



FLEET MANAGEMENT SOLUTION

Deployment Guide

Revision History	
Version No.	6.0
Authorized By	Sagar Shah

© 2018 Mobiliya Technologies

Strictly Private and Confidential. No part of this document should be reproduced or distributed without the prior permission of Mobiliya Technologies.

Contents

1. Introduction	5
1.1. About this Guide	5
1.2. About Mobiliya Fleet Management Solution.....	5
2. System Architecture	6
2.1. Mobile App Components	6
2.2. Cloud Application Components	8
3. Deploying Resources using ARMTemplate on Azure.....	10
Pre-requisites	10
3.1. Step 1: Clone Repository	10
3.2. Step 2: Create Resource Group (Refer only if not created)	10
3.3. Step 3: Create Resources Using Template	12
4. Deployment of Backend Services on Azure	15
Pre-requisites	15
4.1. Step 1: Select Deployment Option.....	15
4.2. Step 2: Set Deployment Source- Local Git	16
4.3. Step 3: Set Deployment Credentials	16
4.4. Step 4: Copy Git URL	17
4.5. Step 5: Repeat Steps for Other Services	17
4.6. Step 6: Deploy Actual Code on Azure	17
4.7. Step 7: Restart Application	18
4.8. Step 8: Verify Service is working	18
5. Deployment of Web Portal on Azure	19
5.1. Web Portal Setup and Configuration changes	19
5.1.1. Pre-requisites	19
5.1.2. Clone The Repository	19
5.1.3. Deploy The Application	19
5.1.4. Deploying New Changes	20
5.1.5. Deployment Verification	21
5.2. PowerBI Reports	23
5.2.1. Prerequisites	23
5.2.2. Server Configuration	23

6. Mobiliya Fleet - Android Application	25
6.1. Prerequisites	25
6.2. Clone the repository for modifications	25
6.3. Steps to build and run application	25

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.

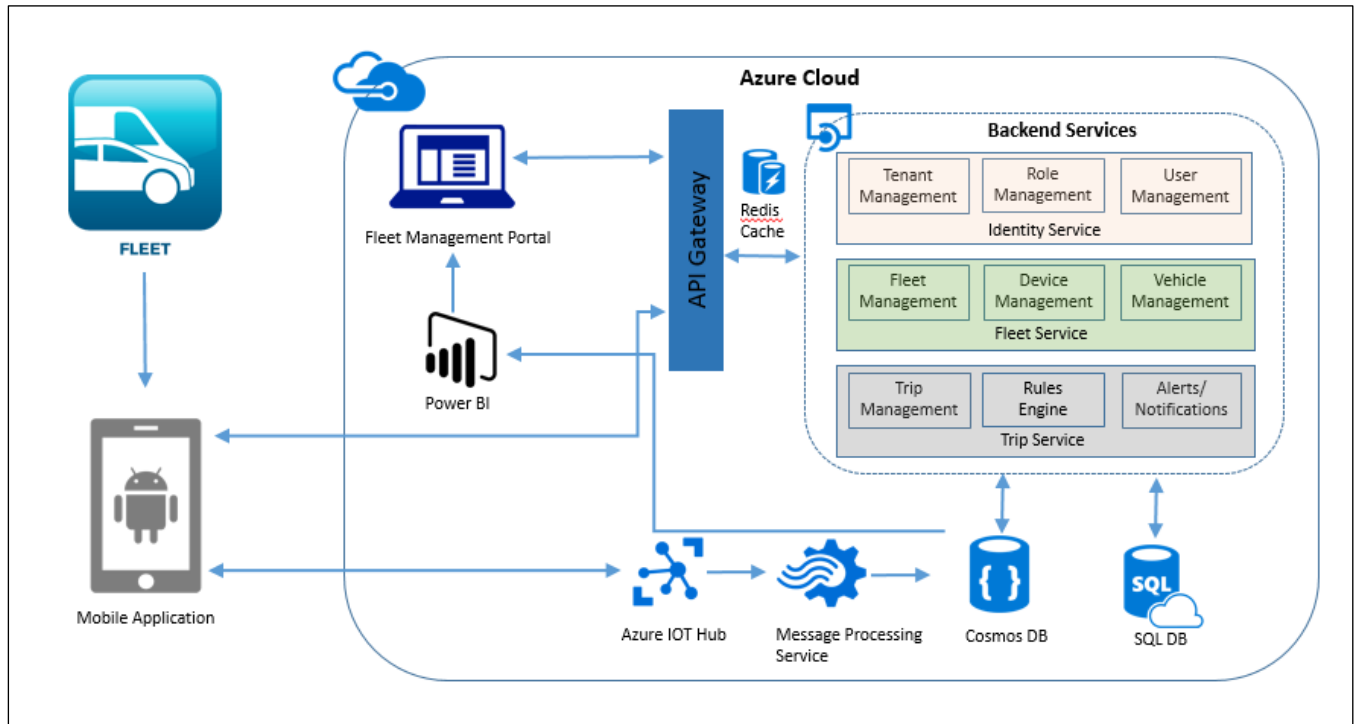
There are 3 types of user actively involved in this system.

- Tenant Admin
- Fleet Admin
- Driver

2. System Architecture

The architecture comprises of two main components:

- Mobile Application
- Cloud Application



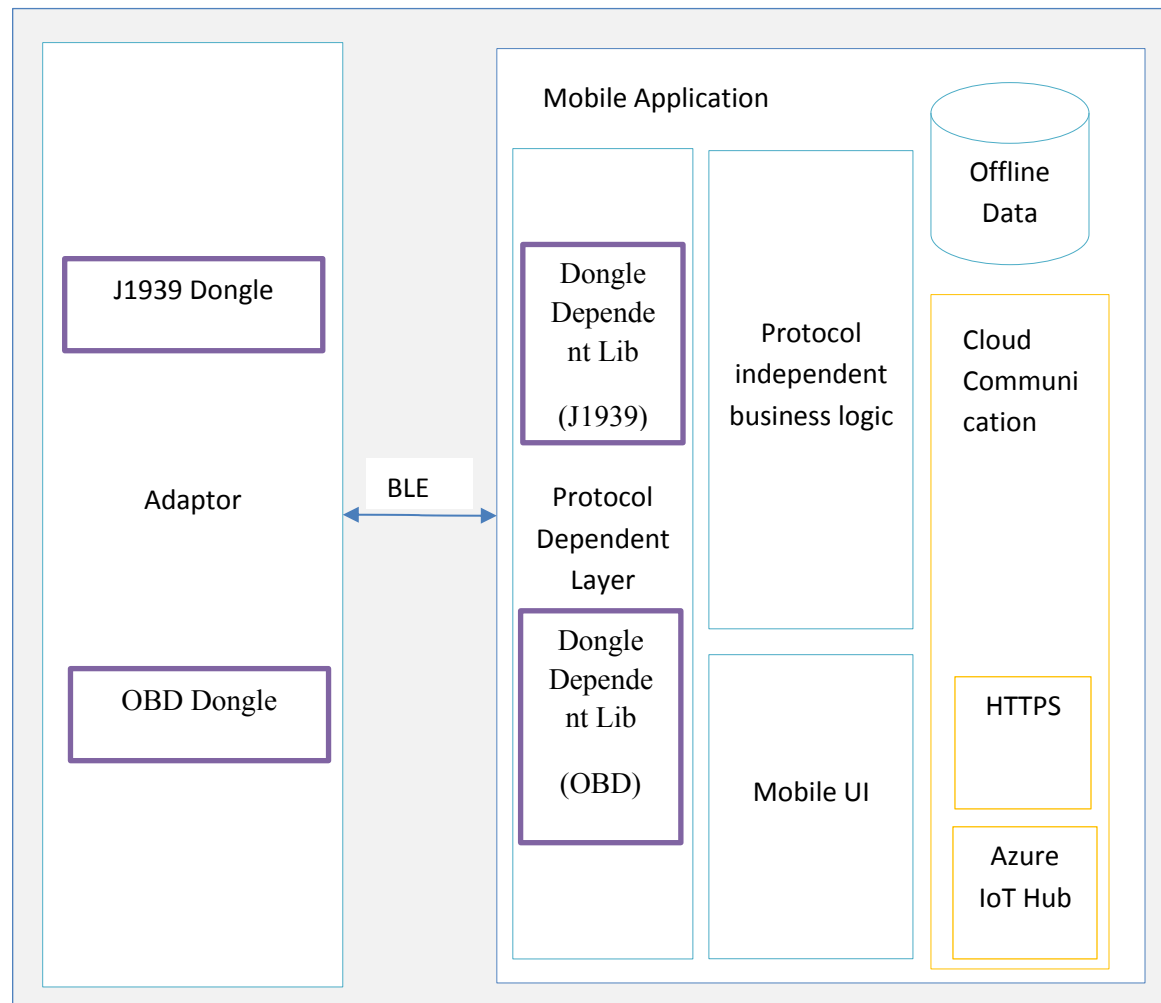
2.1. Mobile App Components

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.

