|  |  |
| --- | --- |
|  | |
|  |  |
|  | https://www.bluemetal.com/~/media/Images/CSS/bluemetal-logo |

d2c2d

Lab Workbook Five

Device to Cloud to Device - a workshop for learning about Windows 10 Core IoT device development, Azure IoT Hub, Stream Analytics and automating Azure using PowerShell

# Workshop Overview

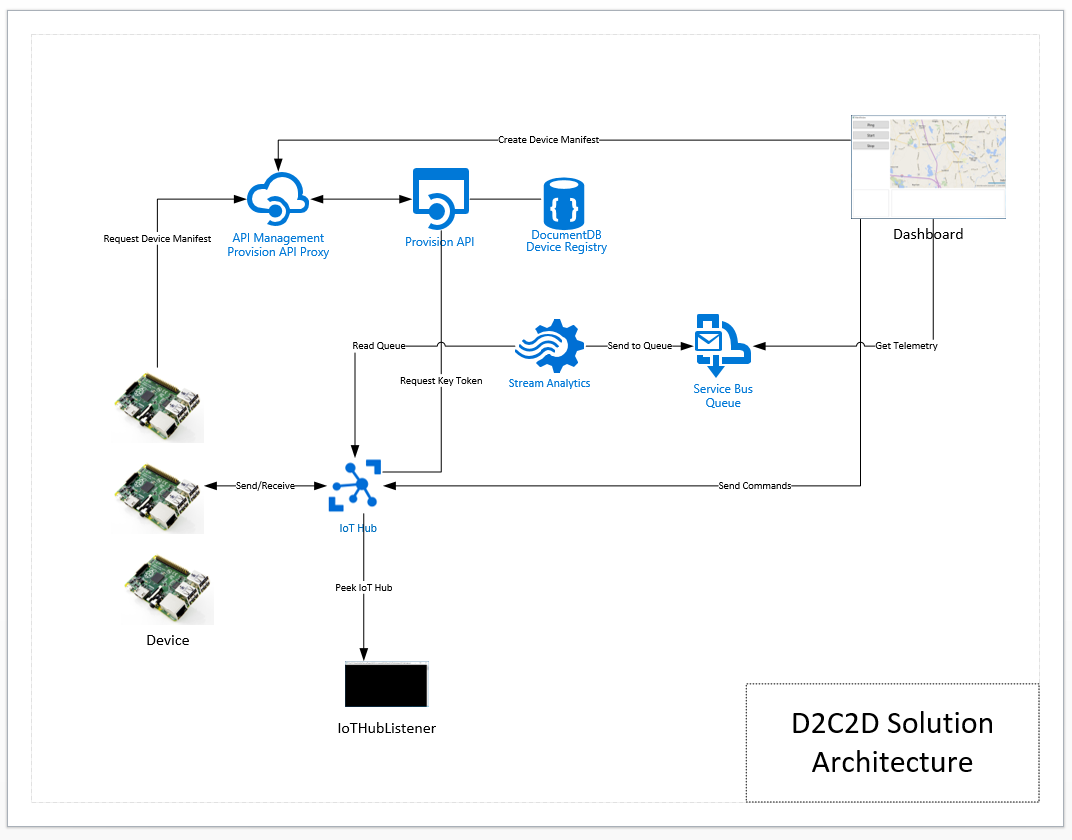
This training program provides foundational knowledge in how to architect and implement an IoT solution using Windows 10 Core IoT hardware devices and Azure IoT Hub and Stream Analytics. Both Device-to-Cloud and Cloud-to-Device communication patterns are discussed, designed, and implemented using best practices.

At the conclusion of this workshop you will have provisioned, using PowerShell, an Azure environment that contains IoT Hub, Stream Analytics Jobs that identify telemetry events and alarm states, and a Service Bus Namespace and set of message queues for backend integration.

You will also have developed a Windows 10 Core IoT application (“device”) that sends telemetry and receives incoming commands from the cloud as well as a real-time dashboard that can communicate bi-directionally with the device (e.g., displaying telemetry readings and sending commands to the remote device).

Device Provisioning and IoT Hub monitoring and techniques for dynamic business rules are also covered.

.



Solution Architecture

The solution you will build and deploy consists of the following components:

* **Device** – a Windows 10 IoT Core IoT solution that dynamically connects to IoT hub and sends heartbeat and climate telemetry as well as responds to command from a dashboard. The device application can run on your local machine or be deployed to a Windows 10 Core IoT device, such as a Raspberry Pi.
* **Dashboard** – a Windows 10 WPF application that lists registered devices, maps location using Bing Maps, and displays incoming device telemetry and alarms.
* **Provision API** – a ReST API that provides endpoints for device and device manifest lookup via a unique serial number. The Dashboard application registers devices, and the Device application uses the API to retrieve its manifest.
* **IoT Hub Listener –** a debugging utility that provides visibility to messages arriving from the device.

and the following Azure Services

* **Service Bus** – two queues are defined: one is a target for all incoming messages, and the other receives messages representing an alarm state, for instance, an out-of-range value
* **IoT Hub –** IoT Hub provides device registration, incoming telemetry at scale, and cloud-to-device message services
* **DocumentDb** – DocumentDb is a NoSQL database service that is used for managing Device Manifests, i.e., a Device Registry
* **Stream Analytics –** the solution leverages two Stream Analytics jobs, one that handles all incoming messages and another that identifies alarm states and routes those messages to a second queue.

