 1 to 7 from file **(Mobiliya_Fleet_PowerBIDeployment_Version_1.x.x.docx)**, which will help to get below parameters which are required while deploying ARM template.

- Client Id
- Client Secret

**It is recommended that users save these parameters for ready reference*

The PowerBI account credentials are configured in the web portal and backend service once you followed above steps.