**PREDICTIVE MAINTENANCE FOR REMOTE FIELD**

**IOT DEVICES**

**GUHANESVAR M (1833015)**

**VENKETESZ RR (1833057)**

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

M.Sc. Decision and Computing Sciences

OF ANNA UNIVERSITY



**November 2022**

**DEPARTMENT OF COMPUTING**

**COIMBATORE INSTITUTE OF TECHNOLOGY**

**(Autonomous Institution affiliated to Anna University)**

**COIMBATORE – 641014**

**Pre-Requirements:**

1. Microsoft Azure subscription (non-Microsoft subscription, must be a pay-as-you subscription).
2. .NET Core 3.1
3. Visual Studio Code version 1.39 or greater
4. C# Visual Studio Code Extension
5. Azure Functions Core Tools version 2.x (using NPM or Chocolatey - see readme on GitHub repository)
6. Azure Functions Visual Studio Code Extension
7. An Azure Databricks cluster running Databricks Runtime 10.3 or above.
8. Node.js 8.0 or above

**Task 1: Download the resource files from git hub**

1. Download the code zip file from <https://github.com/guhan93/Predictive-Maintanence-gvoilpumps/tree/main>
2. Extract the zip file and store in in your desired location

**Task 2: Provision a resource group**

1. Log into the Azure Portal.
2. On the top-left corner of the portal, select the menu icon to display the menu.
3. The portal menu icon is displayed.
4. In the left-hand menu, select Resource groups. On its context menu, select Create.
5. Add Resource Group Menu
6. Create a new resource group with the name Gv\_Oilpumps, ensure the proper subscription and region nearest you are selected. Then select Review + Create.
7. Create Resource Group
8. On the Summary blade, select Create to provision your resource group.

**Task 3: Create an Azure Databricks service**

1. Navigate to the Azure portal.
2. Expand the left menu, and select + Create a resource, type in "Azure Databricks" in the search field, then select Azure Databricks from the results.
3. Azure Databricks is entered in the search field and Azure Databricks is selected.
4. Select Create in the Azure Databricks details page.
5. Within the Azure Databricks Service form, complete the following:

| **Field** | **Value** |
| --- | --- |
| Subscription | *select the appropriate subscription* |
| Resource Group | *select use existing, then Gv\_Oilpumps* |
| Workspace name | *globally unique name* |
| Location | *select the location nearest to you* |
| Pricing tier | *select Standard* |

1. Select Review + Create. On the review screen, select Create.

**Task 4: Create Azure Databricks cluster**

1. In the Azure portal, open your Azure Databricks service you created in the previous task.
2. Select Launch Workspace. Azure Databricks will automatically sign you in through its Azure Active Directory integration.
3. Once in the workspace, select Compute in the left-hand menu, then select Create Cluster.
4. In the New Cluster form, specify the following configuration options:

| **Field** | **Value** |
| --- | --- |
| Cluster name | *enter lab* |
| Cluster Mode | *select Standard* |
| Pool | *select None* |
| Databricks Runtime Version | *select Runtime: 10.3 (Scala 2.12, Spark 3.2.1)* |
| Autopilot Options | *uncheck Enable autoscaling and check Terminate after..., with a value of 120 minutes* |
| Worker Type | *select Standard\_DS3\_v2* |
| Driver Type | *select Same as worker* |
| Workers | *enter 1* |

1. Select Create Cluster.

**Task 5: Import lab notebook into Azure Databricks**

1. Within your Azure Databricks service, select Workspace, select Users, select the dropdown to the right of your username, then select Import.
2. Keep File selected next to Import from, select or drag and drop the file Anomaly Detection.ipynb file
3. The file import form is displayed with the Anomaly Detection.ipynb file selected.
4. After importing the notebook, select Workspace, then select Users. Select your username. You will see a new notebook named Anomaly Detection.
5. The imported notebooks are displayed.

