



FLEET MANAGEMENT SOLUTION

User Guide

Revision History	
Version No.	1.0
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. Intended Audience	9
4. Deploying Resources using ARMTemplate on Azure.....	10
4.1.1. Step 1: Clone Repository	10
4.1.2. Step 2: Create Resource Group.....	10
4.1.3. Step 3: Create Resources Using Template	11
5. Deployment of Backend Services on Azure:	15
5.1. Pre-requisites:	15
5.1.1. Step 1: Select Deployment Option.....	15
5.1.2. Step 2: Set Deployment Source- Local Git	15
5.1.3. Step 3: Set Deployment Credentials	16
5.1.4. Step 4: Copy Git Url	16
5.1.5. Step 5: Repeat Steps For Other Services	17
5.1.6. Step 6: Deploy Actual Code On Azure	17
5.1.7. Step 7: Restart Application	17
5.1.8. Step 8: Verify Service is Working	18
6. Deployment of Web Portal on Azure	19
6.1. Setup and Configuration changes	19
6.1.1. Pre-requisites	19
6.1.2. Clone the repository	19
6.1.3. Environment/config changes	19
6.1.4. Build and Run	19
6.1.5. Deploy:	20
6.2. Tenant Admin.....	21
6.2.1. Login.....	21
6.2.2. Add User.....	22
6.2.3. Fleet	22
6.2.4. Vehicles	23

6.2.5.	Devices	23
6.2.6.	Reports	23
6.2.7.	Rules.....	23
7.	Android Application.....	24
7.1.	Prerequisites	24
7.2.	Configuration changes	24
7.3.	Driver Sign In	25
7.4.	Forgot Password	26
7.5.	Dashboard	26

1. Introduction

1.1. About this Guide

The purpose of this user 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 & usage guidelines 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.

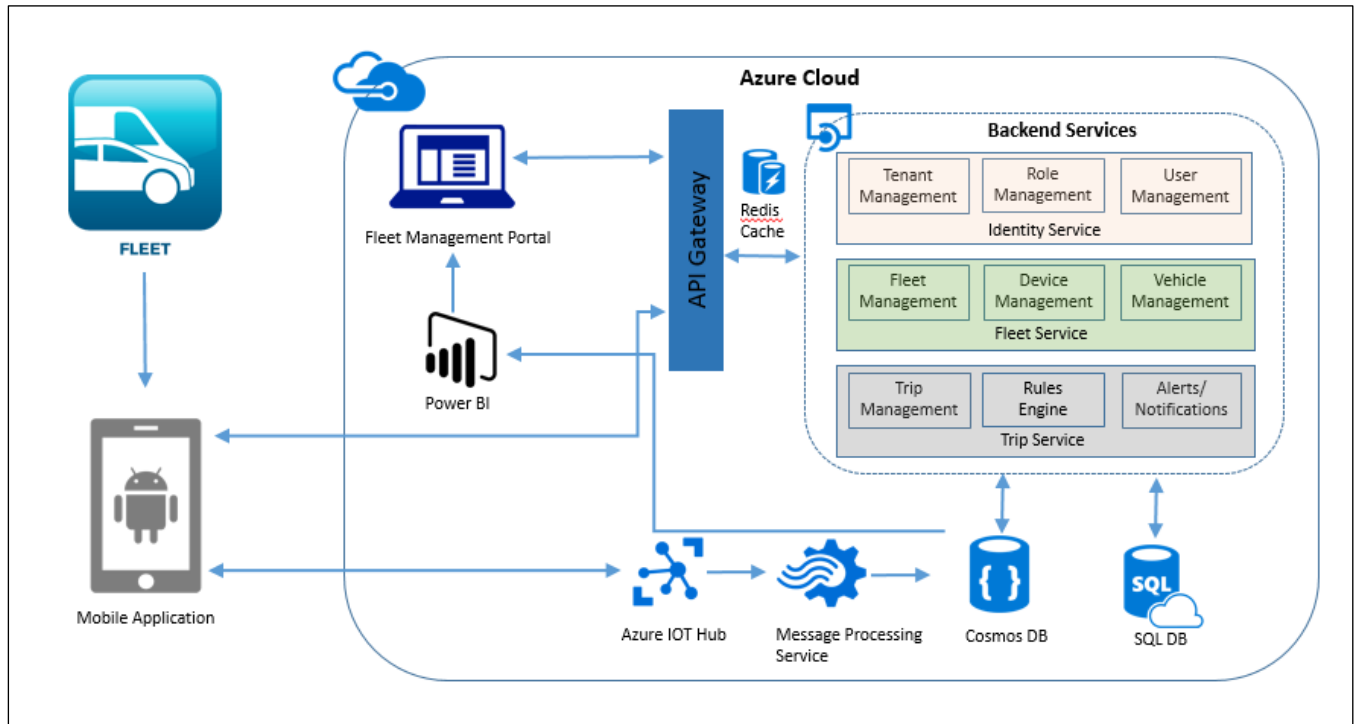
There are 4 types of user actively involved in this system.

- Super Admin (A person who has complete access to system)
- Tenant Admin
- Fleet Admin
- Driver

2. System Architecture

The architecture comprises of two main components:

- Mobile Application
- Cloud Application



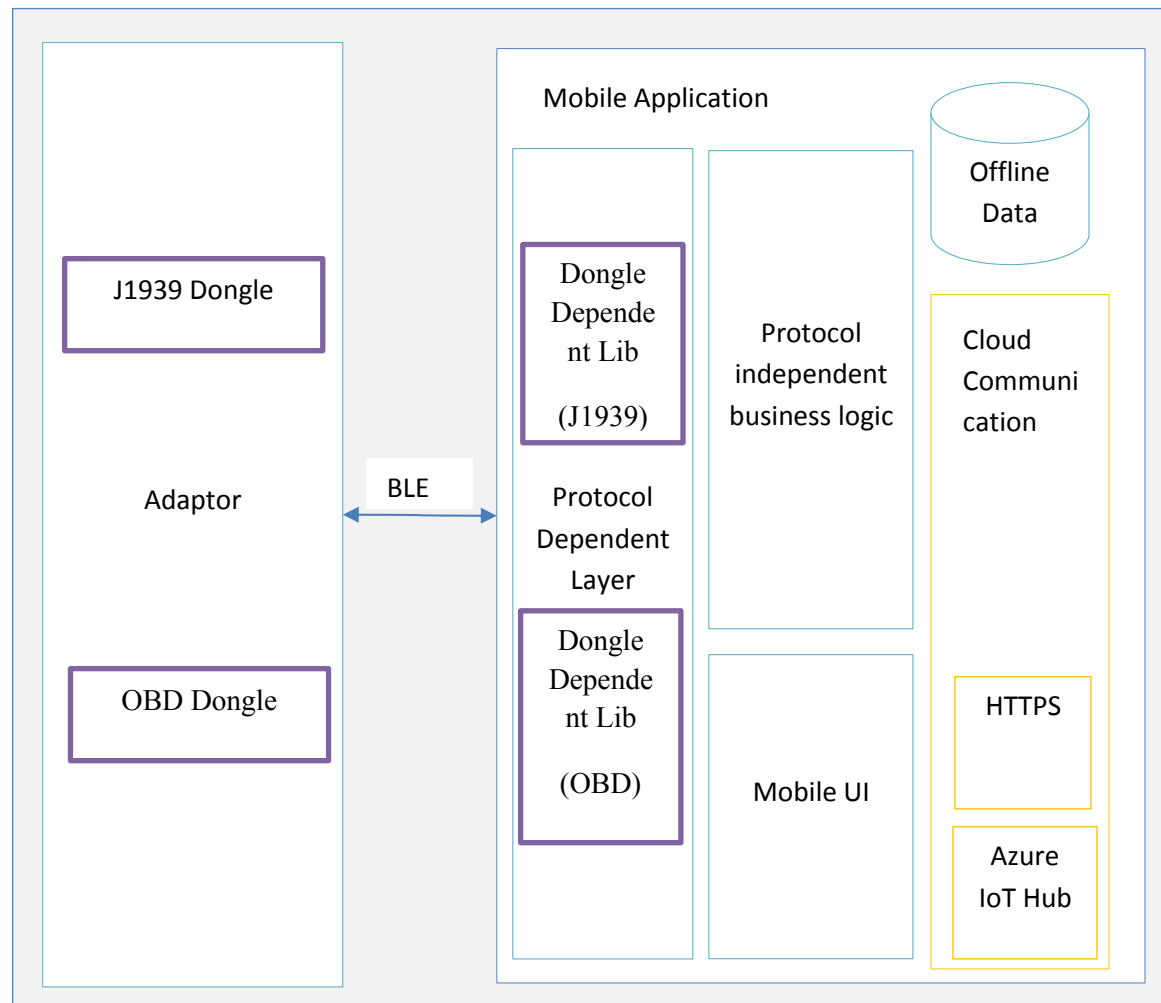
2.1. Mobile App Components

The mobile application has been developed using Android Native Platform. This mobile application 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.

