d2c2d

Lab Workbook Four

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

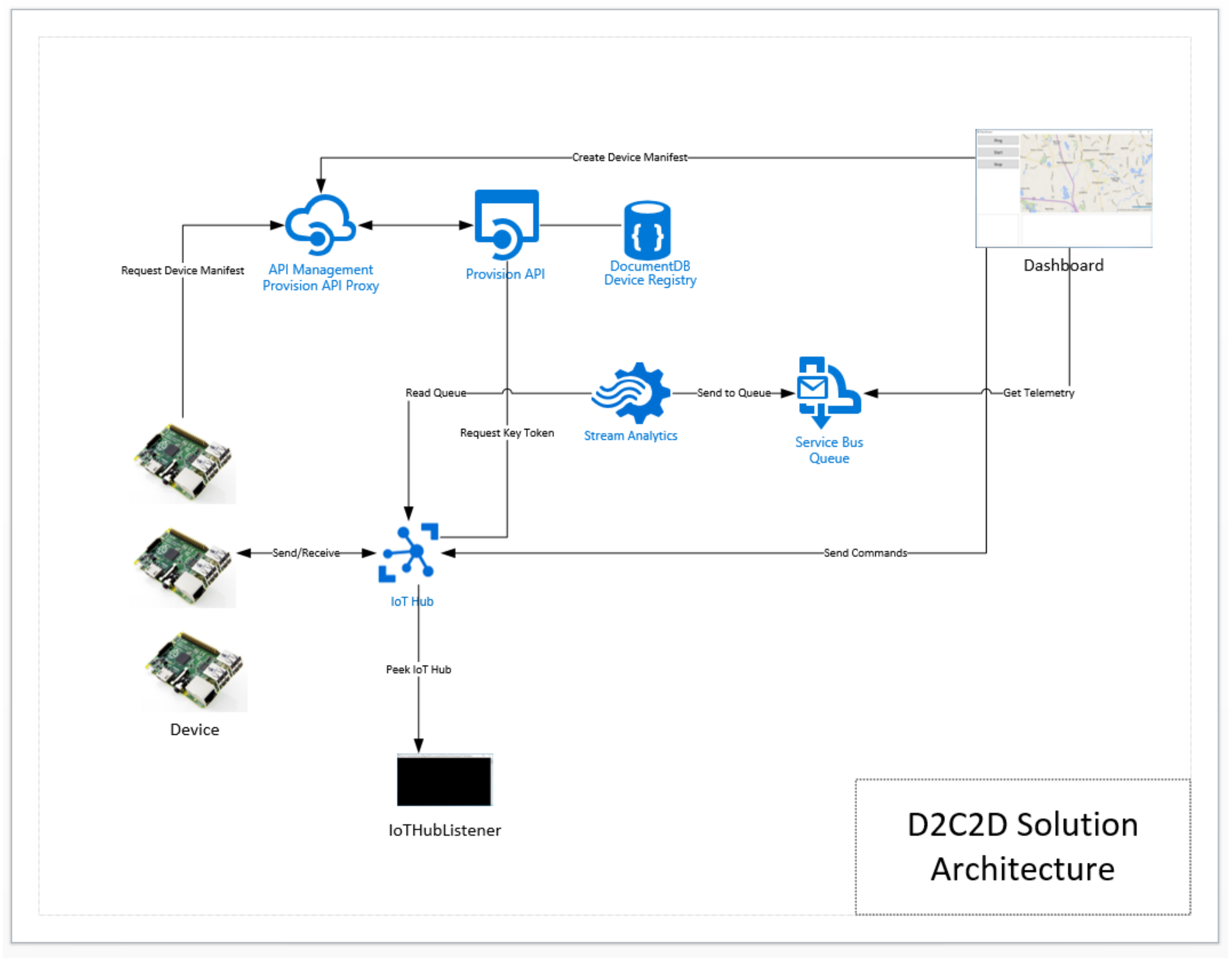
Workshop Overview

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 Four Overview

In Lab 3, we setup the solution to support sending messages from the device to the cloud. In this lab you will add command and control to the solution. Command and control is all about having the ability to send messages from the cloud to the device. In IoT solutions, The messages that are sent from the cloud can be commands such as upgrade firmware, start or stop taking measurements or possible text to display on an LCD screen if the device has one.

In all these cases there is the command and then there are the parameters of the command, such as where to download the firmware file, which telemetry to start or stop. How to package command parameters will be covered in this lab.