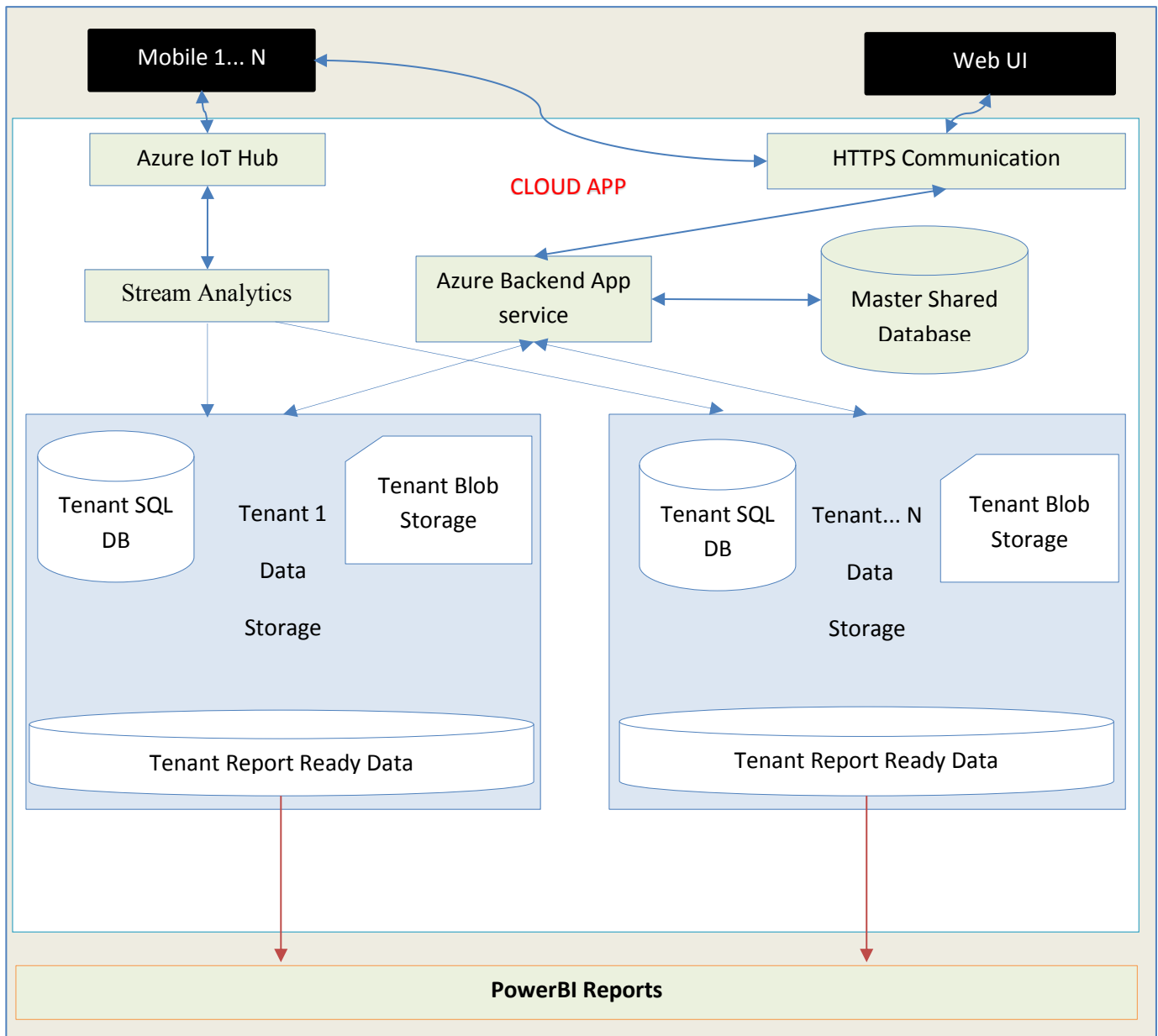
Mobile application has been developed using:

- 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. Active Azure subscription

2. PowerBI setup

User must setup a PowerBI environment using the PowerBI Deployment Guide (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*

3. Link to the ARM (Azure Resource Manager) template that would be used as a script to install the Services,

(<https://github.com/MobiliyaTechnologies/MobiliyaFleetARMTemplate.git>)

4. Follow the Azure Resource Manager Naming Convention,

(<https://docs.microsoft.com/en-us/azure/architecture/best-practices/naming-conventions>)

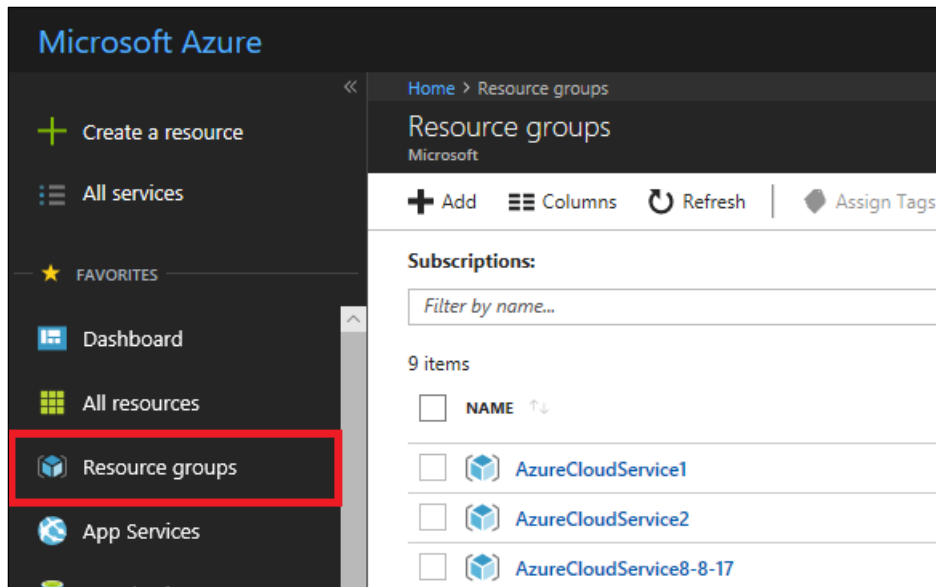
3.1. Step 1: Clone Repository

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:

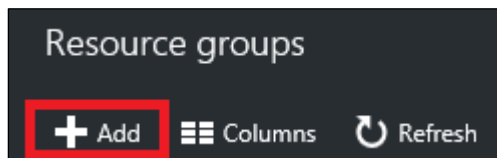
- Clone GitHub repository “MobiliyaFleetARMTemplate” using the URL given below:
(<https://github.com/MobiliyaTechnologies/MobiliyaFleetARMTemplate>)

3.2. Step 2: Create Resource Group (Refer only if not created)

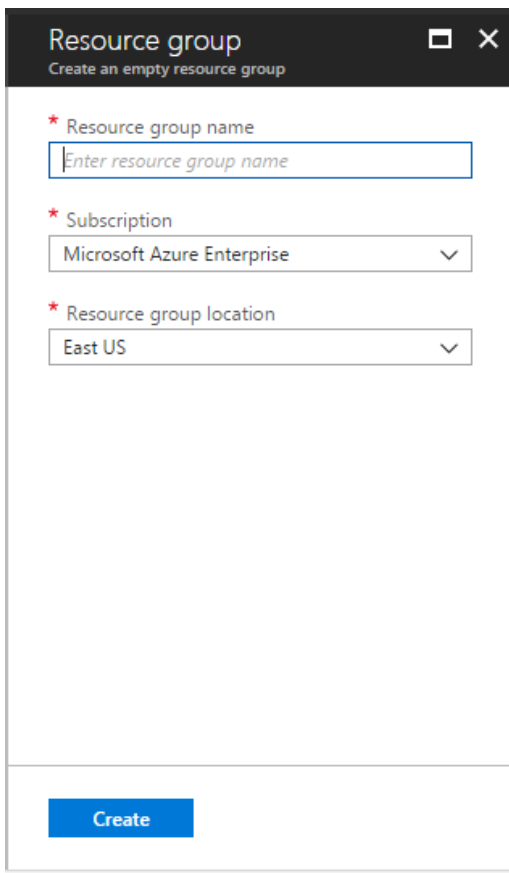
- If the resource group is already created, use existing resource group and skip to next step.
- Login to Azure portal (<https://portal.azure.com>) and select the appropriate subscription if it is not selected by default. Create a resource group that serves as the container for the deployed resources.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



The screenshot shows the 'Resource group' creation form. It includes the following fields:

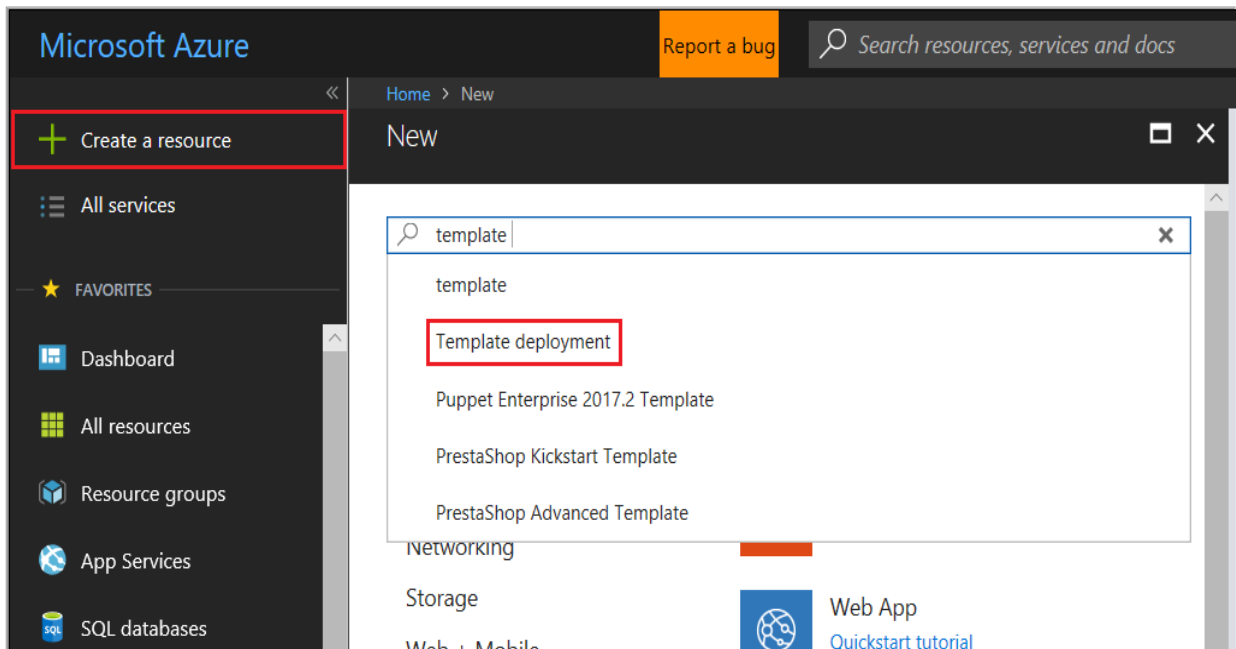
- Resource group name:** A text input field with the placeholder 'Enter resource group name'.
- Subscription:** A dropdown menu currently showing 'Microsoft Azure Enterprise'.
- Resource group location:** A dropdown menu currently showing 'East US'.