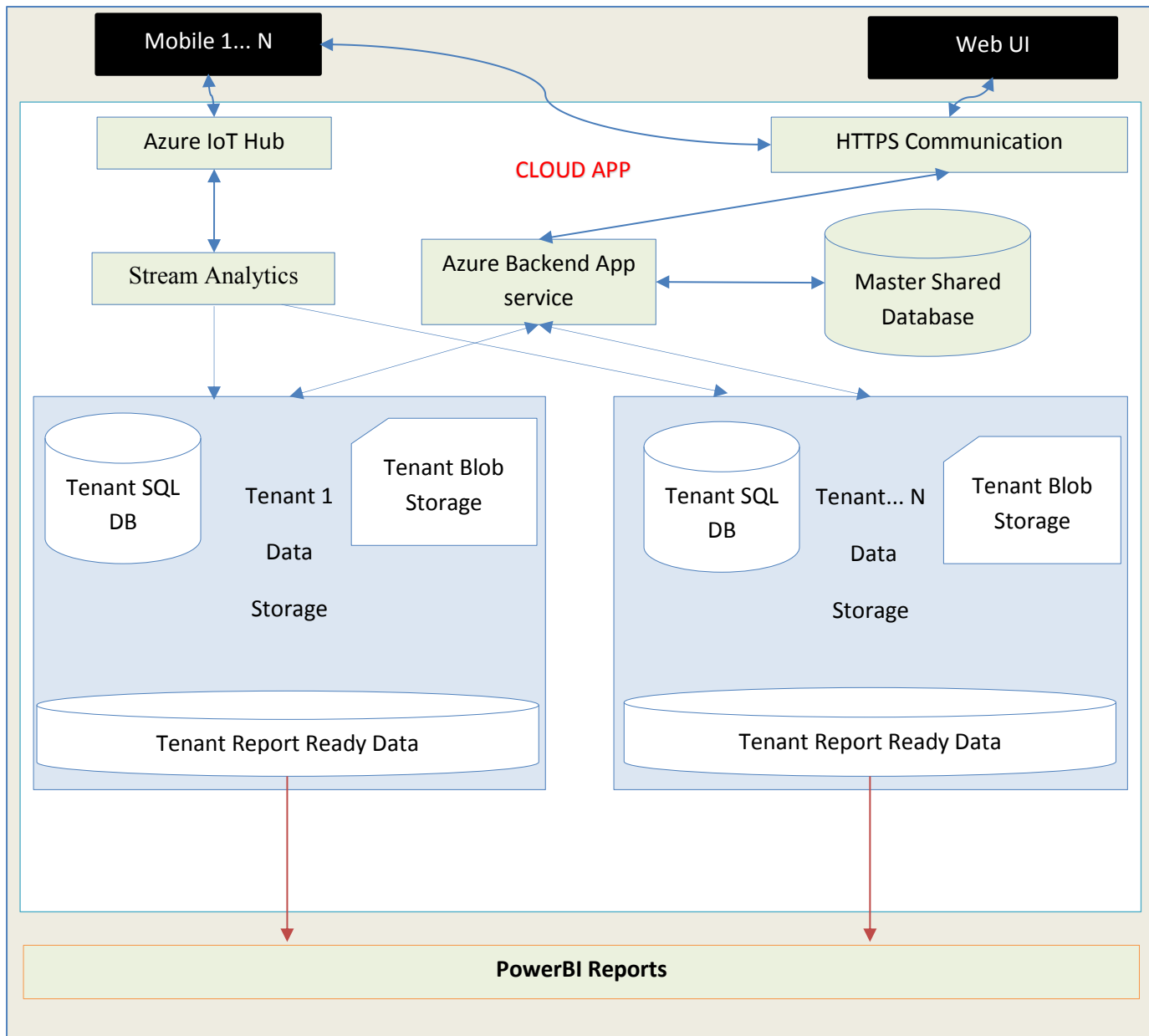
Mobile application has been developed using:

- 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. Intended Audience

This guide is intended for the users who want to use the system. The guide explains different flows for following user roles:

a. Super Admin

- By default, a Super Admin user is created in the system
- A Super Admin can manage user accounts

b. Tenant Admin

- Tenant Admin is added by Super Admin
- Each company which registers will have one Tenant Admin, who can manage users
- Tenant Admin can also manage fleets, vehicles and view reports

c. Fleet Admin

- Fleet Admin is added by tenant
- Fleet Admin can add, update or delete drivers
- Fleet Admin can also remove vehicles from own fleet

d. Driver

- Drivers can only login to the Android app and are not allowed to login on the web portal

4. 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.

4.1. 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. 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>)
3. Read the Azure Resource Manager Naming Convention
(<https://docs.microsoft.com/en-us/azure/architecture/best-practices/naming-conventions>)

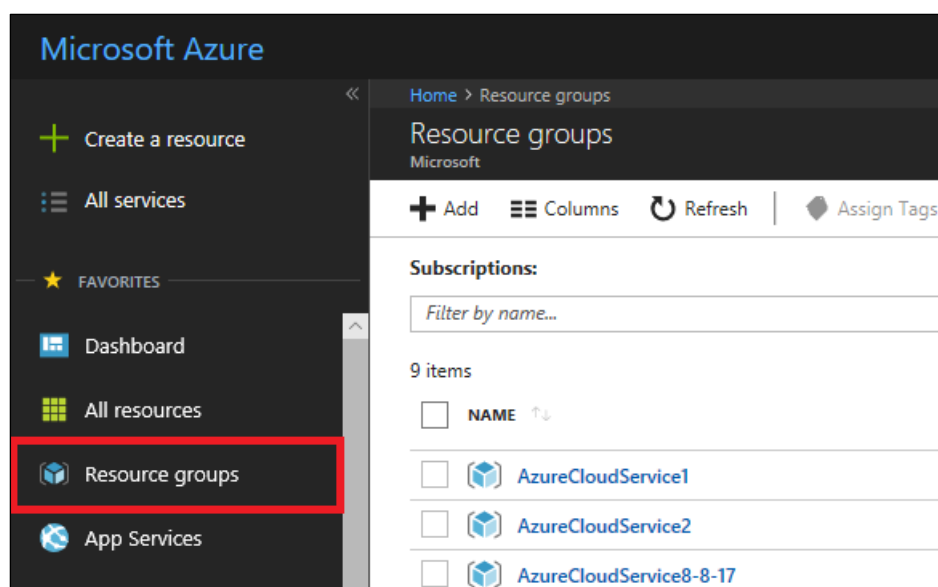
Once the pre-requisites are in place, the next step would be the installation of the ARM script. The steps for installing the ARM script are given below:

4.1.1. Step 1: Clone Repository

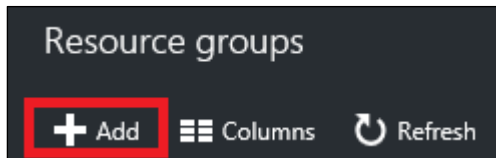
- Clone repository "MobiliyaFleetARMTemplate.git"
(<https://github.com/MobiliyaTechnologies/MobiliyaFleetARMTemplate.git>)

4.1.2. Step 2: Create Resource Group

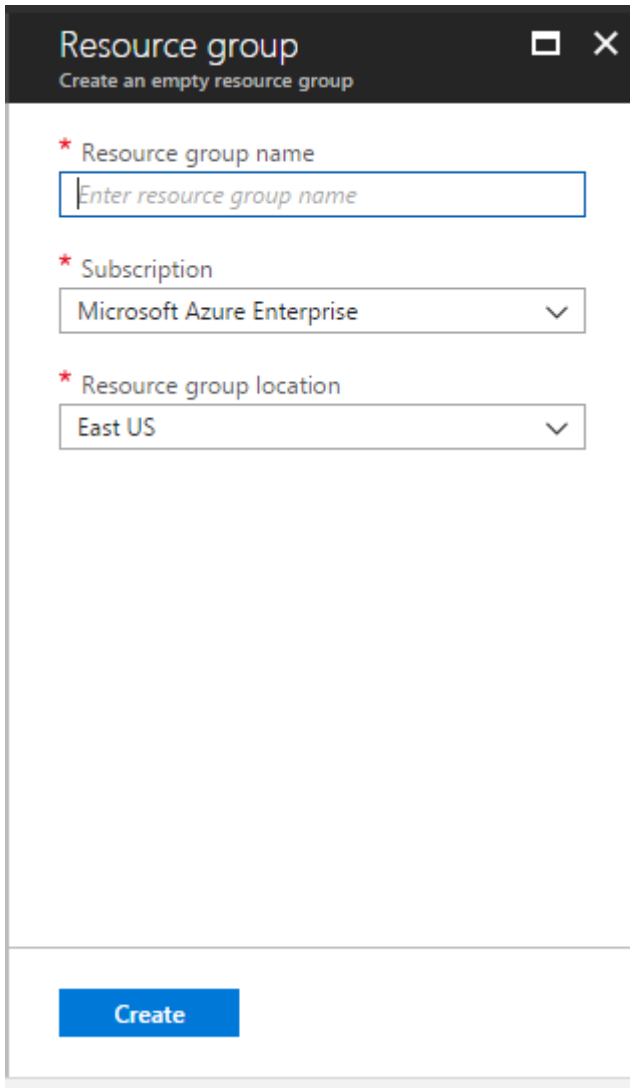
- Login to Azure portal (<https://portal.azure.com>) and select the appropriate subscription if it is not selected by default. Create a resource group that serves as the container for the deployed resources.
- Click on **Resource Groups** from the left menu to create a resource group.