Lab

Step *Details*

1 **Update the Device solution to support receiving commands**

In order for the device to receive incoming messages, it must set up a background thread that listens on the channel provided by IoT Hub. The DeviceClient provides a ReceiveAsync() method for this purpose.

- Add the StartListenTask() method to the MainPage class

```
private static void StartListenTask(TextBox status)
{
    _listenTask = Task.Factory.StartNew(async () =>
    {
        while (true)
        {
            var message = await _deviceClient.ReceiveAsync();

            if (message == null)
                continue;

            var json = Encoding.ASCII.GetString(message.GetBytes());

            var command = JsonConvert.DeserializeObject<Command>(json);

            switch (command.CommandType)
            {
                case CommandTypeEnum.Ping:
```

```
                            var ping = new Ping
                            {
                                Ack = AckMessage,
                                Longitude = _deviceManifest.longitude,
                                Latitude = _deviceManifest.latitude,
                                DeviceId = _deviceManifest.serialnumber
                            };

                            json = JsonConvert.SerializeObject(ping);

                            var pingMessage = new Message(Encoding.ASCII.GetBytes(json));

                            try
                            {
                                await _deviceClient.SendEventAsync(pingMessage);
                            }
                            catch (Exception err)
                            {
                                var errMessage = err.Message;
                                status.Text = errMessage;
                            }
                            break;

                        case CommandTypeEnum.Start:
                            // the command is to start telemetry
                            // unpack the parameters that define the upper and lower bounds
                            var settings = JsonConvert.DeserializeObject<ClimateSettings>(
                                 command.CommandParameters);
                            _sendingTelemetry = true;
                            StartTelemetry(settings, status);
                            break;

                        case CommandTypeEnum.Stop:

                            _sendingTelemetry = false;
                            break;

                        case CommandTypeEnum.UpdateFirmeware:
                            // imagine
                            break;

                        default:
                            throw new ArgumentOutOfRangeException();
                    }

                    await _deviceClient.CompleteAsync(message);
                }
            });
        }
```

One of the commands that is sent to the device from the dashboard is to start sending telemetry. Sending telemetry, like sending Ping messages, is a background thread where readings are taken and then sent as a message to IoT Hub. The loop within the background thread is controlled by the Boolean _sendingTelemetry. Stoping telemetry is simple done by setting this value to false.

When sending telemetry, the Device creates a Climate message and fills out the Temperature and Humidity measurements using the ClimateSettings that were passed in from the Dashboard.

- Add the StartTelemetry() method to the MainPage class

```
        private static void StartTelemetry(ClimateSettings settings, TextBox status)
        {
            _telemetryTask = Task.Factory.StartNew(async () =>
            {
                while (_sendingTelemetry)
                {
                    var random = new Random();

                    var climate = new Climate
                    {
                        Longitude = _deviceManifest.longitude,
                        Latitude = _deviceManifest.latitude,
                        DeviceId = _deviceManifest.serialnumber,
                        Temperature = random.Next((int) settings.MinTemperature,
                           (int) settings.MaxTemperature),
                        Humidity = random.Next((int) settings.MinHumidity,
                           (int) settings.MaxHumiditiy)
                    };

                    var json = JsonConvert.SerializeObject(climate);

                    var message = new Message(Encoding.ASCII.GetBytes(json));

                    try
                    {
                        await _deviceClient.SendEventAsync(message);
                    }
                    catch (Exception err)
                    {
                        var errMessage = err.Message;
                        status.Text = errMessage;
                    }

                    await Task.Delay(5000);
                }
            });
        }
```

2 Update the Dashboard to Send Commands to the Device

The Device uses an object called DeviceClient in order to connect and communicate with IoT Hub. In order to perform service side operations with IoT Hub, the IoT Hub SDK provides the ServiceClient.

The ServiceClient is used by the Dashboard to send messages to specific devices. When you send a message to a device using IoT Hub, it actually sits in a queue and is either a) retrieved by the Device if it is in listen mode, or b) expires.

For purposes of the lab, the Dashboard will send both Start Telemetry and Stop Telemetry messages.

