d2c2d

Lab Workbook Four

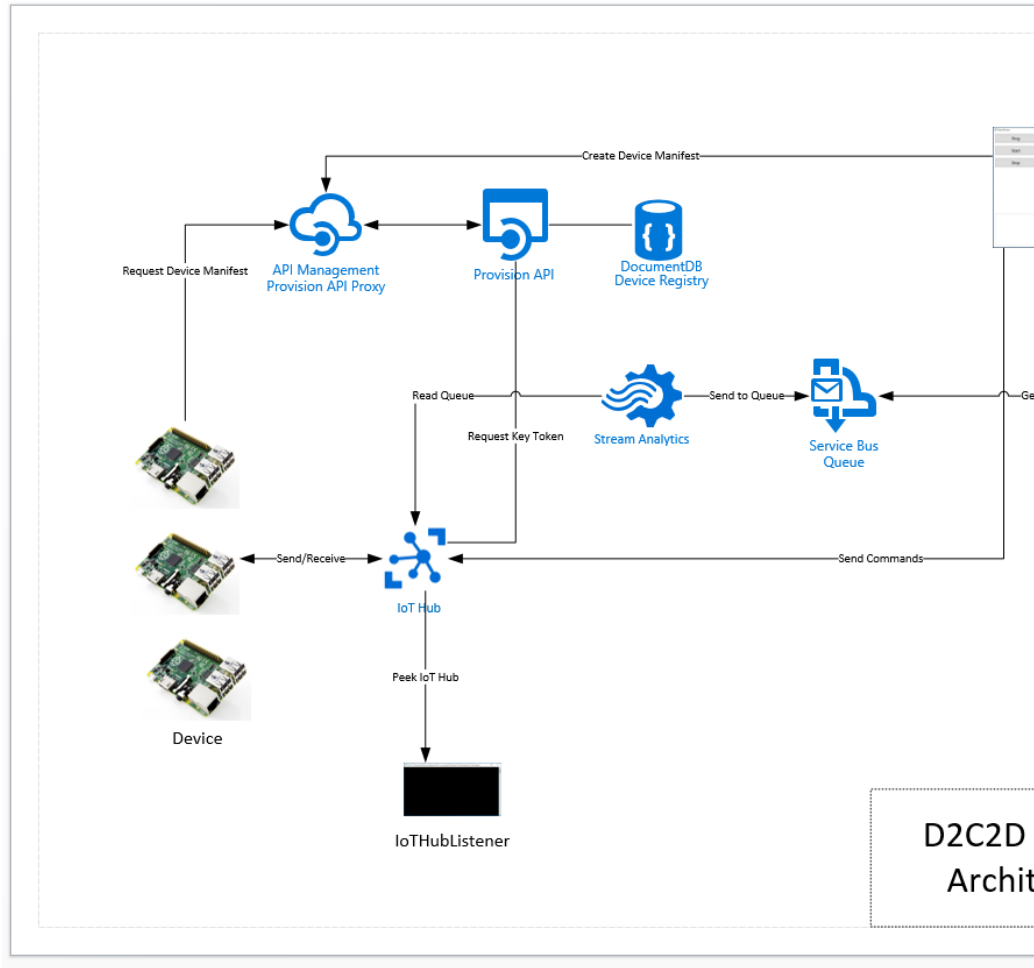
Device to Cloud to Device - a workshop for learning about Windows 10 Cor
development, Azure IoT Hub, Stream Analytics and automating Azure using

Workshop Overview

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

At the conclusion of this workshop you will have provisioned, using PowerShell, an Azu
that contains IoT Hub, Stream Analytics Jobs that identify telemetry events and alarm s
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 t
receives incoming commands from the cloud as well as a real-time dashboard that can
directionally with the device (e.g., displaying telemetry readings and sending command
device).



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 h heartbeat and climate telemetry as well as responds to command from a dashb application can run on your local machine or be deployed to a Windows 10 Cor as a Raspberry Pi.
- **Dashboard** – a Windows 10 WPF application that lists registered devices, maps Bing Maps, and displays incoming device telemetry and alarms.
- **Provision API** – a ReST API that provides endpoints for device and device manif

- **IoT Hub –** IoT Hub provides device registration, incoming telemetry at scale, an message services
- **DocumentDb** – DocumentDb is a NoSQL database service that is used for mana Manifests, i.e., a Device Registry
- **Stream Analytics –** the solution leverages two Stream Analytics jobs, one that l incoming messages and another that identifies alarm states and routes those m second queue.

Lab Four Overview

In Lab Three, you set up the solution to support sending messages from the device to t lab you will add command and control to the solution. Command and control is all abou messages from the cloud to the device. In IoT solutions, the messages that are sent fro be commands such as upgrade firmware, start or stop taking measurements or even te an LCD screen if the device has one.

In the generalized case there is a command and parameters of the command, such as w download the firmware file or which telemetry to start or stop. How to package comma 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 backgro listens on the channel provided by IoT Hub. The DeviceClient provides a Recei 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;`

```
                    DeviceId = _deviceManifest.serialnumber
                };

                json = JsonConvert.SerializeObject(ping);

                var pingMessage = new Message(Encoding.ASCII.GetByt

                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
                var settings = JsonConvert.DeserializeObject<Climat
                     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);
    }
});
}
```

- Add a call to this new method as the last line of the MainPage_OnLoa
  (directly below the call to `StartPingTask(Status);`

```
StartListenTask(Status);
```

One of the commands that is sent to the device from the dashboard is to start
telemetry. Sending telemetry, like sending Ping messages, is a background thr

- Add the StartTelemetry() method to the MainPage class :

```
private static void StartTelemetry(ClimateSettings settings, TextBox st
{
    _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.MaxHumidity)
            };

            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 to connect and communicate w
order to perform service-side operations with IoT Hub, the IoT Hub SDK provi
ServiceClient.

The ServiceClient is used by the Dashboard to send messages to specific devic
send a message to a device using IoT Hub, it actually sits in a queue and is eith
the Device if it is in listen mode, or it expires.

```
            MinHumidity = 0,
            MaxHumidity = 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 wir