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



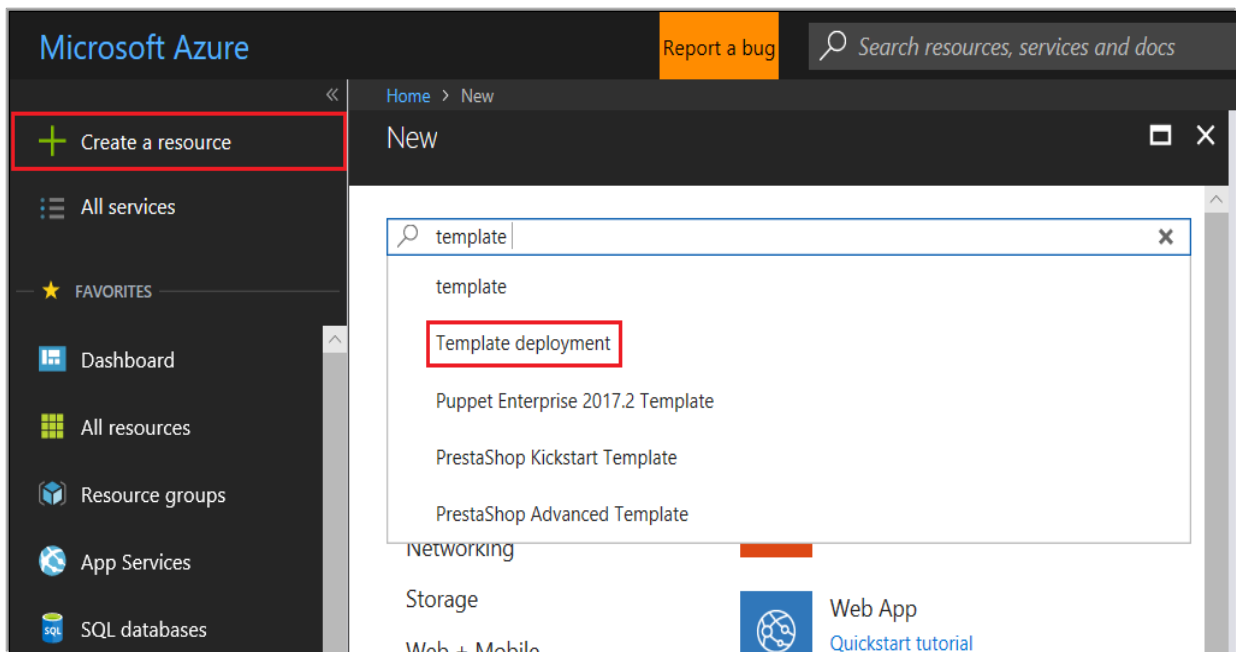
Provide a name and location for the new resource group. Click on Create.



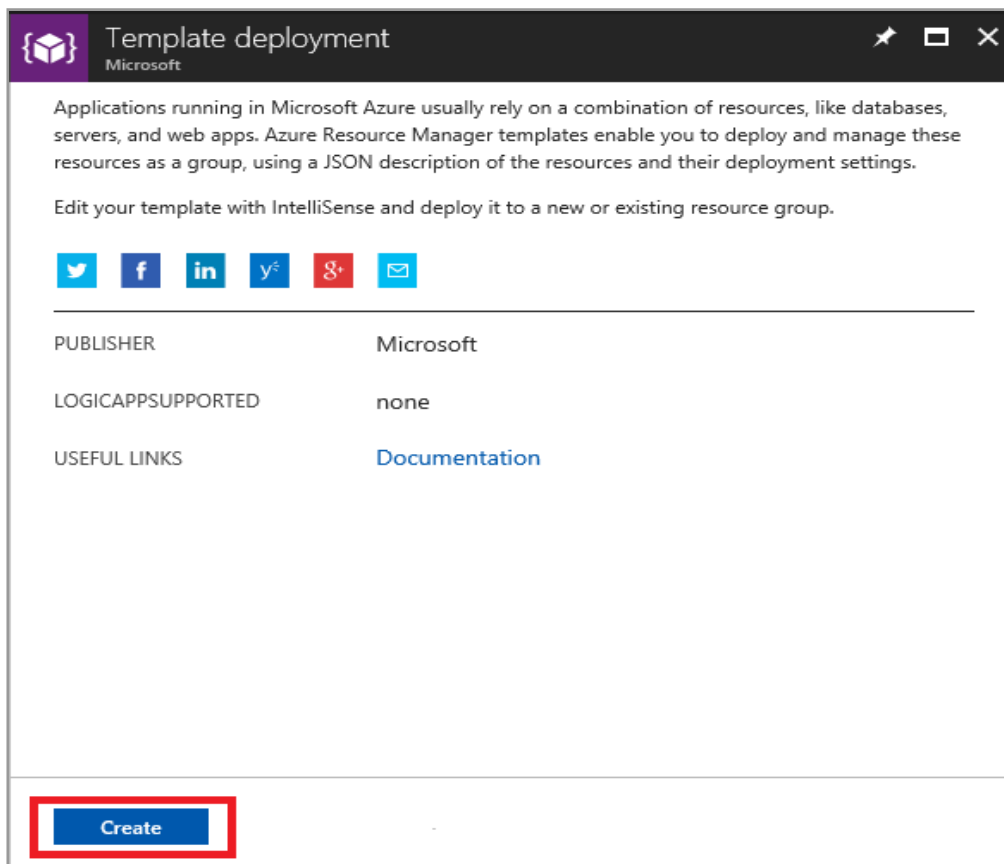
4.1.3. Step 3: Create Resources Using Template

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

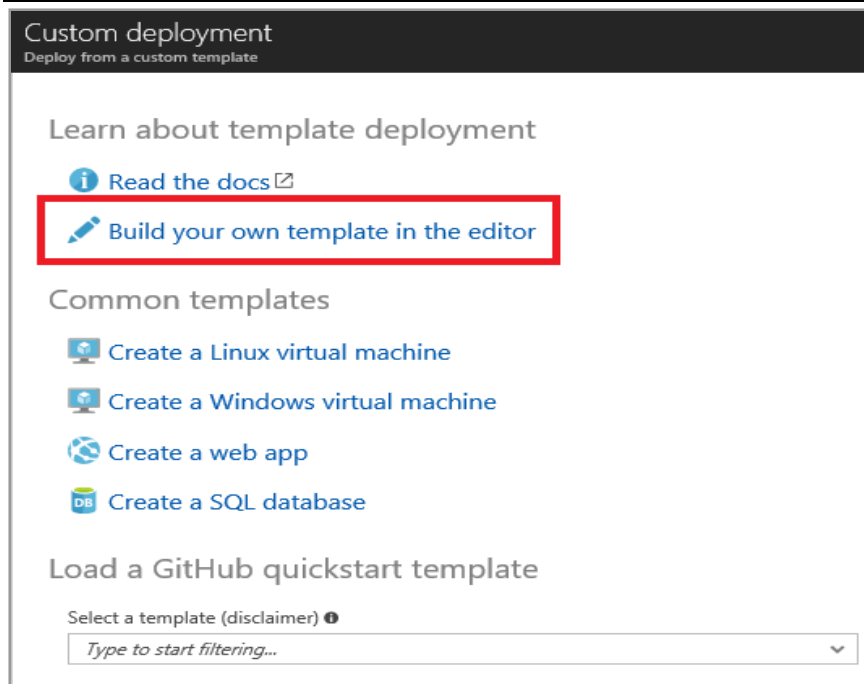
- Click on **+Create a resource** from the left menu.
- Select **Template deployment option**
- load the azuredeploy.json template from your cloned project directory('c:\users\..\projects').