- Add the code that handles Climate Messages inside the MessageTask background thread. Add the following swtich label:

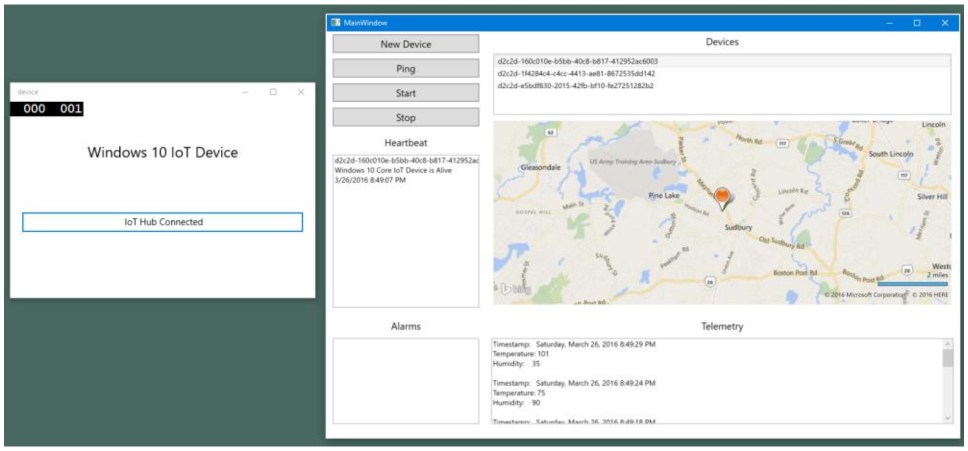
```
case MessageTypeEnum.Climate:
    var climate = JsonConvert.DeserializeObject<Climate>(
       messageBody);
    Application.Current.Dispatcher.Invoke(
          DispatcherPriority.Background,
        new ThreadStart(delegate
        {
            var currTelemetry = TelemetryFeed.Text;
```

```
                                TelemetryFeed.Text = string.Empty;
                                TelemetryFeed.Text += $"Timestamp:
                                   {climate.Timestamp.ToLongDateString()}
                                   {climate.Timestamp.ToLongTimeString()}\r\n";
                                TelemetryFeed.Text += $"Temperature:
                                   {climate.Temperature}\r\n";
                                TelemetryFeed.Text += $"Humidity:
                                    {climate.Humidity}\r\n\r\n";
                                TelemetryFeed.Text += $"{currTelemetry}\r\n\r\n";
                            }));
                        break;
```

- Add the StartButton_Click() and StopButton_Click() event handlers

```
private void StartButton_Click(object sender, RoutedEventArgs e)
{
    var climateSettings = new ClimateSettings
    {
        MinHumidity = 0,
        MaxHumiditiy = 100,
        MinTemperature = 75,
        MaxTemperature = 110
    };

    var command = new Command
    {
        CommandType = CommandTypeEnum.Start,
        CommandParameters = JsonConvert.SerializeObject(climateSettings),
        DeviceId = _currDevice.serialnumber
    };

    var json = JsonConvert.SerializeObject(command);
    var message = new Message(Encoding.ASCII.GetBytes(json));

    _serviceClient.SendAsync(_currDevice.serialnumber, message);
}

private void StopButton_Click(object sender, RoutedEventArgs e)
{
    var command = new Command
    {
        CommandType = CommandTypeEnum.Stop,
        DeviceId = _currDevice.serialnumber
    };
    var json = JsonConvert.SerializeObject(command);
    var message = new Message(Encoding.ASCII.GetBytes(json));
    _serviceClient.SendAsync(_currDevice.serialnumber, message);
}
```

- Test your implementation
    - Start the Device Solution
    - Start the Dashboard
    - Click the Start Button
    - You should see messages arriving in the Telemetry output window



```
d2c2d-160c010e-b5bb-40c8-b817-412952ac6003
d2c2d-1f4284c4-c4cc-4413-ae81-8672535dd142
d2c2d-e5bdf830-2015-42fb-bf10-fe27251282b2
```

```
d2c2d-160c010e-b5bb-40c8-b817-412952ac
Windows 10 Core IoT Device is Alive
3/26/2016 8:49:07 PM
```

```
Timestamp:   Saturday, March 26, 2016 8:49:29 PM
Temperature: 101
Humidity:    35

Timestamp:   Saturday, March 26, 2016 8:49:24 PM
Temperature: 75
Humidity:    90
```

N **Congratulations! You have competed Lab 4**

Let’s review:

- You added the background threads in the Device solution to listen for commands and to start and stop sending telemetry
- You updated the Dashboard to send commands to the Device