BLUEMETAL
An Insight company

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

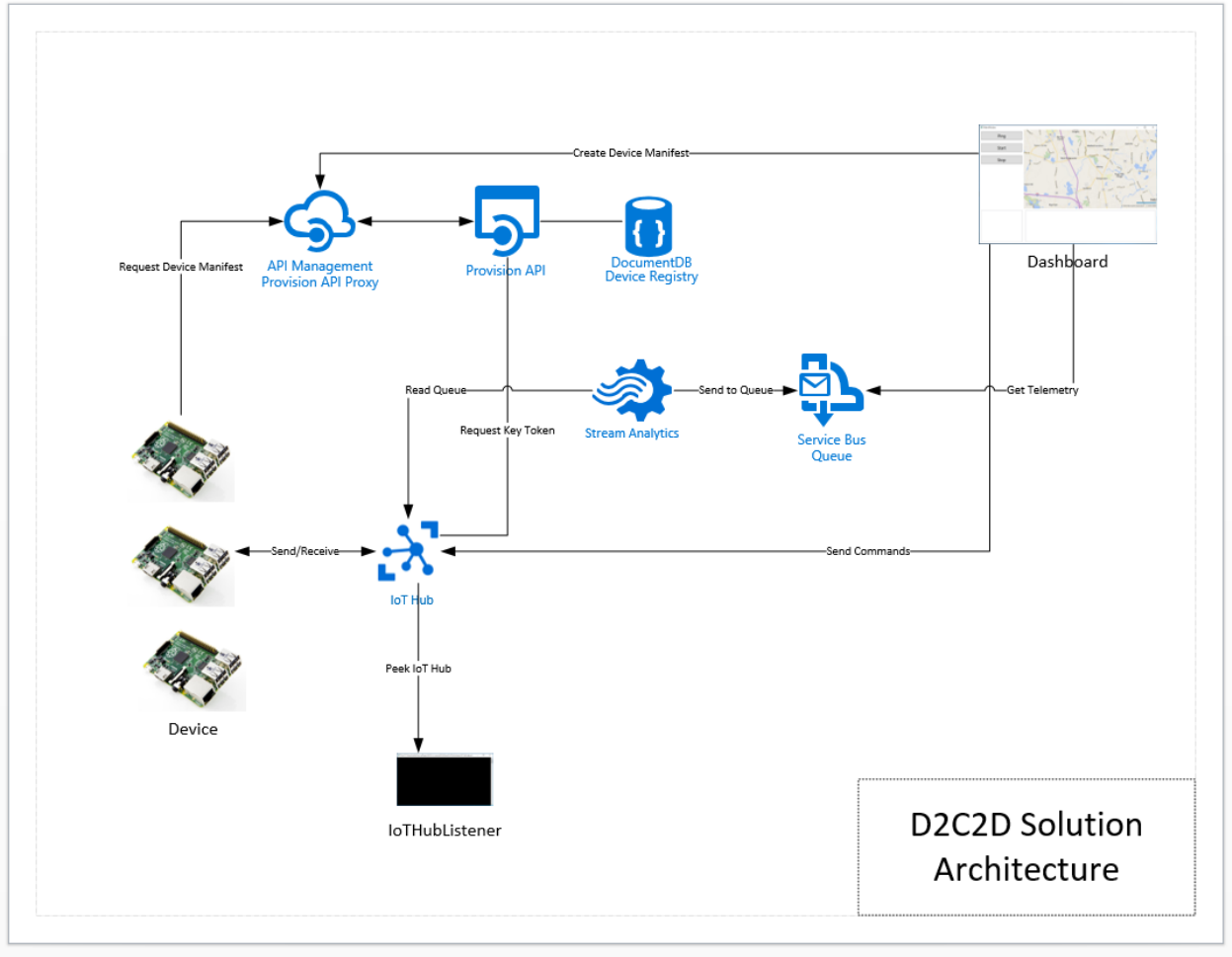
Workshop Overview

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 an Azure environment using PowerShell 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 develop a Windows 10 Core IoT application that sends telemetry and receives incoming commands as well as develop a real-time dashboard that displays incoming telemetry and has the ability to send commands to the remote device. Device Provisioning, IoT Hub monitoring and techniques for dynamic business rules will be covered.

.



Solution Architecture

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

- **Device** – a Windows 10 IoT Core IoT solution that dynamically connects to IoT hub providing heartbeat and climate telemetry and processes several incoming commands. The device application will run on yoru local system or can be deployed to a Windows 10 Core IoT device
- **Dashboard** – a Windows 10 WPF application that displays registered devices, map location using Bing Maps, incoming device telemetry and alarms
- **Provision API** – ReST API the provides end points for device registration with IoT Hub and DocumentDb and device manifest lookup via unique serial number. The Dashboard application registered devise 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

- **API Management** – provides proxy, policy injection and developer registration services for ReST APIs