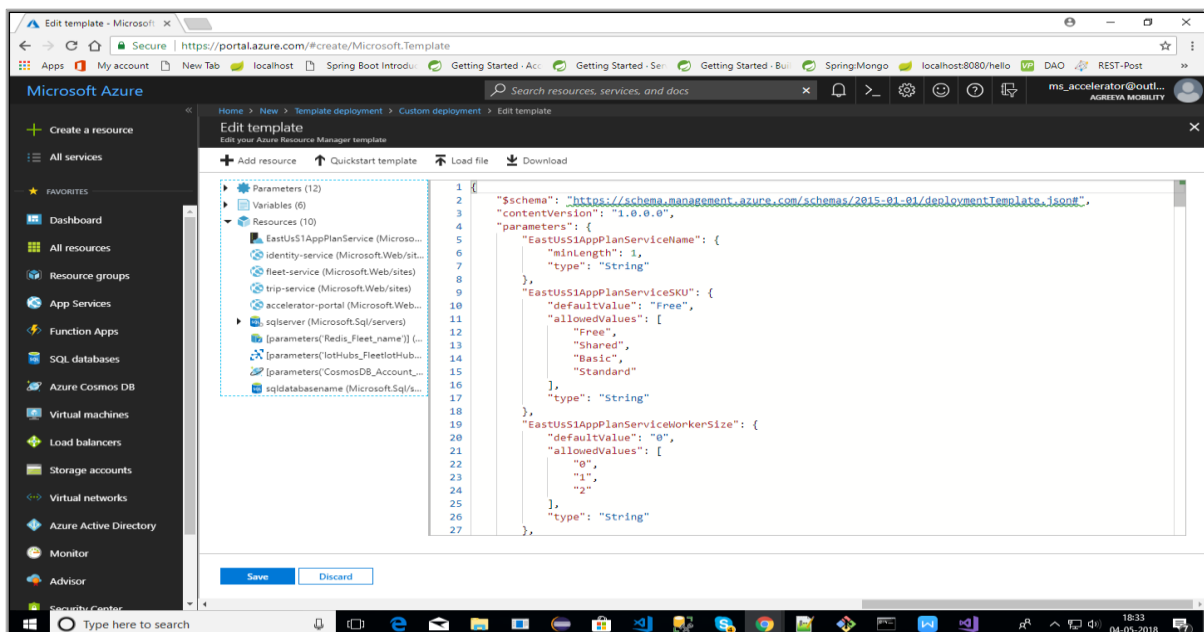
- Click on **Create**



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



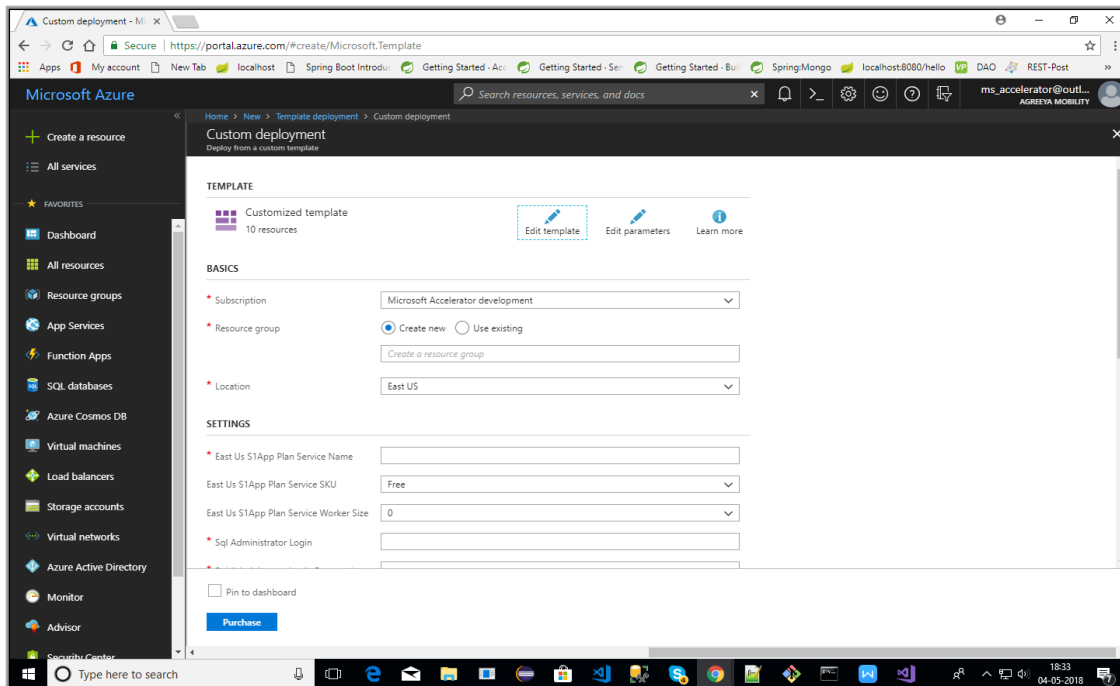
- Click on **Load file** located in the top navigation bar. Upload deployment template file (azuredeploy.json) from your cloned project in step 1.



- 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"**



- * 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

- Names of the resources will be those that you have provided in the form
- Now you can proceed with the further deployment steps.

5. Deployment of Backend Services on Azure:

The following steps will enable user in deploying the Web App

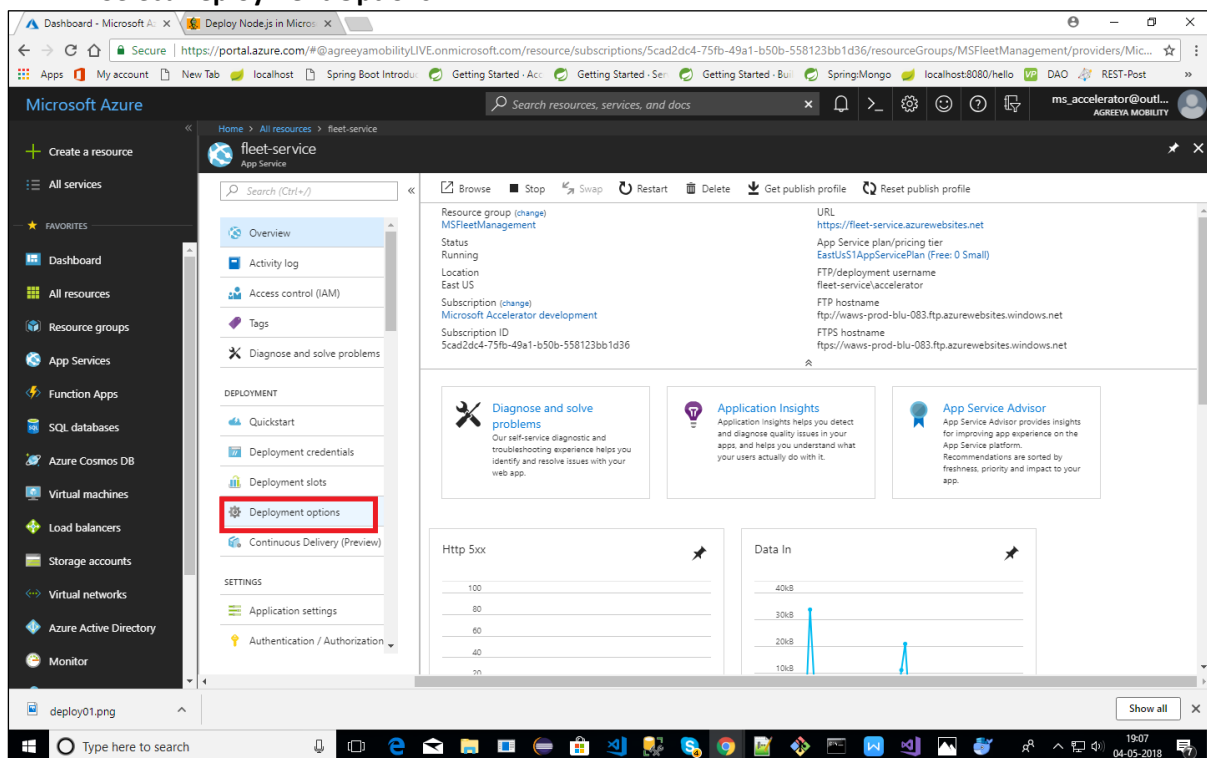
5.1. Pre-requisites:

Download the Fleet Management Project from github on your local computer using below URLs:

- **Identity Service:** <https://github.com/MobiliyaTechnologies/identityService.git>
- **Fleet Service:** <https://github.com/MobiliyaTechnologies/FleetService.git>
- **Trip Service:** <https://github.com/MobiliyaTechnologies/TripService.git>