A blue 'Create' button is located at the bottom of the form.

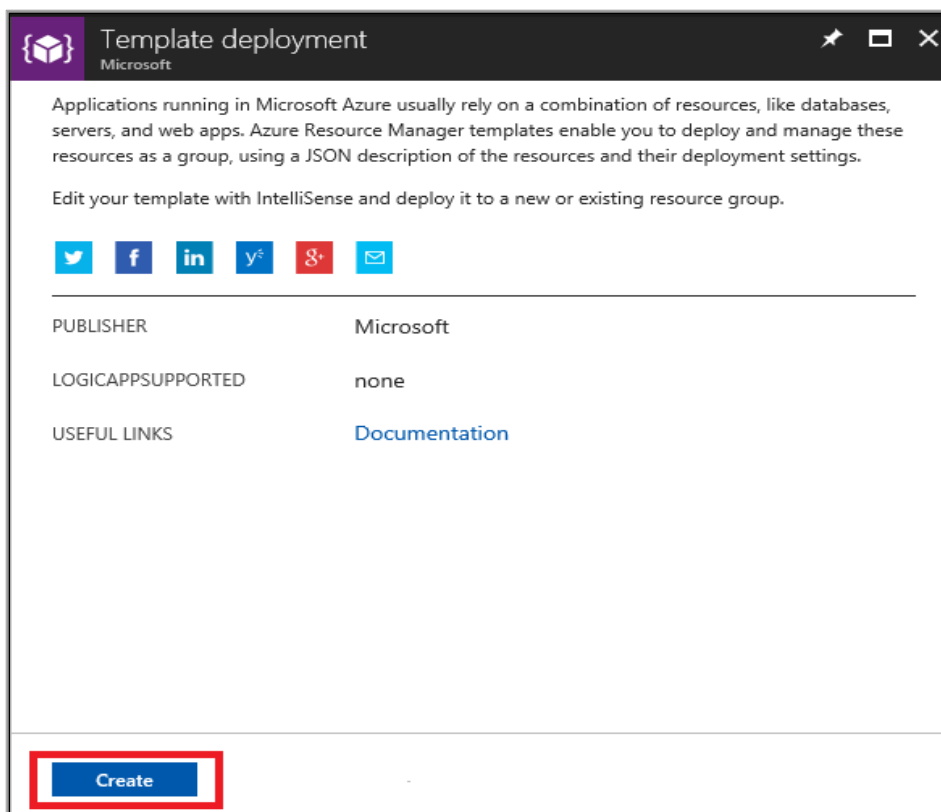
3.3. Step 3: Create Resources Using Template

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

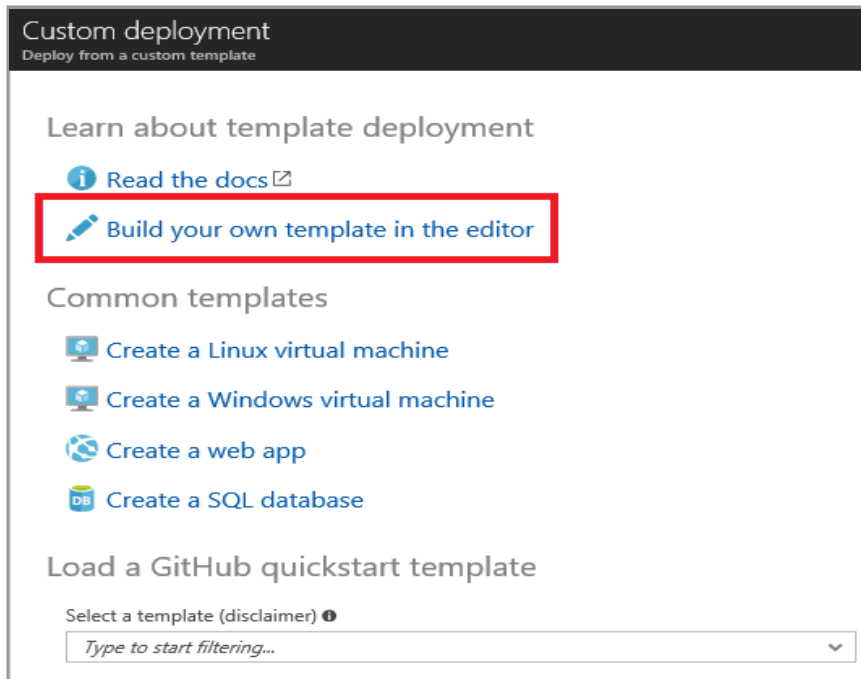
- Click on **+Create a resource** from the left menu.
- Select **Template deployment option**
- Load the azuredeploy.json template from your cloned project directory ('c:\users\...\projects').



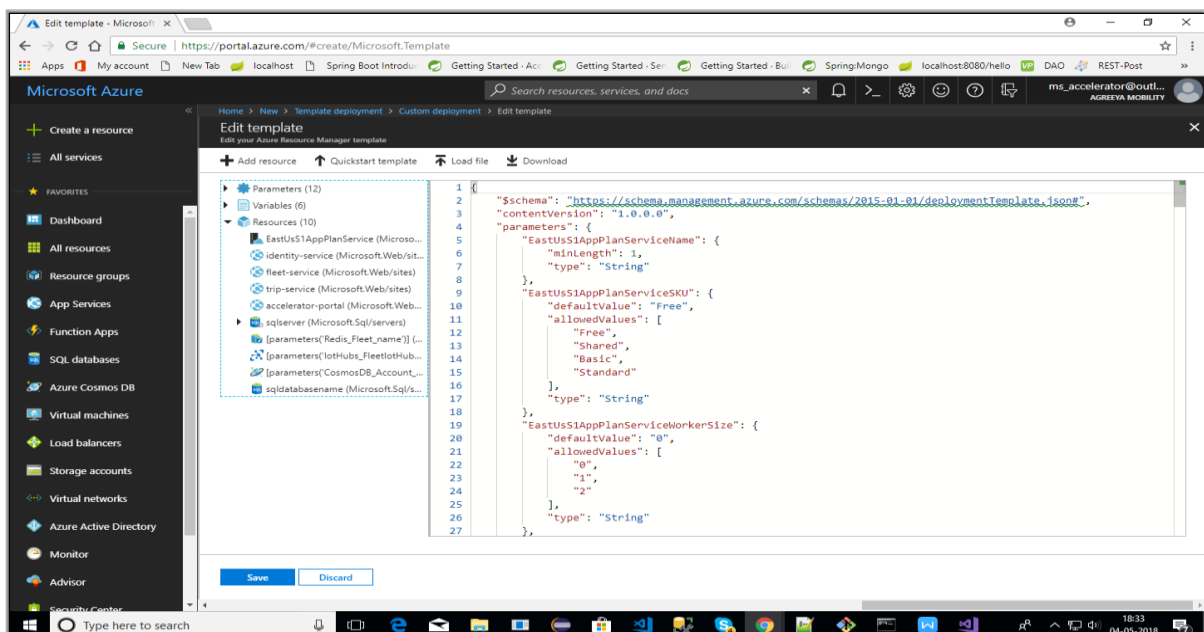
- Click on **Create**



- Click on **Build your own template in the editor**



- Click on **Load file** located in the top navigation bar. Upload deployment template file (azuredeploy.json) from your cloned project in step 1.

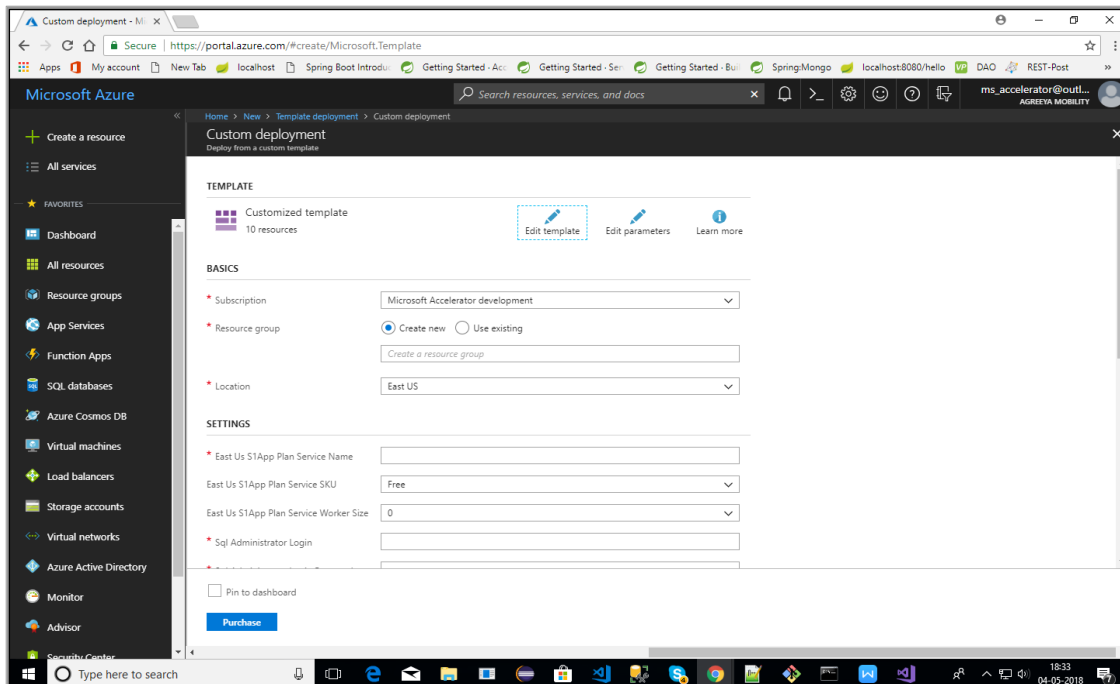


- Once the file is uploaded, click on **Save** button. A new form appears which accepts some values before the actual deployment.