**Task 6: Create Azure Machine Learning service workspace**

1. Navigate to the Azure portal.
2. Select + Create a resource, type in "machine learning" in the search field, then select Machine Learning from the results.
3. Create a resource is highlighted and Machine Learning is selected.
4. Select Create in the Machine Learning details page.
5. Within the Machine Learning form, complete the following:

| **Field** | **Value** |
| --- | --- |
| Subscription | *select the appropriate subscription* |
| Resource Group | *select Gv\_oilpumps* |
| Workspace name | *globally unique name* |
| Region | *select the region nearest to you* |
| Storage account | *keep the default* |
| Key vault | *keep the default* |
| Application insights | *keep the default* |
| Container registry | *None* |

1. Select Review + Create. On the review screen, select Create.

**Task 7: Model the telemetry data**

The telemetry being used for data simulation are as follows, we will be using this information for

Simulating data for the rod pumps:

**Telemetry schema:**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| SerialNumber | String | Unique serial number identifying the rod pump equipment |
| IPAddress | String | Current IP Address |
| TimeStamp | DateTime | Timestamp in UTC identifying the point in time the telemetry was created |
| PumpRate | Numeric | Speed calculated over the time duration between the last two times the crank arm has passed the proximity sensor measured in Strokes Per Minute (SPM) - minimum 0.0, maximum 100.0 |
| TimePumpOn | Numeric | Number of minutes the pump has been on |
| MotorPowerkW | Numeric | Measured in Kilowatts (kW) |
| MotorSpeed | Numeric | including slip (RPM) |
| CasingFriction | Numeric | Measured in PSI (psi) |

**Task 8: Create an IoT Central application**

1. Navigate to the Azure portal.
2. Expand the left menu, and select + Create a resource, type in "IoT Central" in the search field, then select IoT Central application from the results.
3. On the IoT Central application resource overview screen, select Create.
4. In the IoT Central Application, fill out the form with the following settings:

| **Field** | **Value** |
| --- | --- |
| Resource name | Enter a globally unique name. |
| Application URL | Keep the default. |
| Subscription | Select the appropriate subscription. |
| Resource group | Select Gv\_oilpumps. |
| Pricing plan | Select Standard 2. |
| Template | Select Custom application. |
| Location | Select the region nearest to you. |

1. Select Create.
2. Wait for the application to be provisioned.

**Task 9: Create the Device Template**

1. In the Azure Portal, open the Gv\_oilpumps resource group. Select the IoT Central Application resource from the listing.
2. From the IoT Central Application Overview screen, select the IoT Central Application URL. This opens the IoT Central application in a new tab in your browser.
3. We need to define the type of equipment we are using, and the data associated with the equipment. In order to do this, we must define a Device Template. Select the Device Templates menu item from the left-hand menu. Then select + New from the toolbar menu.
4. On the Select type screen, select IoT device as the custom device template type. Select the Next: Customize button.
5. On the Customize form, for the device template name, enter Rod Pump. Keep the Gateway device checkbox unchecked. Select the Next: Review button.
6. On the Review screen, select the Create button.
7. On the Rod Pump device template screen, select Custom model beneath the Create a model heading.
8. On the Rod Pump Model screen, select the Model parent item in the central navigation pane. Select Add capability to default component from beneath the Capabilities heading.
9. The Capabilities of the device model describe the telemetry expected from the device, the commands it responds to, and its properties (device twin properties). We'll begin defining the Telemetry values, populate the capabilities as described in the following table. Use the + Add capability button at the bottom of the screen to add additional telemetry values. Select the Save button from the toolbar once complete. You will need to expand the capability form to edit the Schema, Unit, Display Unit and Description fields.

| **Display Name** | **Name** | **Capability Type** | **Semantic Type** | **Schema** | **Unit** | **Display Unit** | **Description** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pump Rate | PumpRate | Telemetry | None | Double | None | SPM | Speed calculated over the time duration between the last two times the crank arm has passed the proximity sensor measured in Strokes Per Minute (SPM) |
| Time Pump On | TimePumpOn | Telemetry | Time Span | Double | Minute | Minutes | Number of minutes the pump has been on |
| Motor Power | MotorPowerKw | Telemetry | Power | Double | None | kW | Measured in Kilowatts (kW) |
| Motor Speed | MotorSpeed | Telemetry | Angular Velocity | Integer | Revolution per minute | RPM | including slip (RPM) |
| Casing Friction | CasingFriction | Telemetry | None | Double | None | PSI | strokes per minute |