# Lab Five Overview

In this lab you will deploy a new Stream Analytics Job that identifies alarm states in the incoming telemetry based on business rules that have been staged in Blob storage. These messages are routed to the alarms Service Bus Queue and displayed in the Dashboard.

# Lab

|  |  |
| --- | --- |
| **Step** | **Details** |
| **1** | **Provision a Stream Analytics Job that uses Reference Data**  In order to identify alarm states, you first need to know what the rules are for alarm states, i.e. what are the upper and lower bounds for temperature and humidity. These upper and lower bounds can be placed in a JSON file and staged in Blob storage.  The rules file which you will use is located in the automation/deploy/rules folder and named *devicerules.json*. It contains a field for message type and the upper and lower bounds for both temperature and humidity. The rules file can be joined with any message that is of type 2, i.e., Climate messages, enableing you to compare compare Temperature and Humidity fields to the upper and lower bounds described by the rule.  [  {  "MessageType": 2,  "TempUpperBound": 100.0,  "TempLowerBound": 32.0,  "HumidityUpperBound": 75.0,  "HumidityLowerBound": 10.0  }  ]  Stream Analytics has a feature where reference data can be applied as an additional input on the job definition and then joined with incoming messages to compare and/or replace values. The rules file will be defined as an input source of type reference data. You will then perform the join and comparison operation in a query.  SELECT  Stream.Id,  Stream.DeviceId,  Stream.MessageType,  Stream.Longitude,  Stream.Latitude,  Stream.[Timestamp],  Stream.Temperature,  Stream.Humidity  INTO  alarmqueue  FROM  iothub as Stream  **JOIN refdata Ref on Stream.MessageType = Ref.MessageType**  **WHERE ((Stream.Temperature > Ref.TempUpperBound) or**  **(Stream.Temperature < Ref.TempLowerBound) or**  **(Stream.Humidity > Ref.HumidityUpperBound) or**  **(Stream.Humidity < Ref.HumidityLowerBound)**  To provision this Stream Analytics job, run the 05-Provision-SAJob-2.ps1 script and provide the parameters as prompted:   * .\05-Provision- SAJob-2.ps1   **Subscription**: [the name of your subscription]  **ResourceGroup**: [the name of your resource group, d2c2d for example]  **Azure Location**: [East US for example]  **Prefix**: [a unique prefix to be used in the naming of service components]  **Suffix**: [dev | tst | stg | prd]  Validate that the script provisions the Stream Analytics job – named ‘d2c2d-alarms-queue’ - by navigating to the Azure Portal Resource Groups screen and clicking on that resource. Note that the job has two inputs, one for IoT Hub and the other for the reference data |
| **2** | **Update the Dashboard to display Alarm Messages**  You’ll now want to read incoming messages from the *alarms* queue and display them on the screen, using the same pattern as you did for the telemetry messages.   * Add the code below to the MainWindow\_OnLoaded() routine right after the messageTask code block:   var alarmTask = Task.Factory.StartNew(() =>  {  while (true)  {  var alarm = \_alarmClient.Receive();  var messageBody = string.Empty;  if (alarm == null) continue;  try  {  messageBody = alarm.GetBody<string>();  var obj = JsonConvert.DeserializeObject<MessageBase>(messageBody);  switch (obj.MessageType)  {  case MessageTypeEnum.NotSet:  throw new Exception("Message Type Not Set");  case MessageTypeEnum.Ping:  break;  case MessageTypeEnum.Climate:  var climate = JsonConvert.DeserializeObject<Climate>(  messageBody);  Application.Current.Dispatcher.Invoke(  DispatcherPriority.Background,  new ThreadStart(delegate  {  var currAlarm = AlarmFeed.Text;  AlarmFeed.Text = string.Empty;  AlarmFeed.Text += $"!!! ALARM !!!\r\n";  AlarmFeed.Text += $"Timestamp: {climate.Timestamp.ToLongDateString()} {climate.Timestamp.ToLongTimeString()}\r\n";  AlarmFeed.Text += $"Temperature: {climate.Temperature}\r\n";  AlarmFeed.Text += $"Humidity: {climate.Humidity}\r\n\r\n";  AlarmFeed.Text += $"{currAlarm}\r\n\r\n";  }));  break;  case MessageTypeEnum.Command:  // noop  break;  default:  throw new ArgumentOutOfRangeException();  }  alarm.Complete();  }  catch (Exception err)  {  Application.Current.Dispatcher.Invoke(  DispatcherPriority.Background, new ThreadStart(delegate  {  var currAlarm = AlarmFeed.Text;  AlarmFeed.Text = string.Empty;  AlarmFeed.Text += $"{err.Message}\r\n";  AlarmFeed.Text += $"{messageBody}\r\n\r\n";  AlarmFeed.Text += $"{currAlarm}\r\n\r\n";  alarm.Abandon();  }));  }  }  });   * Test your solution   + Start the Device Solution   + Start the Dashboard Solution   + Click the Start Button |
| **3** | **Congratulations! You have competed Lab 5**  Let’s review:   * You deployed a new Stream Analytics job that uses a reference data file containing business rules for temperature and humidity * You updated the Dashboard with a background thread that listens for alarm messages and displays them on the screen |