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"**

Note: For “PowerBIUserName” and “PowerBIPassword”: use own PowerBI credentials.
 For “PowerBIClientId” and “PowerBIClientSecret”: Get it from step 2 as described in the pre-requisites section above.



- * 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.

4. Deployment of Backend Services on Azure

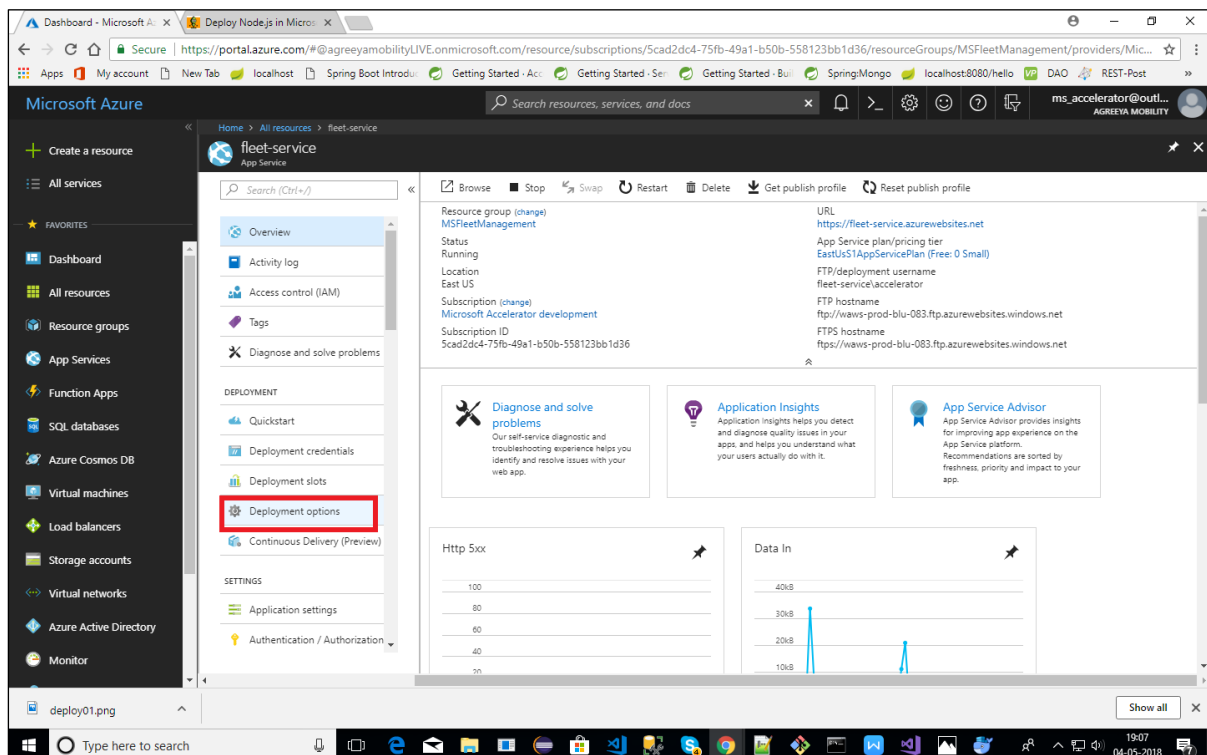
The following steps will enable user in deploying the Web Application.

Pre-requisites

- User must have Git installed.
- Download the Fleet Management Projects from GitHub on your local computer using below URLs:
 - *Identity Service*: <https://github.com/MobiliyaTechnologies/IdentityService>
 - *Fleet Service*: <https://github.com/MobiliyaTechnologies/FleetService>
 - *Trip Service*: <https://github.com/MobiliyaTechnologies/TripService>

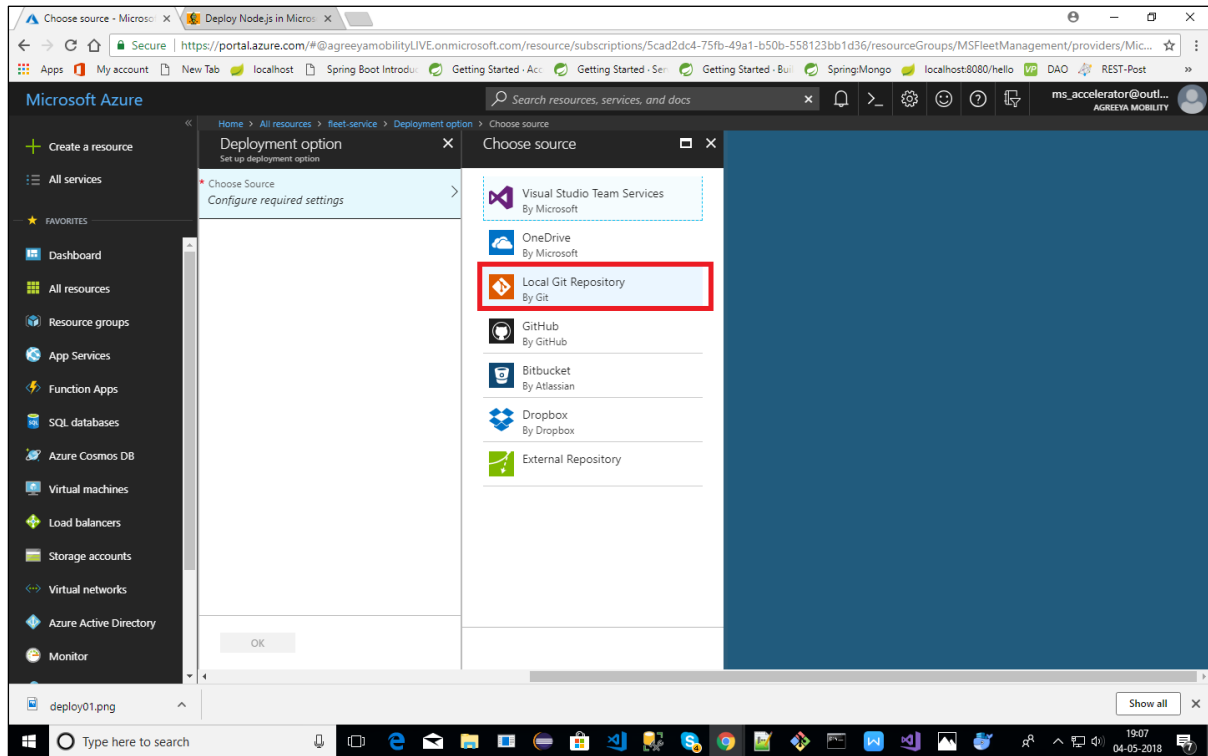
4.1. Step 1: Select Deployment Option

- Go to azure portal (<https://portal.azure.com>).
- Once resources are created, click on any app service (Identity/Fleet/Trip)
- Select **Deployment Options**.



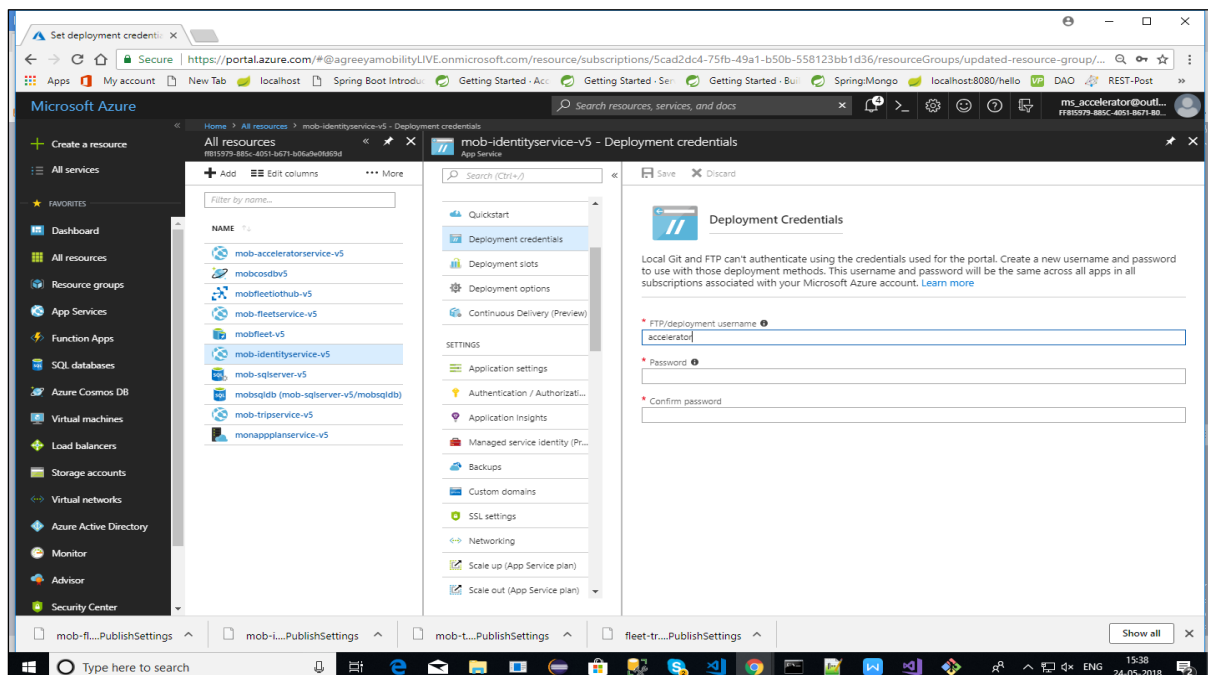
4.2. Step 2: Set Deployment Source- Local Git