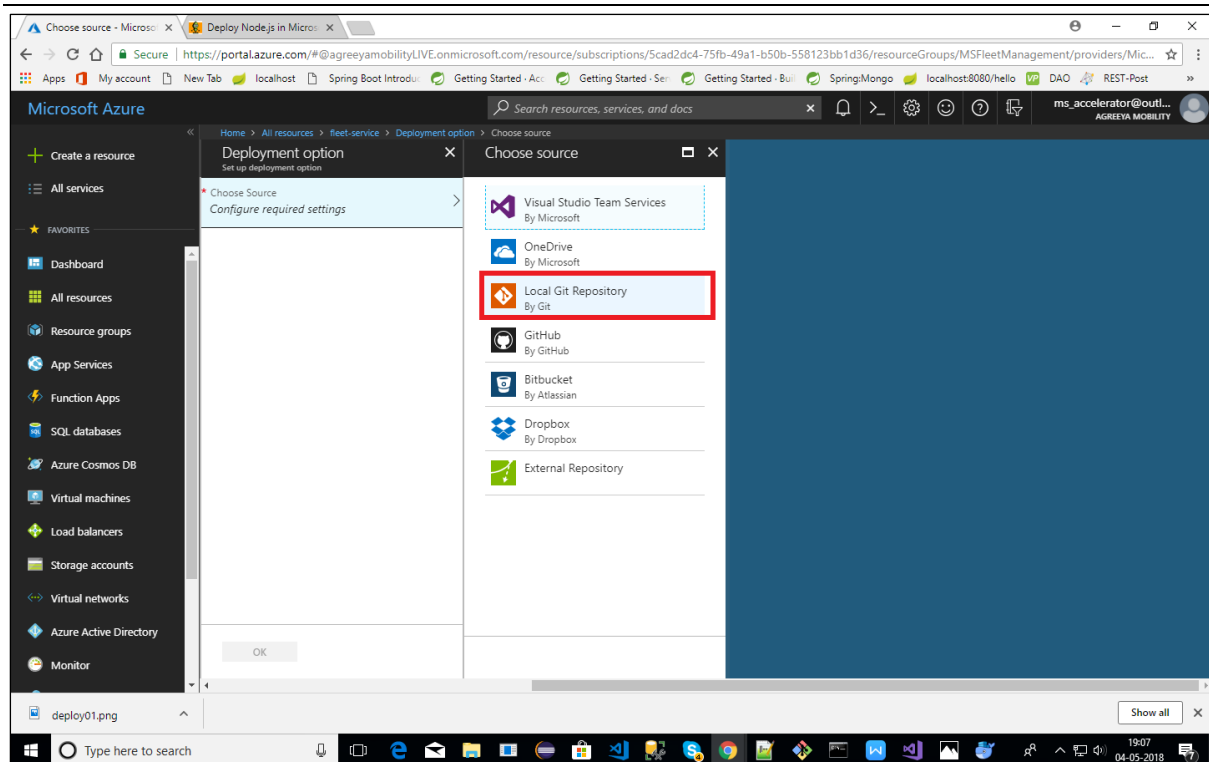
5.1.1. Step 1: Select Deployment Option

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



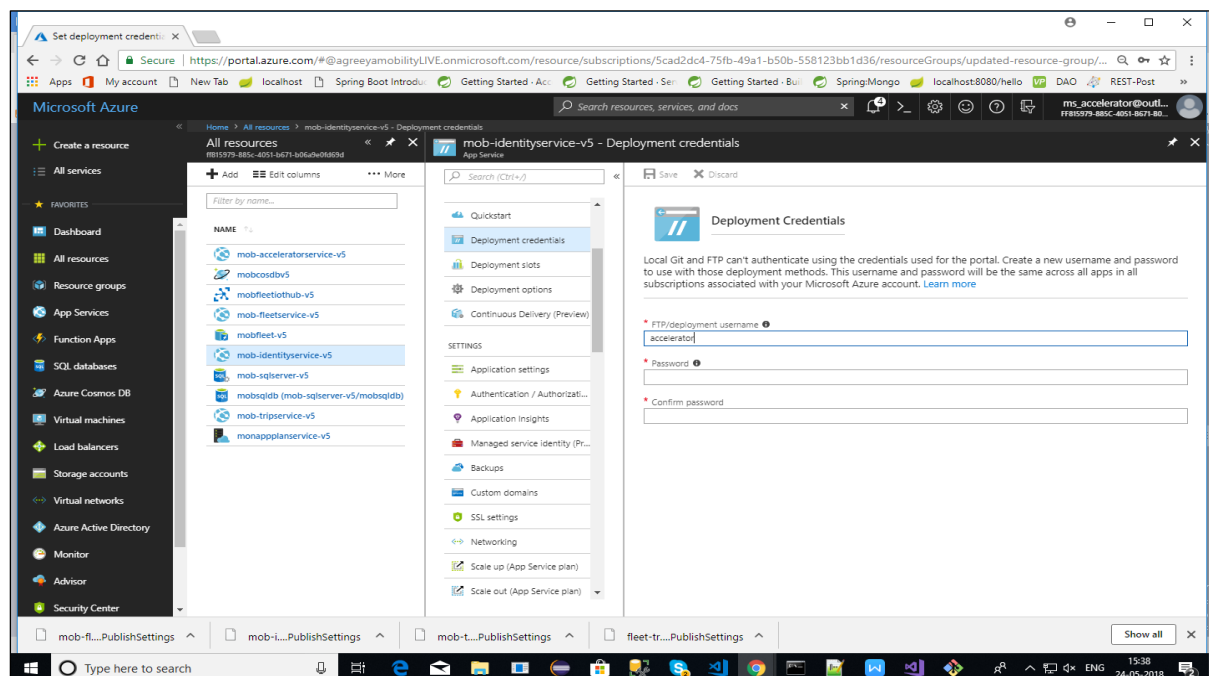
5.1.2. Step 2: Set Deployment Source- Local Git

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



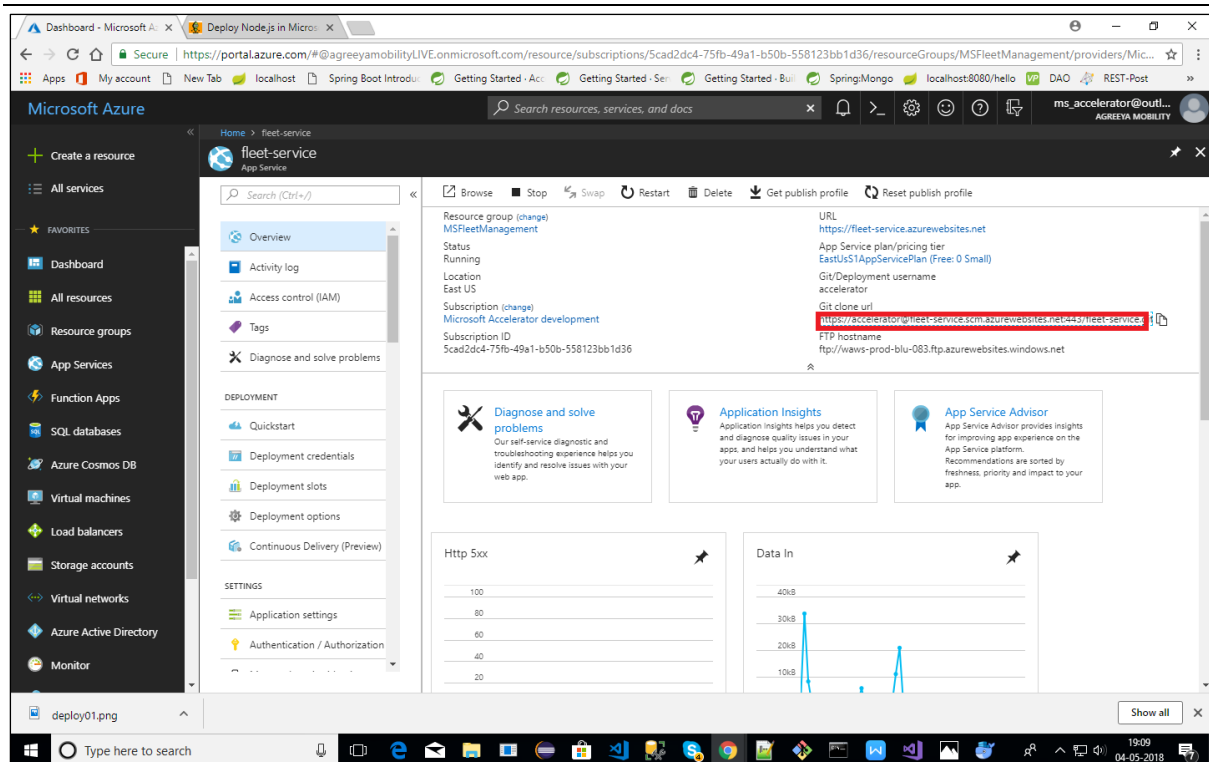
5.1.3. Step 3: Set Deployment Credentials

- Set Deployment Credentials.



5.1.4. 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.



5.1.5. Step 5: Repeat Steps For Other Services

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

5.1.6. 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".
```

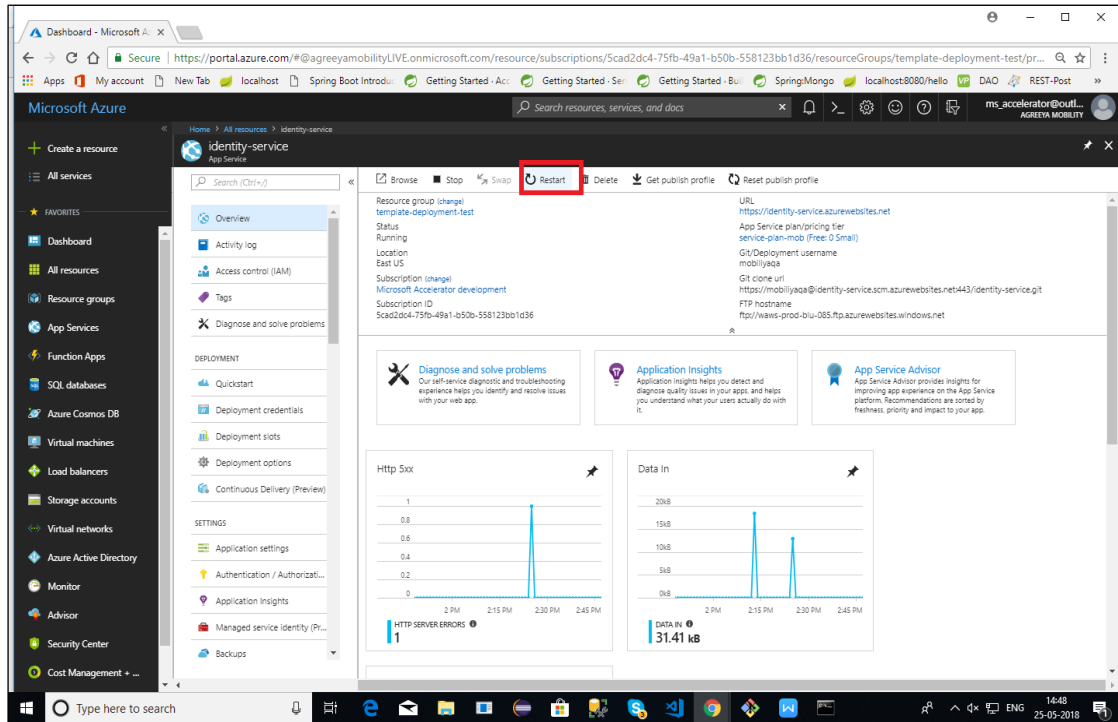
Push the changes to Azure git repository using the command given below:

```
$ git push origin master
```

- The application would be deployed.

5.1.7. Step 7: Restart Application

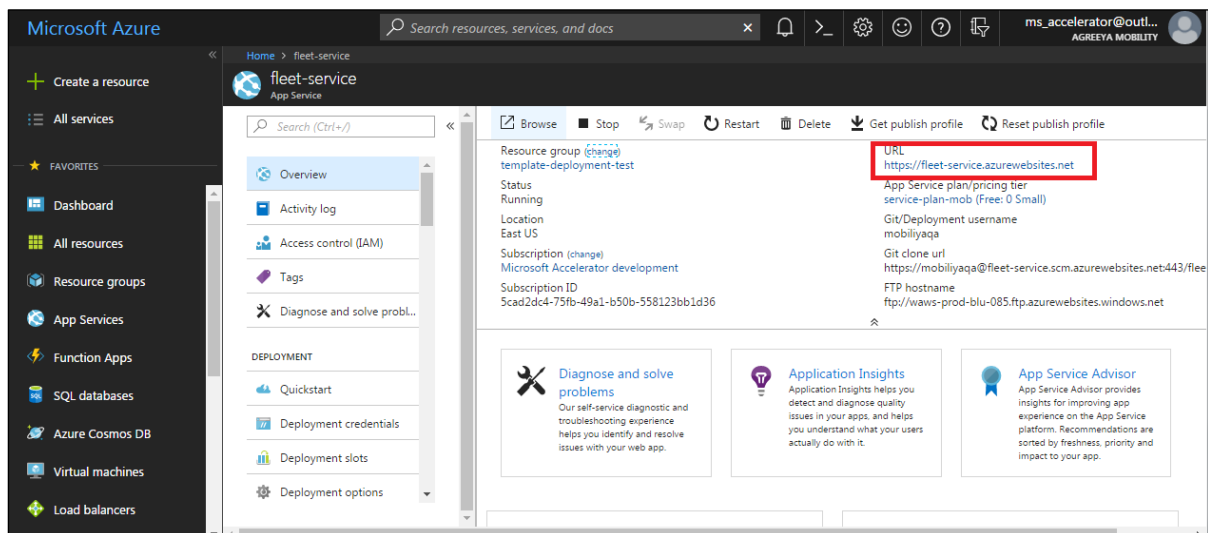
- On Azure portal, Restart your application.



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

5.1.8. Step 8: Verify Service is Working

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



6. Deployment of Web Portal on Azure

6.1. Setup and Configuration changes

6.1.1. Pre-requisites

- You need to have Git and Node.js installed.

6.1.2. Clone the repository

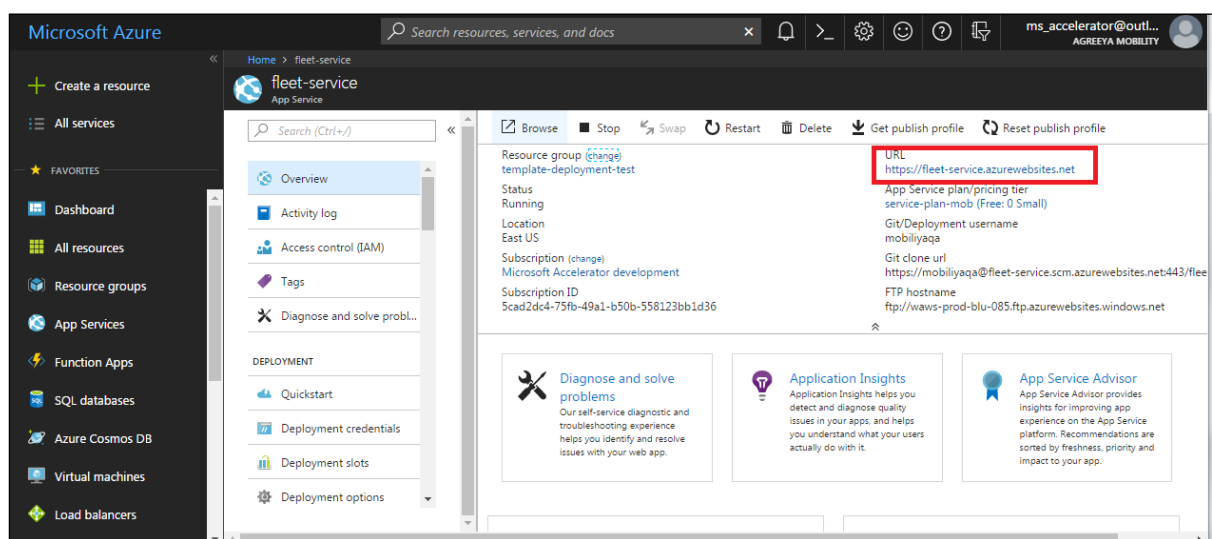
- You have to clone the repository “MobiliyaFleetWebPortal”.
(<https://github.com/MobiliyaTechnologies/MobiliyaFleetWebPortal.git>)

6.1.3. Environment/config changes

- This step assumes that you have deployed backend services and that they are up and running.

You need to change configuration URLs in ‘environment.ts’ file (located at /src /environments / environment.ts), change SERVICE_URL values.

Note: You can find service URL on azure portal->app service overview (the image given below is for fleet service)



- USER (line no 5): set identity service URL
- FLEET (line no 6): set fleet service URL
- TRIP (line no 7): set trip service URL

6.1.4. Build and Run

To run your application open terminal/git bash in a folder you have cloned in step 2.

- Install the npm packages described in package.json using the command given below:

`npm install`

- You can run the application using the command given below:

`npm run ng serve`

Application default runs on 4200 port. Open <http://localhost:4200> in browser.

iii. Once the application is tested locally, you are ready to deploy your application.

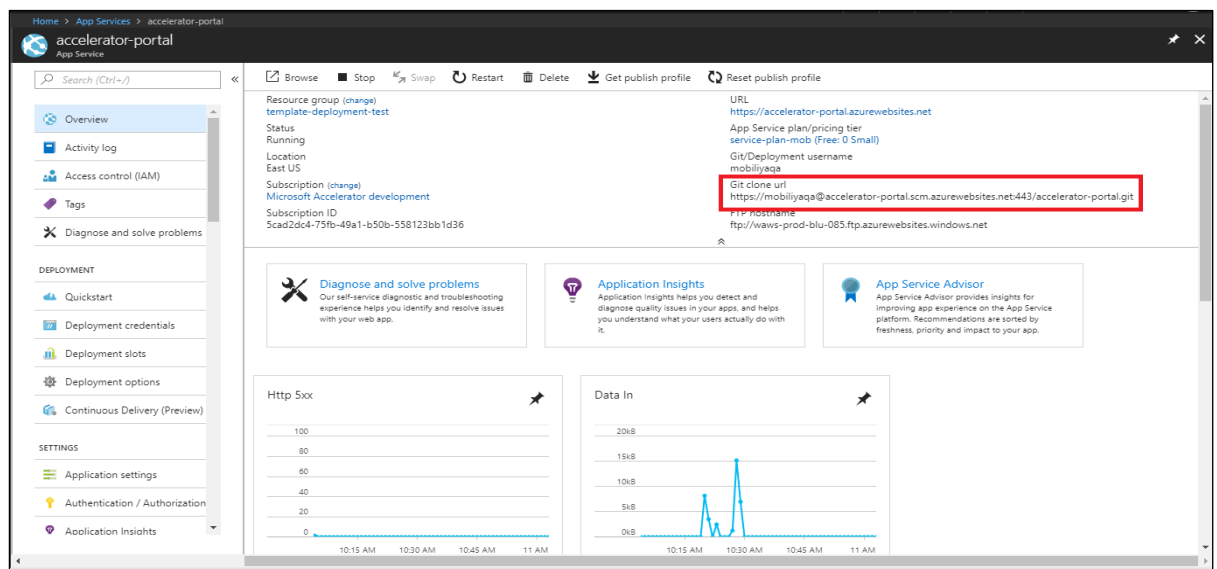
iv. Build the application using the command given below (Generates dist folder)