1. We also need to define the current state of the pump, whether it is running or not. Remaining on the same screen, Select + Add capability. Add the state with the display name of Power State, field name of PowerState, capability type of Telemetry and semantic type of State. For Value schema, select String then add the values Unavailable, On, and Off (in each the Display name, Name and Value columns). Select the Save button from the toolbar menu.
2. In the device template, Properties are metadata associated with the equipment. These are added on the same form as the capabilities, only using Property for the Capability type field. For our template, we will expect a property for Serial Number, IP Address, and the geographic location of the pump. Remaining on the same screen, select + Add capability. Define the device properties as follows, then select Save from the toolbar menu:

| **Display Name** | **Name** | **Capability Type** | **Semantic Type** | **Schema** | **Writable** | **Description** |
| --- | --- | --- | --- | --- | --- | --- |
| Serial Number | SerialNumber | Property | None | String | Off | The Serial Number of the rod pump |
| IP Address | IPAddress | Property | None | String | Off | The IP address of the rod pump |
| Pump Location | Location | Property | Location | Geopoint | Off | The geo. location of the rod pump |

1. Operators and field workers will want to be able to turn on and off the pumps remotely. In order to do this, we will define a command. Remaining on the same screen, select + Add capability. Select the Commands tab and select the New button to add a new command. Create a command as follows, and select Save from the toolbar:
   * Display Name - Toggle Motor Power
   * Name - ToggleMotorPower
   * Capability Type - Command
   * Request - On
   * (Request) Display name - Toggle
   * (Request) Name - Toggle
   * (Request) Schema - Boolean
   * Description - Toggle the motor power of the pump on and off.
2. Now, we can define device specific views to help us visualize telemetry and state of the Rod Pumps. We can also create forms to allow pump operators to execute commands on a device. From the centre pane navigation menu of the Rod Pump device template, select the Views item. In the Select to add a new view screen, select the Visualizing the device card.
3. A View is composed of one or more tiles that display information related to a specific device. In the Create view form, set the View name to Dashboard. Underneath Add a tile, select Start with devices. Then expand the Telemetry drop down and select each item individually, then choosing the Add tile button to add the chart to the View surface. Feel free to arrange the tiles as desired on the View design surface.
4. Remaining on the Dashboard view, expand the Property drop down list, and add a tile for IP Address, Serial Number, and Pump Location. Note that the tile for Pump Location renders with a map icon, meaning that IoT Central has identified the property as geography-based data, and will render it on a map appropriately.
5. A portion of the Dashboard View design surface is shown with tiles added for various properties of the device.
6. Spend some time now and investigate the various visualizations and settings you can set on each tile. For instance, you have the ability to customize chart types, colors, and axes. You can also resize each tile individually. Select the Save button in the toolbar menu to save the Dashboard view.
7. On the Rod Pump device template screen, select the Views item from the central navigation pane, and choose Visualizing the device once again to create a new view. Name this view, Command and add a tile for the Toggle Motor Power command. Once complete, press the Save button in the toolbar. This View will allow pump operators to initiate the toggle power command from the IoT Central application.
8. Finally, we can add a thumbnail image to represent the equipment. Select Device templates, then select Rod Pump. Select the circle icon to left of the template name. This will allow you to select an image file. After setting the thumbnail, select the Publish button in the device template toolbar. Finally, select Publish to publish the device template.

**Task 10: Create and provision real devices**

1. In the left-hand menu of your IoT Central application, select Devices. Select the Rod Pump template from the Devices blade and select the + New button from the toolbar to add a new device.
2. A modal window will be displayed with an automatically generated Device ID and Device Name. You are able to overwrite these values with anything that makes sense in your downstream systems. We will be creating three real devices in this lab. Create the following as real devices, ensure Simulate this device remains toggled off:

| **Device ID** | **Device Name** |
| --- | --- |
| DEVICE001 | Rod Pump - DEVICE001 |
| DEVICE002 | Rod Pump - DEVICE002 |
| DEVICE003 | Rod Pump - DEVICE003 |

1. On the Devices list, notice how all three real devices have the provisioning status of Registered.

**Task 11: Run the Rod Pump Simulator**

1. In IoT Central, select Devices from the left-hand menu. Then, from the devices list, select the link for Rod Pump - DEVICE001, and select Connect located in the toolbar.
2. The device screen for DEVICE001 is displayed with the Connect button highlighted.
3. A Device connection modal is displayed, make note of the ID Scope, Device ID, as well as the primary key value.
4. The device connection key information is displayed.
5. Repeat steps 1 and 2 for DEVICE002 and DEVICE003.