- Click on choose resource and select **Local Git Repository** option as shown below:



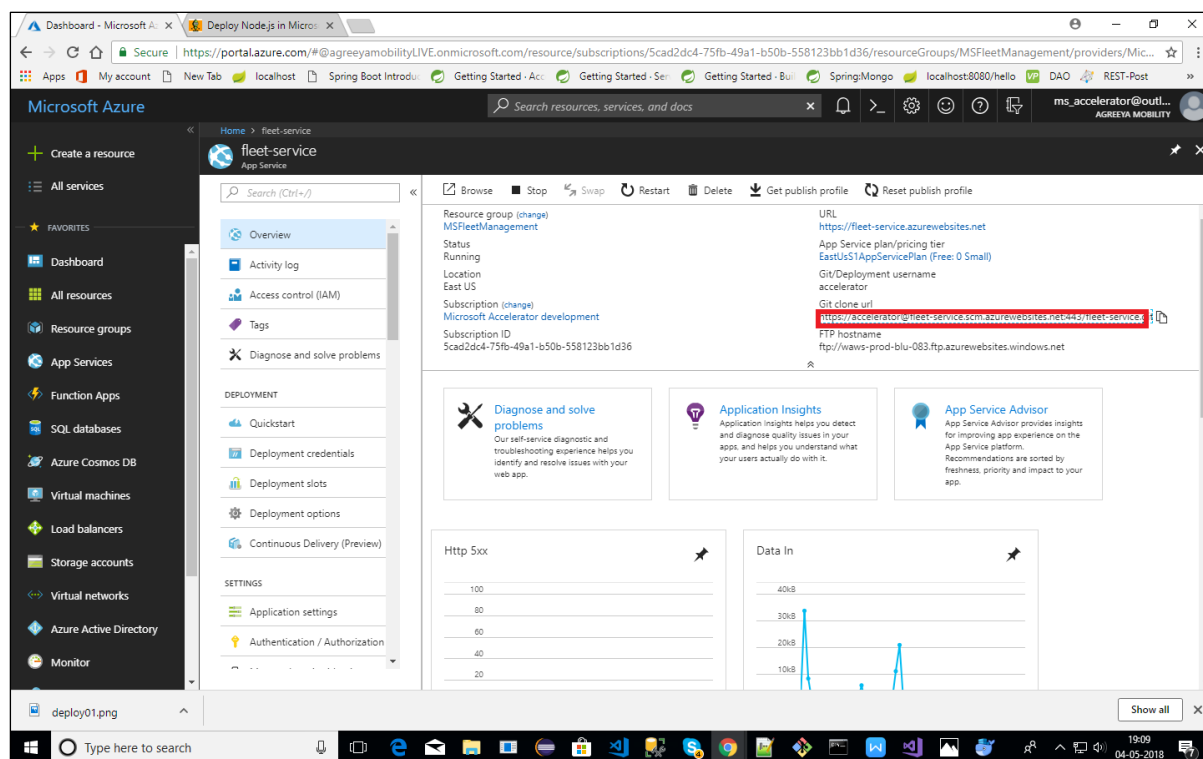
4.3. Step 3: Set Deployment Credentials

- Set Deployment Credentials.**



4.4. Step 4: Copy Git URL

- Copy the Git URL link (as shown in the image below)
- Clone it on your local computer using the credentials set in **Step 3**.
 - This will create empty Git repos on your computer.



4.5. Step 5: Repeat Steps for Other Services

- Repeat step 1, 2 and 4 for rest of the services (Identity/Trip).

4.6. Step 6: Deploy Actual Code on Azure

- Move the actual code cloned from GitHub to empty repositories created in above steps for all the services (Fleet/Identity/Trip).
- On your computer, move the cloned project from section 5.1 (IdentityService/ FleetService/ TripService) to empty Azure git repositories created in step 1 to 5 above.
 - Open Git Bash.
 - Change the current working directory to Azure git repository.
 - Stage the file for commit to Azure git repository using below command.

```
$ git status
```

```
$ git add .
```

Commit the files to Azure git repository using the command given below:

```
$ git commit -m "Add existing files".
```

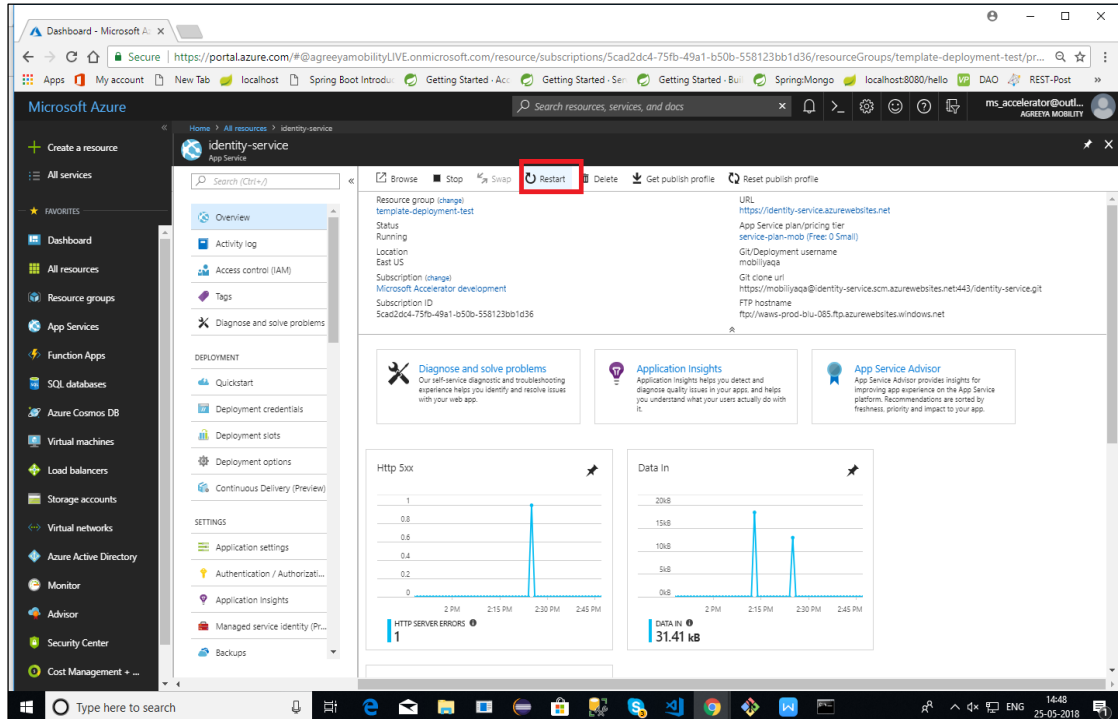
Push the changes to Azure git repository using the command given below:

```
$ git push origin master
```

- The application would be deployed.

4.7. Step 7: Restart Application

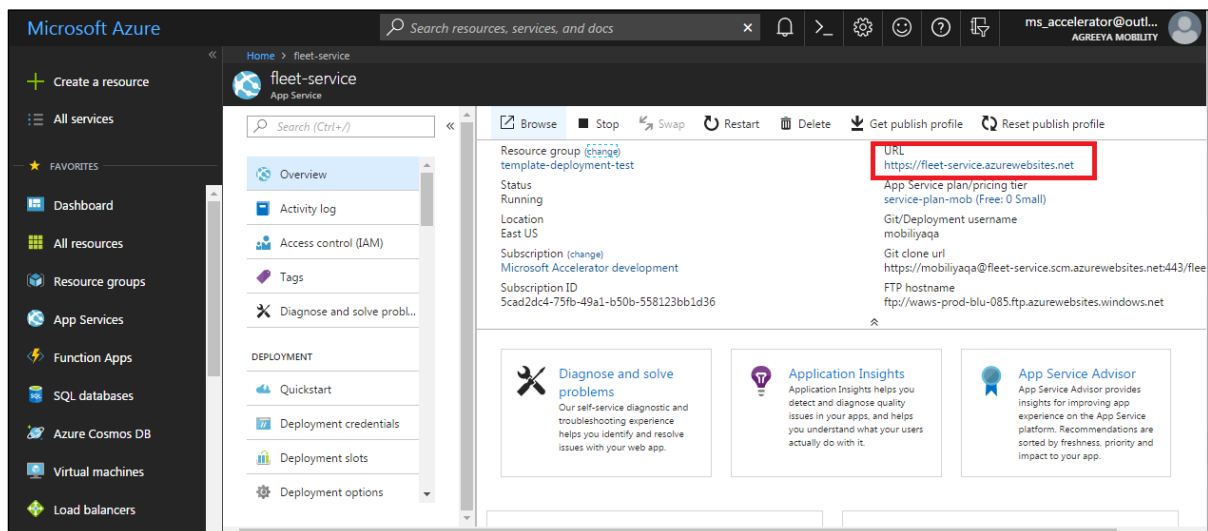
- On Azure portal, restart the application.