- **Service Bus Namespace** – two queues are defined, one that is a target for all incoming messages, the other will have receive messages that contain data that is an alarm state, 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 manage Device Manifests, i.e. a Device Registry
- **Stream Analytics Job –** two solution uses two Stream Analytics jobs, one that handles all incoming messages routing them to one queue and the other identifies alarm states and routs those messages to another queue

Lab Five Overview

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

Lab

Step	*Details*

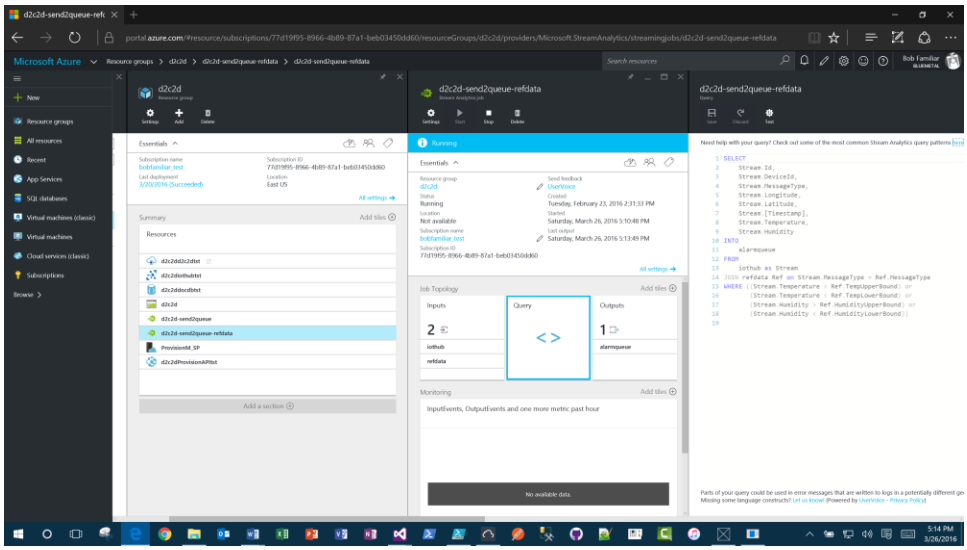
1	**Provision a Stream Analytics Job that uses Reference Data**
	
	In order to identify alarm states, we first need to know what the rules are for alarm states, i.e. what are the upper and lower bounds for temperature and humidity. These upper and lower bounds can be placed in a JSON file and staged the Blob storage.
	
	The rules file which we will use is located in the automation/deploy/rules folder. The file is called *devicerules.json*. It contains a field for message type and then the upper and lower bounds for both temperature and humidity. Using this rules file we can join this record with any message that is of type 2, i.e. Climate messages, and then compare Temperature and Humidity fields to the upper and lower bounds.
	
	`[`
	`  {`
	`  "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. Our rules file will be defined as an input source of type reference data. We will then perform the join and comparison operation in our 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)
```

There is a PowerShell script that will provision this job. Run PowerShell Console as Administrator 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]

This script will deploy a Stream Analytics job called 'd2c2d-alarms-queue'. Validate that the script provisions the Stream Analytics job by navigating to the Azure Portal Resource Groups screen and clicking on that resource. Note that the job has 2 inputs, one for IoT Hub and the other for the reference data



2 Update the Dashboard to display Alarm Messages

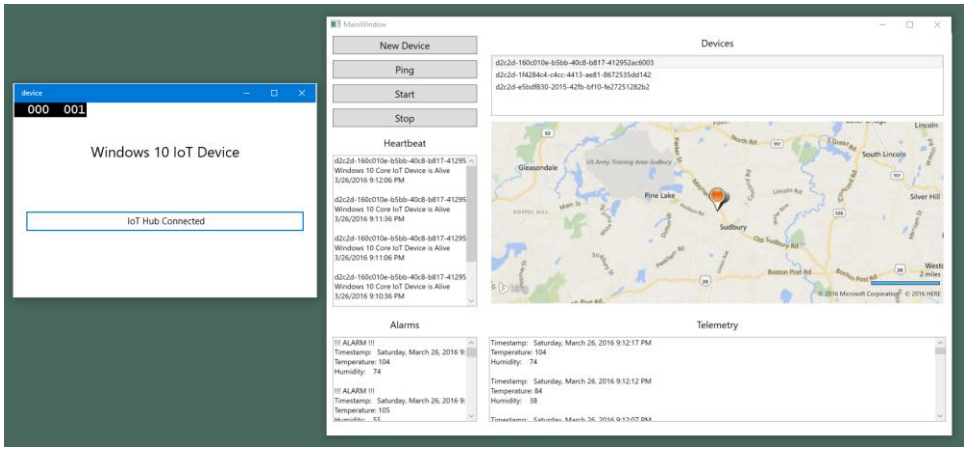
We have used the pattern already. We want to read incoming messages from a queue and display them on the screen n this case we need a background thread that will listen on the alarms queue and display those messages in the Alarm window on the Dashboard.

- Add the code below to the MainWindow_OnLoaded() routine right after the messageTask code block using the d2c2d-DashboardAlarmTask snippet

```
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 that contains 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