**Task 12: Open the Visual Studio solution, and update connection string values**

1. Using Visual Studio Code, open the downloaded Github code folder
2. Open the appsettings.json file, and copy & paste the ID Scope and Device Primary Key values into the file.

**Task 13: Run the application**

1. Within Visual Studio Code, expand the .vscode sub-folder, then open launch.json. Update the console setting to externalTerminal. This will cause the debugger to launch the console window into an external terminal instead of within Visual Studio Code. This is a required step since the internal terminal does not support entering values (ReadLine).
2. Using Visual Studio Code, Debug the current project by pressing F5.
3. Once the menu is displayed, select option 1 to generate and send telemetry to IoT Central.
4. The command prompt displays the choice to generate and send telemetry to IoT Central.
5. Allow the simulator to start sending telemetry data to IoT Central, you will see output similar to the following:
6. A command prompt shows a few provisioned devices and their corresponding measurements.
7. Allow the simulator to run while continuing with this lab.
8. After some time has passed, in IoT Central select the Devices item in the left-hand menu. Note that the provisioning status of DEVICE001, DEVICE002, and DEVICE003 now indicate Provisioned.

**Task 14: Create an Event Hub and continuously export data from IoT Central**

1. Log into the Azure portal, and open your Gv\_oilpumps resource group.
2. On the top of the screen, select the Create button. When the marketplace screen displays, search for and select Event Hubs. This will allow you to create a new Event Hub Namespace resource. Select the Create button on the resource overview screen.
3. Configure the event hub as follows, select the Review + create\* button, and then Create

|  |  |
| --- | --- |
| **Field** | **Value** |
| Subscription | *select the appropriate subscription* |
| Resource Group | Gv\_oilpums |
| Name | *anything (must be globally unique)* |
| Location | *select the location nearest to you* |
| Pricing Tier | Standard |

1. Once the Event Hubs namespace has been created, open it and select the + Event Hub button at the top of the screen.
2. In the Create Event Hub form, configure the hub as follows and select the Create button.
3. From the top menu, select the + Consumer Group button to create a new consumer group for the hub. Name the consumer group ingressprocessing and select the Create button.

**Task 15: Configure continuous data export from IoT Central**

1. Return to the IoT Central application, from the left-hand menu, select Data export. Then select the + New button from the toolbar menu.
2. Begin configuring the data export with the following values:

| **Field** | **Value** |
| --- | --- |
| Display Name (Header) | Event Hub Feed |
| Enabled (Header) | **On** |
| Type of data to export | **Telemetry** |

1. In the Destinations section, select the create a new one link. Configure the new destination as follows, then select the Create button:

| **Field** | **Value** |
| --- | --- |
| Destination name | iot-central-event-hub-feed |
| Destination type | **Azure Event Hubs** |
| Connection string | see subsection below on how to get the connection string |
| Event Hub | iot-central-feed |

Obtain the connection string as follows:

* Navigate to your Event Hubs namespace in the Azure portal.
* Select Shared access policies on the left-hand menu, then select the RootManageSharedAccessKey and copy the Connection string-primary key.
* Return to IoT Central and paste the connection string into the Connection string field, then select the iot-central-feed event hub you created.

1. Select Save from the toolbar menu on the Event Hub Feed continuous export screen.
2. The Event Hub Feed export will be created, and then started (it may take a few minutes for the export to start). Return to the Data export list to see the current status of the feed.

**Task 16: Use Azure Databricks and Azure Machine Learning service to train and deploy predictive model**

1. In the Azure portal, open your lab resource group, then open your Azure Databricks Service.
2. Select Launch Workspace. Azure Databricks will automatically sign you in through its Azure Active Directory integration.
3. In Azure Databricks, select Workspace, select Users, then select your username.
4. Select the Anomaly Detection notebook to open it.
5. Before you can execute the cells in this notebook, you must first attach your Databricks cluster. Expand the dropdown at the top of the notebook where you see Detached. Select your lab cluster to attach it to the notebook. If it is not currently running, you will see an option to start the cluster.
6. Provide values in the Cmd 56 cell. You can find your Machine Learning workspace information from the resource group details in the Azure portal.
7. The screenshot displays the resources in our resource group, highlighting the Machine learning resource.
8. Copy the scoring web service URL from the last cell's result after executing it. You will use this value to update a setting in your Azure function in the next exercise to let it know where the model is deployed.