* Now every time you push code to azure repo, site will be automatically deployed.

4.8. Step 8: Verify Service is working

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



Note: Save URL highlighted in red above. This will be required for configuring android app. Follow same steps to get URLs for other services.

5. Deployment of Web Portal on Azure

5.1. Web Portal Setup and Configuration changes

5.1.1. Pre-requisites

- User must have Git installed.

5.1.2. Clone The Repository

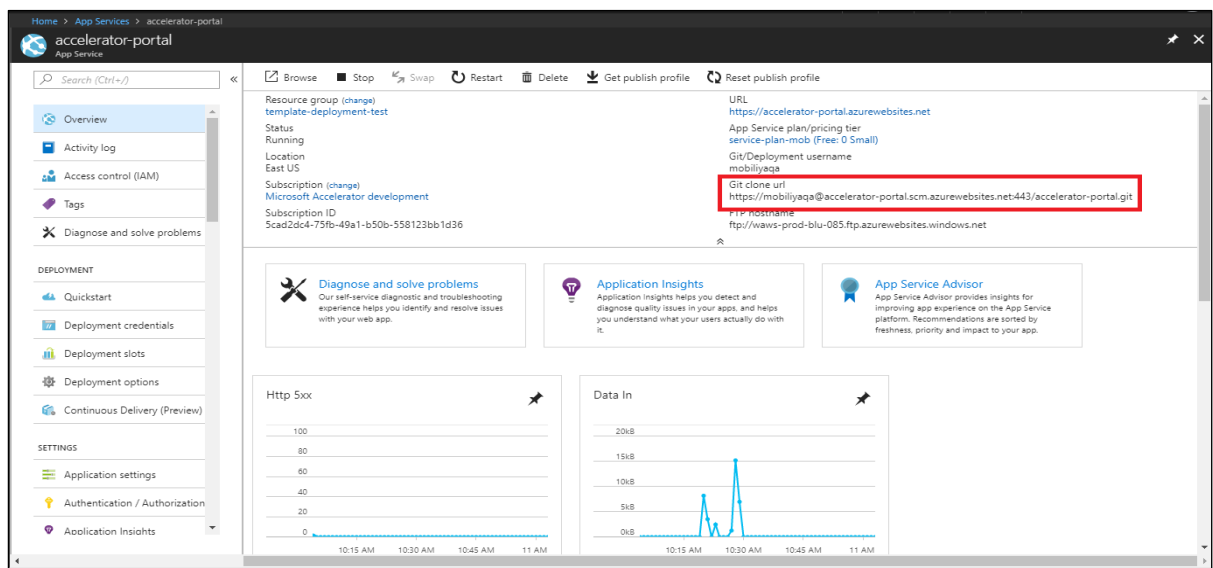
- User must clone the repository “MobiliyaFleetWebPortal” using the URL given below:

<https://github.com/MobiliyaTechnologies/MobiliyaFleetWebPortal>

5.1.3. Deploy The Application

To deploy your application, follow the steps given below:

- Open Azure portal (<https://portal.azure.com>).
Follow **step 1** and **step 2** from section **5: Deployment of web app services in azure**, to set azure Local Git Repository.
- Clone the local Git repository of azure app service (accelerator-portal) using URL as shown in the image given below:



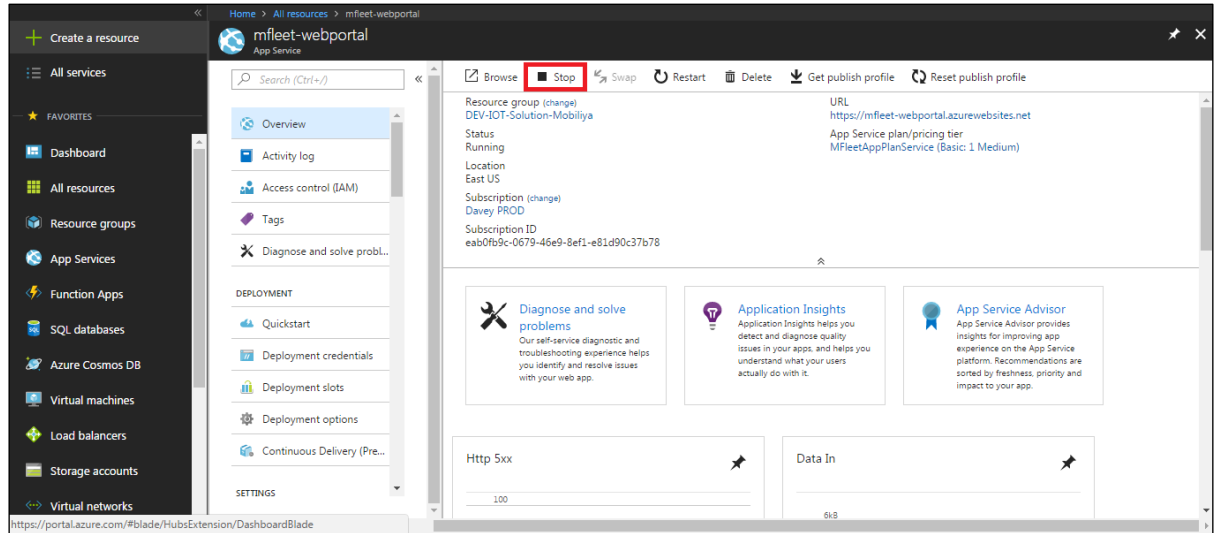
- Go to source folder (cloned in step 2). Copy all the files into newly cloned repository in above step.
- Push the code using the commands given below:
 - `git add .`
 - `git commit -m "message-string"`
 - `git push origin master`

Note: The deployment will take some time (20 to 30 minutes) as it will automatically download and configure packages required for web-portal to run.

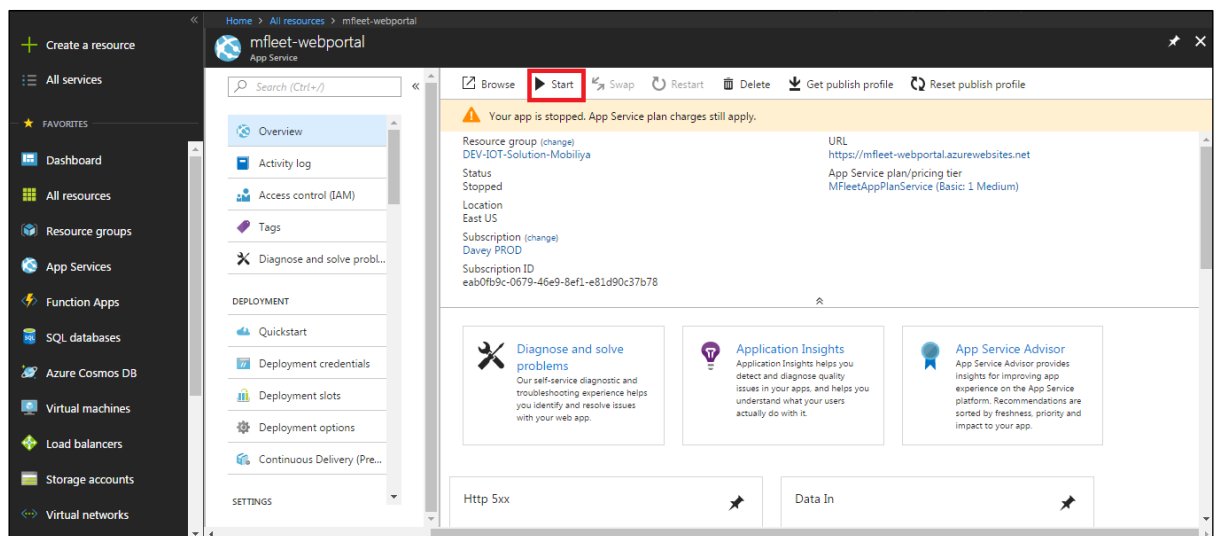
5.1.4. Deploying New Changes

This is an optional step. This is required only if the user wants to add new changes in the code that has already been deployed. Skip this step if no changes are required.

- i. To deploy new changes, user must first **“Stop”** the web app (refer below image).

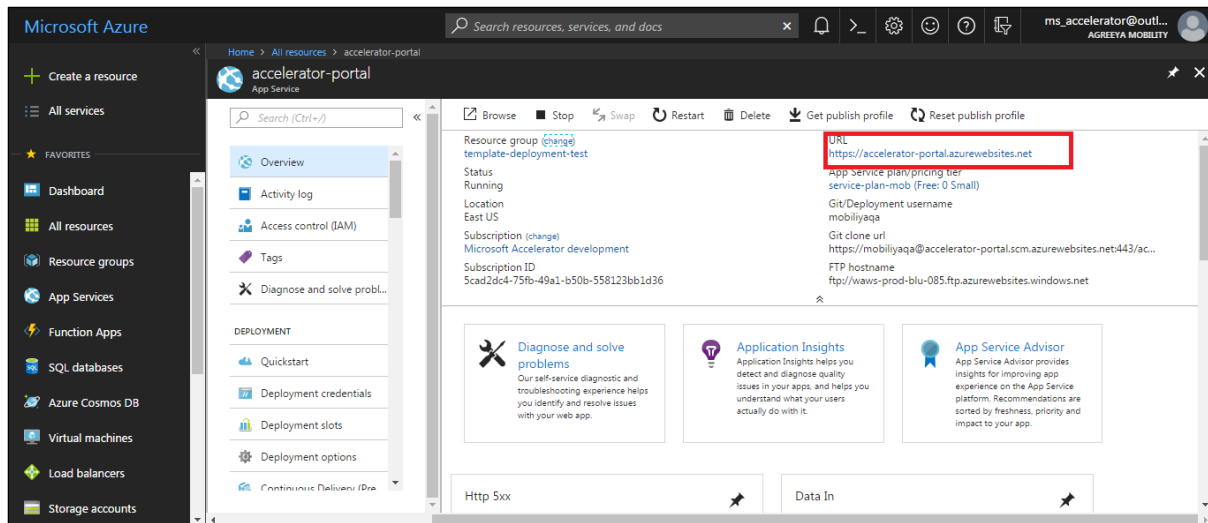


- ii. Follow steps mentioned in 6.1.3 to deploy updated code.
- iii. Then **“Start”** the application (refer below image).