```
npm run ng build
```

6.1.5. Deploy:

To deploy your application, follow the steps given below:

- i. 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.
- ii. Clone the local git repository of azure app service (accelerator-portal) using URL as shown in the image given below:

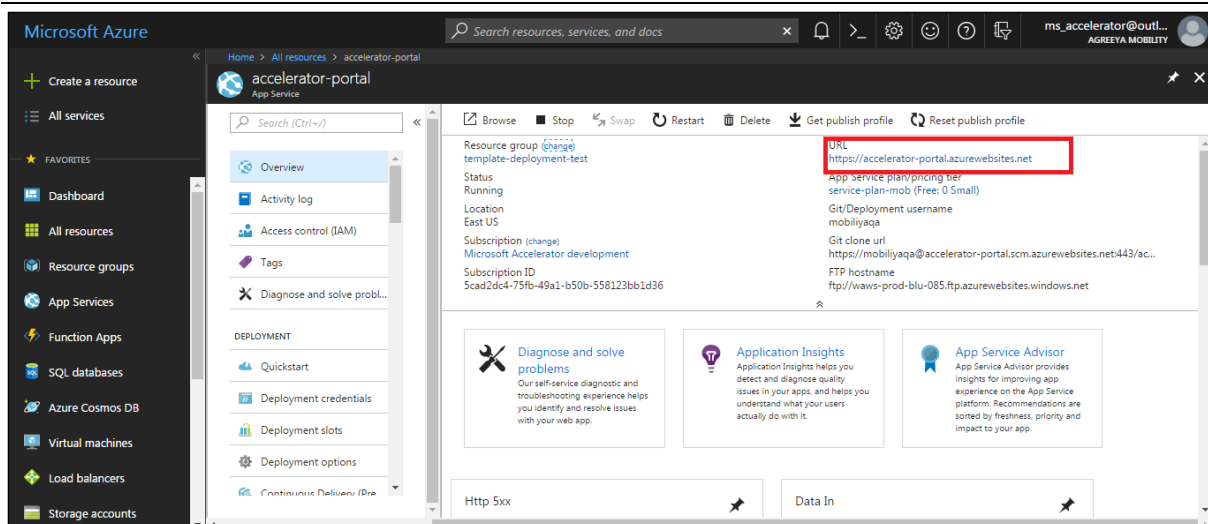


iii. Go to source folder (cloned in step 2). Copy files from dist folder into newly cloned repository in above step.

iv. You need to push your code using commands given below:

- `git add .`
- `git commit -m "message-string"`
- `git push origin master`

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



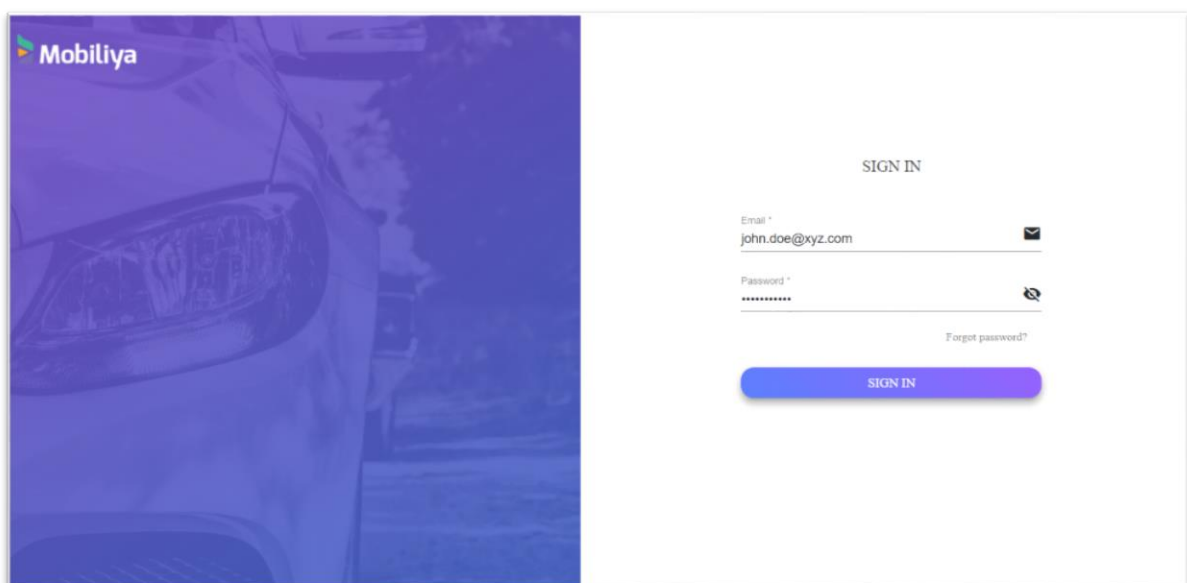
6.2. Tenant Admin

- To start using the system login as a Tenant Admin, user must follow the steps given below:

6.2.1. Login

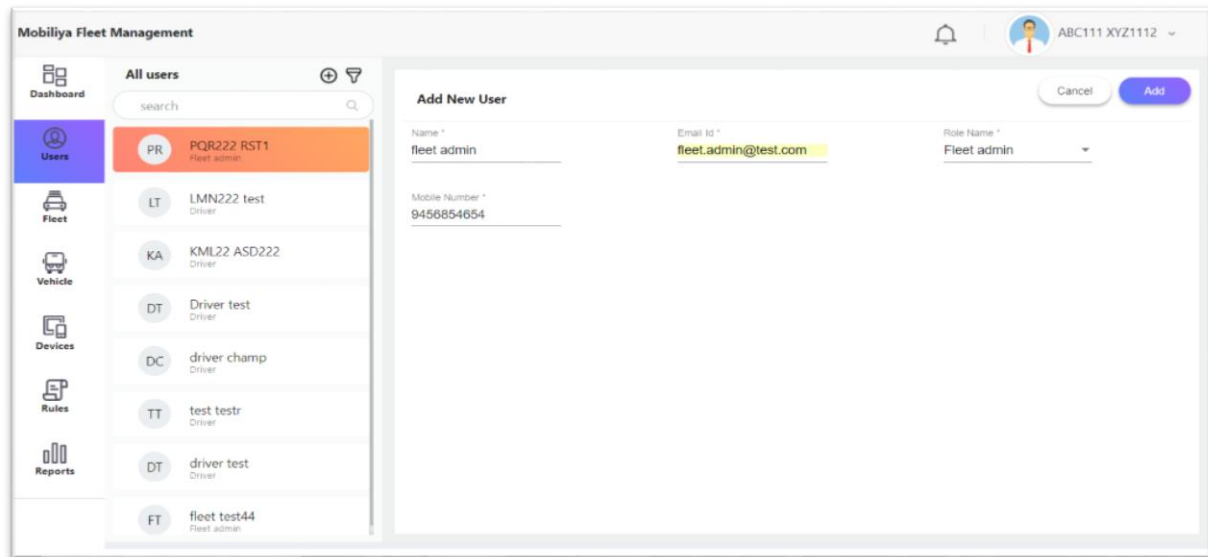
- Tenant admin account is created by default. You need to open accelerator portal URL in your browser, enter your username and password and click on Sign In button.

Note : Default tenantadmin username: tenantadmin@mobiliya.com, password: welcome

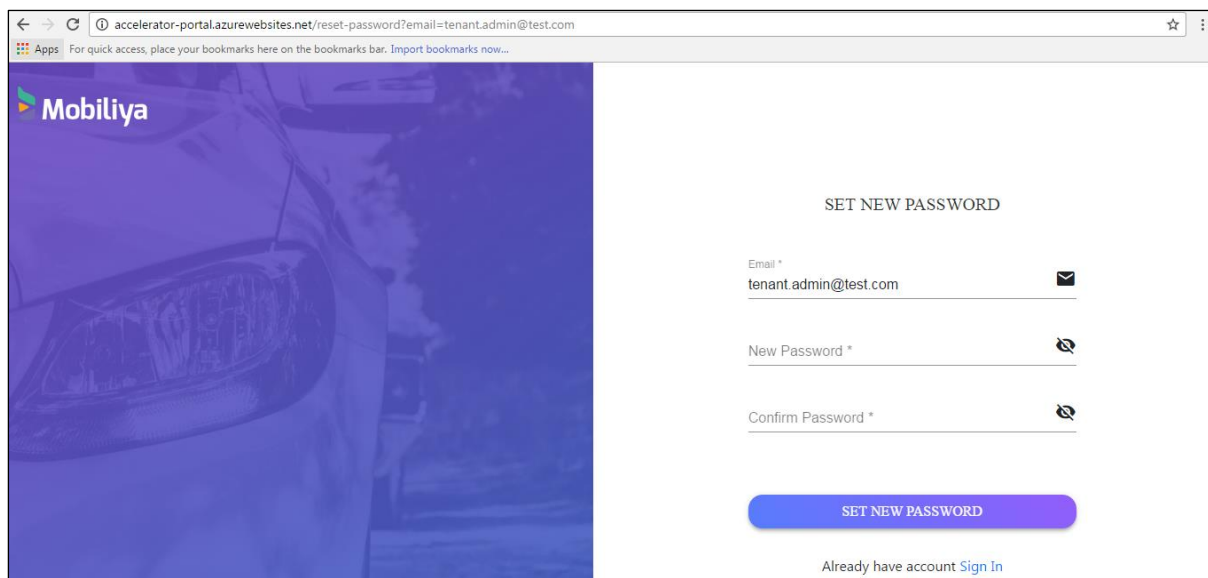


6.2.2. Add User

- On logging in, super admin is directed to the user-management page. Tenant admin can add fleet admin/driver. Click on **Add** button to add user.



- Once tenant Admin adds a user, the newly added user will receive an email containing a link to set new password.
- When the user opens that link, it will prompt to set a new password on a screen as shown below



6.2.3. Fleet

Tenant admin can add, edit or delete fleets.

6.2.4. Vehicles

Tenant admin can add, edit or delete vehicles. This section also contains trip information of vehicles.

6.2.5. Devices

Super admin can add, edit or delete devices. Devices are nothing but dongle which is associated with vehicle.

6.2.6. Reports

Tenant admin can view reports. Reports are generated in **Power BI**. Reports are based on vehicle that is selected. Reports include maximum of one-week data where date can be selected.

6.2.7. Rules

Tenant admin can create speeding or Geo-fencing rules and assign it to drivers within a fleet.

7. Android Application

7.1. Prerequisites

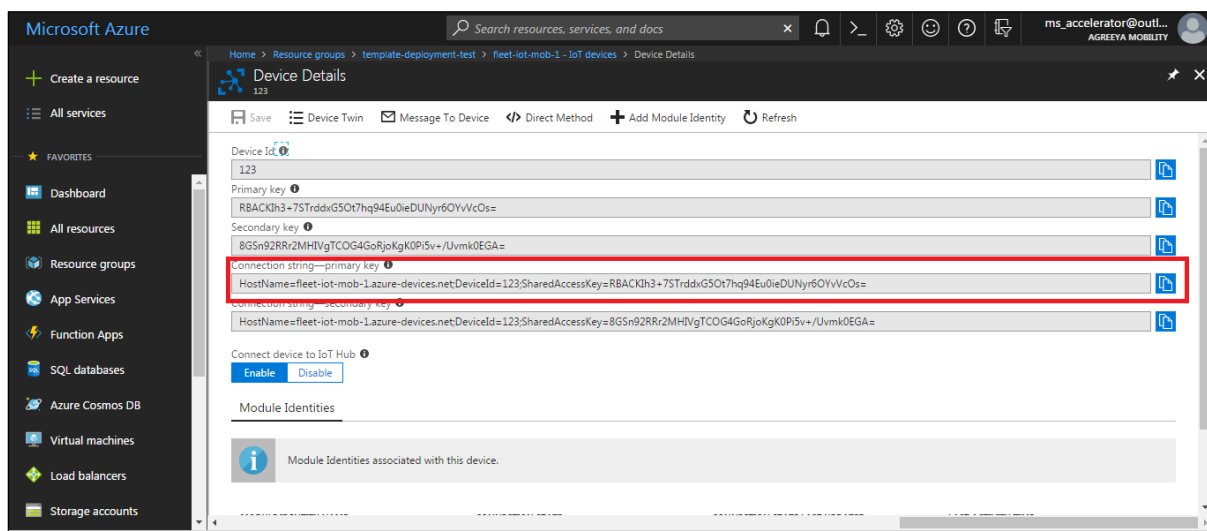