**Task 17: Create an Azure Function to predict pump failure**

1. Return to the Azure Portal.
2. Open your resource group for this lab.
3. From the top menu, select the + Create button, and search for Function App. Select the Function App app. Then, select Create.
4. Configure the Function App with the following settings, then select Next: Hosting >:

| **Field** | **Value** |
| --- | --- |
| App Name | *your choice, must be globally unique* |
| Subscription | *select the appropriate subscription* |
| Resource Group | use existing, and *select Gv\_oilpumps* |
| Publish | *select Code* |
| Runtime Stack | *select .Net* |
| Version | *select 3.1* |
| Location | *select the location nearest to you* |

1. Configure the Hosting options as follows, then select Review + create:

| **Field** | **Value** |
| --- | --- |
| Storage Account | *retain the default value of create new* |
| Operating System | Windows |
| Plan Type | Consumption (Serverless) |

1. On the Review blade, select Create, then wait until the Function App is created before continuing.

**Task 18: Create a notification table in Azure Storage**

1. In the Azure portal, select Resource groups from the left-hand menu, then select the Gv\_oilpumps link from the listing.
2. Select the link for the storage account that was created with the Function App in Task 1. The name will start with "storageaccount".
3. From the Storage Account left-hand menu, select Tables from the Data storage section, then select the + Table button, and create a new table named DeviceNotifications.
4. Keep the Storage Account open in your browser for the next task.

**Task 19: Create the local settings file for the Azure Functions project**

1. Using Using Visual Studio Code, open the code folder
2. In this folder, create a new file named local.settings.json and populate it with the values of coonection string of azure web storage, evnethub primary key and prediction model endpoint.

**Task 20: Run the Function App locally**

1. Select Ctrl+F5 to run the Azure Function code.
2. After some time, you should see log statements indicating that a message has been queued.

**Task 21: Prepare the Azure Function App with settings**

* + - 1. You can now exit the locally running functions by selecting the Terminal window by pressing the Ctrl+c keys.

**Task 22: Prepare the Azure Function App with settings**

1. In Task 6, we created a local settings file to hold environment variables that are used in our function code. We need to mirror these values in the Azure Function App as well. In the Azure portal, access the Gv\_oilpumps resource group, and open the pumpfunctions Function Application.
2. Select the Configuration option in the left-hand menu.
3. In the Application Settings section, we will add the following application settings to mimic those that are in our local.settings.json file. Add a new setting by selecting the New application setting button.

| **Setting** | **Value** |
| --- | --- |
| fabrikam-oil\_RootManageSharedAccessKey\_EVENTHUB | *event hub shared access key value from the local.settings.json file* |
| PredictionModelEndpoint | *prediction model endpoint value from local.settings.json file* |

1. Once complete, select the Save button from the top menu to commit the changes to the application configuration.

**Task 23: Deploy the Function App into Azure**

Now that we have been able to successfully run our Functions locally, we are ready to deploy them to the cloud. The first step to deployment is to ensure that you are logged in to your Azure Account. To log into your Azure account, select the following shortcut to display the command palette: Ctrl+Shift+p.

* + - 1. In the textbox of the command palette, type in Azure:Sign In, and select enter (or select the command from the list). This will open a Microsoft Authentication webpage in your default browser. Logging into this window will authenticate Visual Studio Code with your ID.
      2. Once authenticated, we are ready to deploy - once again select Ctrl+Shift+p to open the command palette. Type Azure Functions: Deploy and select the Azure Functions: Deploy to Function App command from the list.

1. The first step of this command is to identify where we are deploying the function to. In our case, we have already created a Function App to house our function called pumpfunctions. Select this value from the list of available choices.
2. You may be prompted if you want to deploy to pumpfunctions, select the Deploy button in this dialog.
3. After some time, a notification window will display indicating the deployment has completed.
4. A notification is shown indicating the deployment to pumpfunctions has completed.
5. Returning to the Azure Portal, in the Gv\_oilpumps resource group, open the pumpfunctions function app and observe that our function that we created in Visual Studio Code has been deployed.