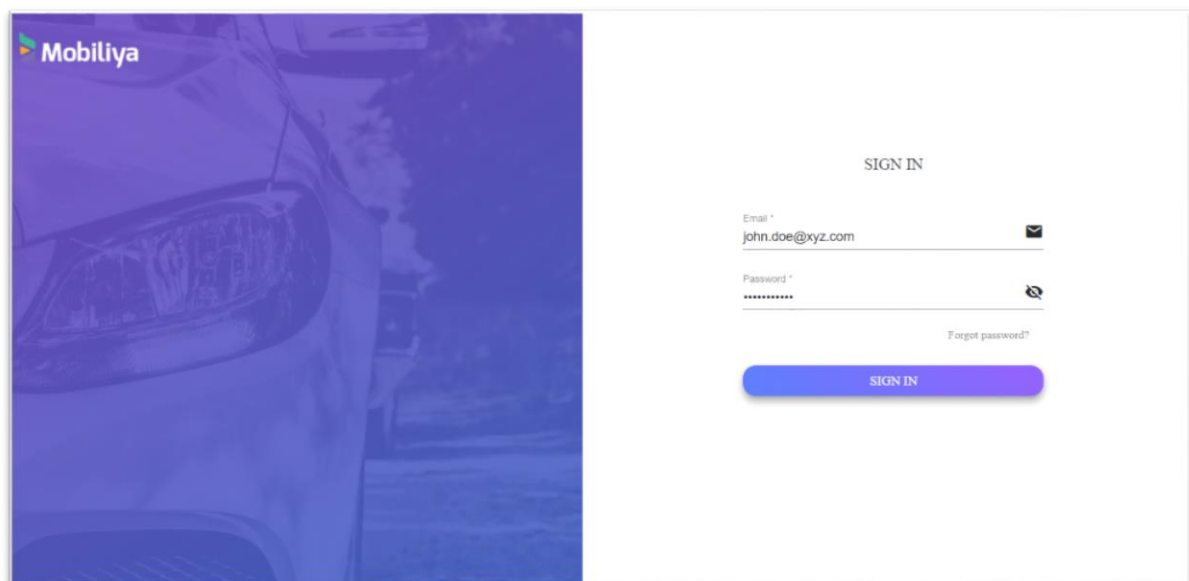


5.1.5. Deployment Verification

Once the user has successfully pushed the code, the application will start running. To start using application, open application portal URL as shown in the image given below.



On successful deployment, launch the application URL in a browser. It will open the login page.



The default login credentials for a tenant are pre-configured in the web portal as mentioned below:

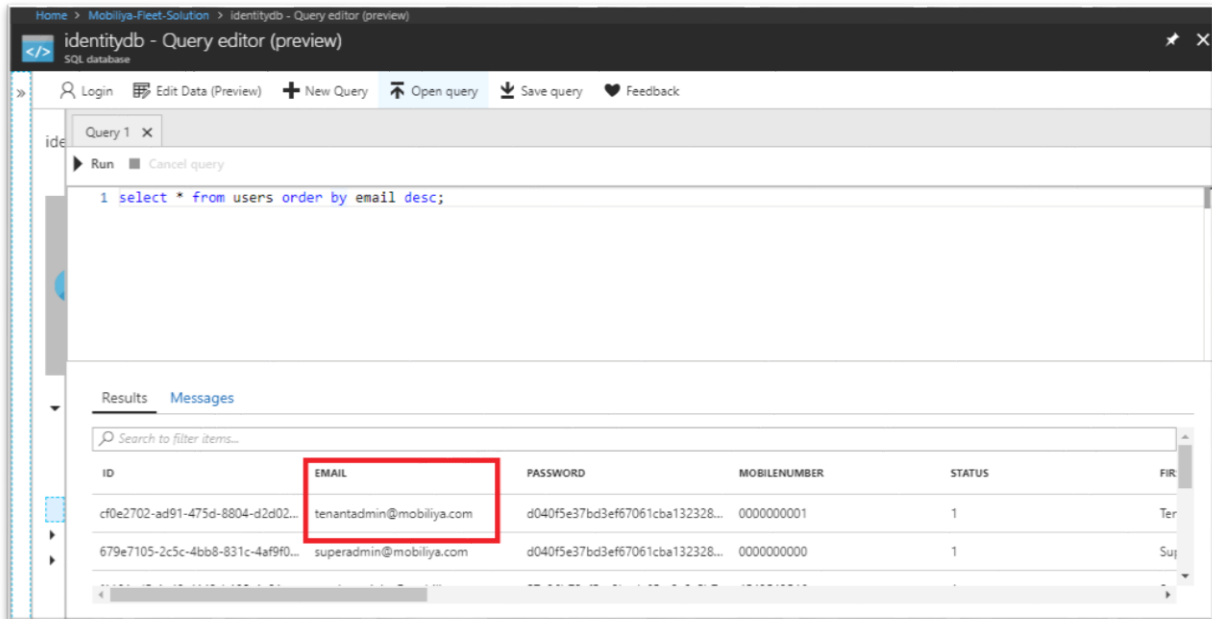
Tenant Admin:

- Username: tenantadmin@mobiliya.com
- Password: welcome

To change the login username for tenant,

- Go to **Resource Groups** -> **<Resource group name>** -> **<SQL-server-name>**
- Click on **Query editor** (left side panel)
- Login using SQL server credentials entered in ARM template while deploying resources.

- On successful login, user will find tables listed in the left side panel.
- Update the 'EMAIL' field under 'users' table using SQL Query.



The screenshot shows the 'identitydb - Query editor (preview)' interface. The query editor contains the following SQL query:

```
1 select * from users order by email desc;
```

The results are displayed in a table with the following columns: ID, EMAIL, PASSWORD, MOBILENUMBER, STATUS, and FIR. The 'EMAIL' column is highlighted with a red box.

ID	EMAIL	PASSWORD	MOBILENUMBER	STATUS	FIR
cf0e2702-ad91-475d-8804-d2d02...	tenantadmin@mobiliya.com	d040f5e37bd3ef67061cba132328...	0000000001	1	Ter
679e7105-2c5c-4bb8-831c-4af9f0...	superadmin@mobiliya.com	d040f5e37bd3ef67061cba132328...	0000000000	1	Suj

5.2. PowerBI Reports

5.2.1. Prerequisites

- The PowerBI account credentials are configured in the web portal and backend service as part of ARMTemplate deployment steps before.
- The PowerBI templates and report generation steps as mentioned in the GitHub URL are given below:

<https://github.com/MobiliyaTechnologies/MobiliyaFleetPowerBI>

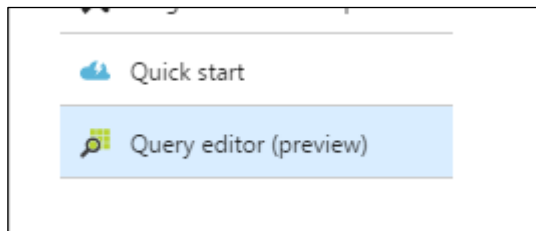
- The Reports are generated using the template files and the report IDs for each reports are noted.

5.2.2. Server Configuration

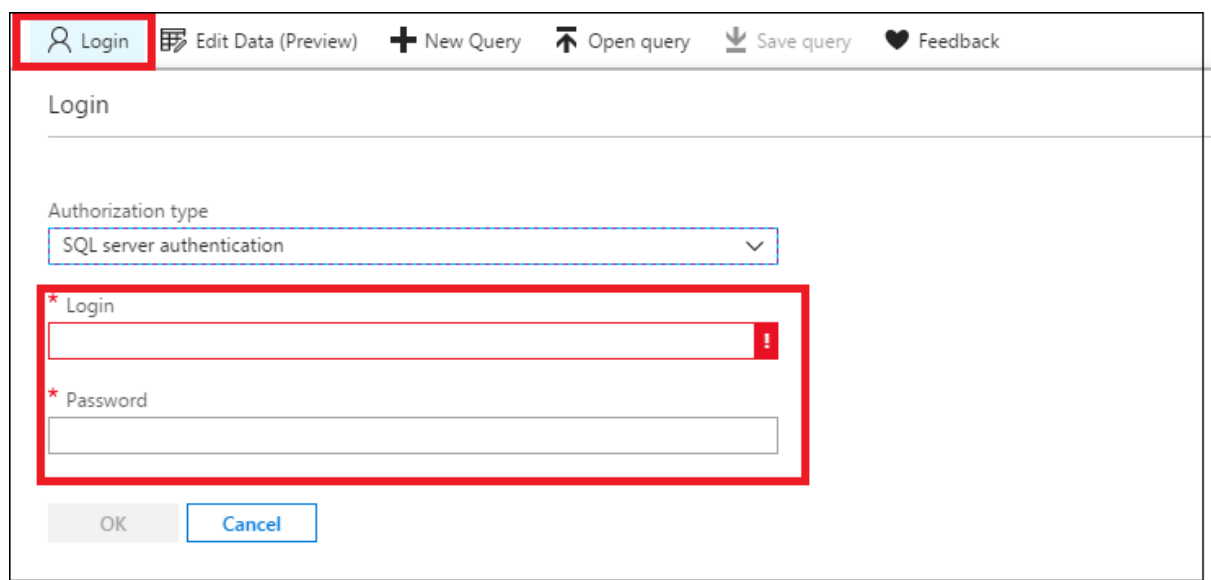
- Once the steps mentioned above are successfully completed, user will receive report IDs which need to be updated in the SQL database (**report** table) created after deploying ARM template.
- Open azure portal (<https://portal.azure.com>) in web browser.