- Install Android Studio 3.2 and download Android SDK from <https://developer.android.com/studio/>
- Clone Android code from the command given below on git bash
 - [git clone https://github.com/MobiliyaTechnologies/MobiliyaFleetAndroid.git](https://github.com/MobiliyaTechnologies/MobiliyaFleetAndroid.git)
- Import this code in Android Studio by following the steps given below:
 - **File-> New-> Import project->path of the downloaded code**
- Build and run this code by following the steps given below:
 - **Build->Clean->Rebuild project, Run->Select android device**
- Device support - Android 6.0 Marshmallow and above

7.2. Configuration changes

Once the deployment of the IOT Hub is done user has to change the connection string in the android code file Constants.java(path \android-accelerator\src\main\java\com\mobiliya\fleet\utils\Constants.java), line number 14 "IOT_CONNSTRING".

To Get IoT Hub connection string follow below steps

On Azure portal, Go to Resource Groups -> <Resource group name> -> <iot hub name> -> IoT devices -> <device name>

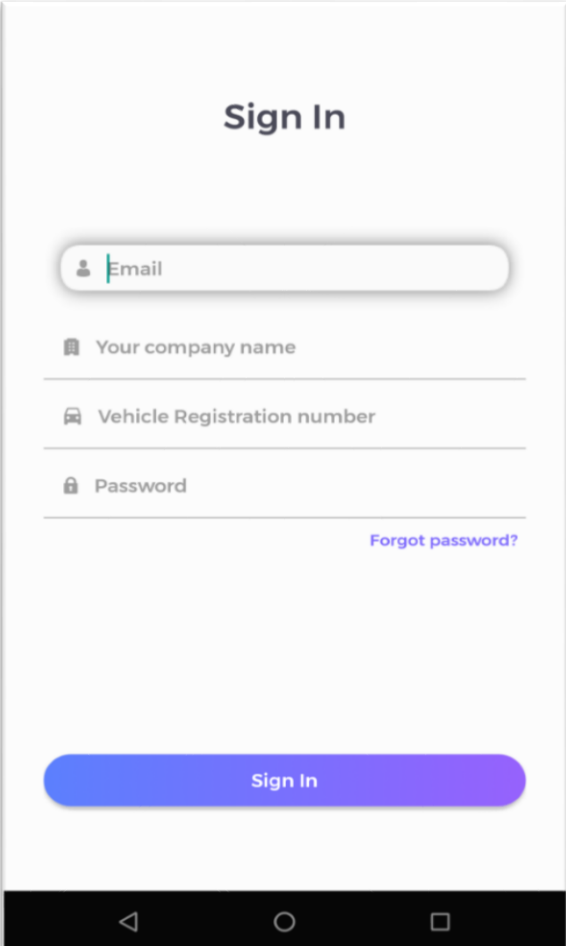


When URL for REST call changes, user has to change below strings from Constants.java file

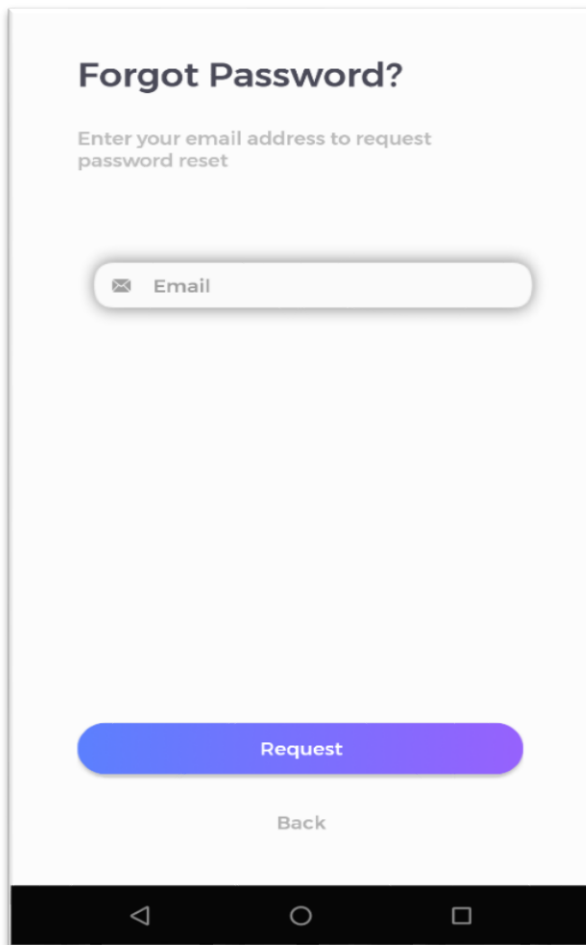
- "IDENTITY_DEV_URL" Line no 24,
- "FLEET_DEV_URL", Line no 25,
- "TRIP_DEV_URL" Line no 30

7.3. Driver Sign In

User can Sign In from here. User can change a password using Forgot Password option.



The image shows a mobile application screen titled "Sign In". It features four input fields: "Email" (with a person icon), "Your company name" (with a building icon), "Vehicle Registration number" (with a car icon), and "Password" (with a lock icon). Below the password field is a link labeled "Forgot password?". At the bottom of the form is a large blue button labeled "Sign In". The screen is displayed on a mobile device with a black navigation bar at the bottom.



7.4. Forgot Password

The Forgot Password option can be accessed from the Sign-In Screen. User can reset password from here in case user cannot recall the existing password.

7.5. Dashboard

After successfully logging in and connecting with the dongle, driver is redirected to the dashboard where he can manage trips and view his own ratings and overall vehicle health & condition.