Go to **Resource Groups** -> **<Resource group name>** -> **<SQL-server-name>**

Click on **Query editor** (left side panel)



Login using SQL server credentials entered in ARM template while deploying resources.
(Refer image given below)



On successful login, user will find tables listed in the left side panel.

Update/insert reports data into **dbo.reports** table using SQL query.

(Please find reference image below. User can update report id/tenant id using SQL query)

	id	reportName	reportId	tenantId	createdAt	updatedAt
1	1	mileage	0458ae69-5d84-431a-902d-0c0004bf4b21	3548e3fa-a8e7-48ee-9df8-b3e326f30345	2018-05-31 09:57:06.9560000 +00:00	2018-05-31 09:57:06.9560000 +00:00
2	2	rpm	cbdebe17-0433-4e7b-886e-1cee2eef74ed	3548e3fa-a8e7-48ee-9df8-b3e326f30345	2018-05-31 09:57:06.9560000 +00:00	2018-05-31 09:57:06.9560000 +00:00
3	3	speed	d783eedb-9093-4051-b4bb-ad883f8b884a	3548e3fa-a8e7-48ee-9df8-b3e326f30345	2018-05-31 09:57:06.9560000 +00:00	2018-05-31 09:57:06.9560000 +00:00
4	4	fault_codes	f3ea076a-9ddf-458d-90ef-d7e5fe85d6f5	3548e3fa-a8e7-48ee-9df8-b3e326f30345	2018-05-31 09:57:06.9560000 +00:00	2018-05-31 09:57:06.9560000 +00:00
5	5	average_mileage	72abb58e-b756-4a1a-9b5a-3587b5c43b5d	3548e3fa-a8e7-48ee-9df8-b3e326f30345	2018-05-31 09:57:06.9560000 +00:00	2018-05-31 09:57:06.9560000 +00:00

- Once these entries are updated in the table the reports are ready to be displayed on the web portal.

6. Mobiliya Fleet - Android Application

6.1. Prerequisites

- Device support - Android 6.0 Marshmallow and above.
- Android application is tested on OBD-II (VIVTRON) and J1939 (BlueFire LE simulator) dongles supporting Bluetooth.

6.2. Clone the repository for modifications

If a user wants to do any more modifications in the app, then user can download source code from the location given below:

<https://github.com/MobiliyaTechnologies/MobiliyaFleetAndroid>

6.3. Steps to build and run application

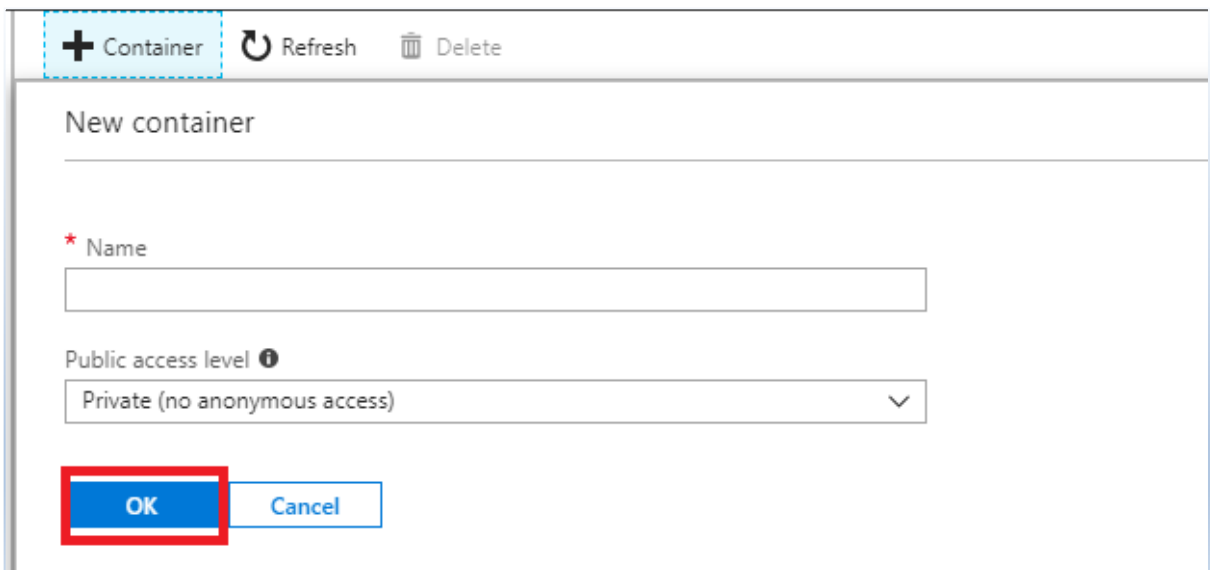
- Clone or Download code from Git hub.
- Import Code directly in Android studio and build the application.
- Click Run->Run <app_name> to install app on android device. Alternatively, user can build a signed apk and install on the android device.

Note: You need to upload this apk on blob storage created using ARM template. Follow below steps. (This is required in order to integrate application apk URL in welcome mail that is send to driver when he/she is added in the system)

- 1 Go to **Resource Groups** -> <Resource group name> -> <blob storage account>
2. Select **Blobs** option.
3. Click on create container(refer image below)



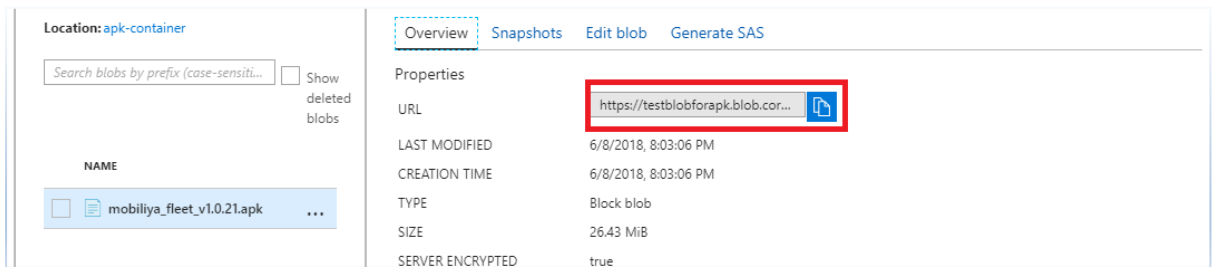
4. Fill the information(set Public **access level to blob**) and click on **OK**.



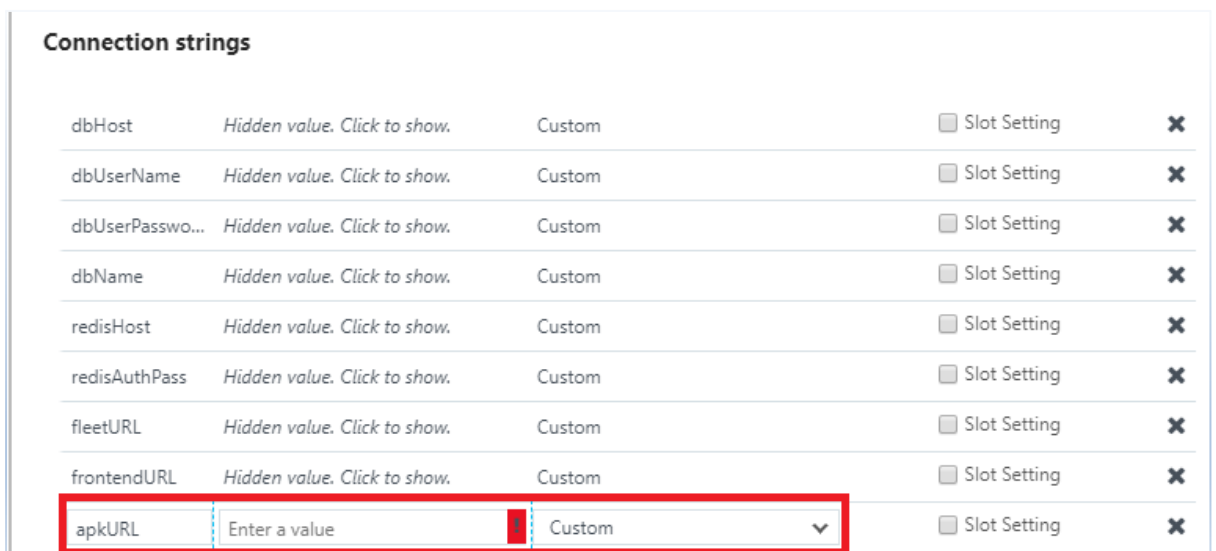
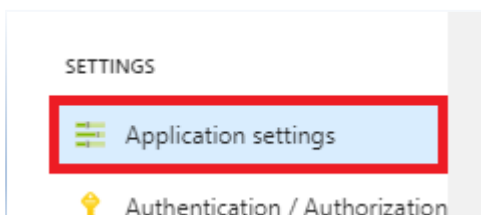
5. Click on upload and select the apk file from your local system and upload it.



6. Copy blob URL



7. Paste blob url in connection string property “apkURL” of identity service(refer below images). Go to identity app service. And click on “Application settings”



After successful deployment of the backend services, web portal and android application, follow the user guide document for the detailed usage of the solution.