Lab 3

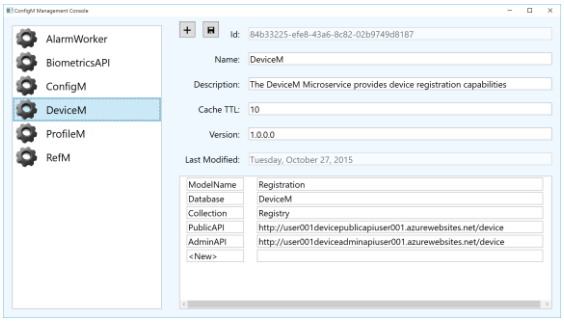
SUBJECT	TEACHER	DATE
IoT Hands-On Lab	Bob Familiar	November 19, 2015

OVERVIEW

In Lab 3, you will configure the microservice environment using the ConfigM Management Console. ConfigM is a microservice that provides dynamic discovery services at runtime. Next you will develop a console application that will simulate hundreds of biomedical devices sending messages to IoT Hub.

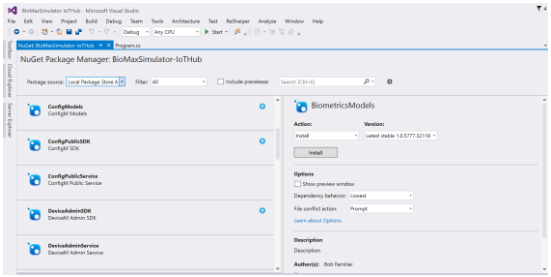
LAB 3		

STEP 1	• Using Visual Studio, open the ConfigMConsole solution file located in the folder HandsOnLabs\Microservices\Config\Consoles\ConfigMConsole
	• Open the App.Config file and update the 'ConfigM' application setting value to be the URL to your ConfigM Public API service. The URL should include '/config' on the end
	• Note: the URL for the service can be found in the Azure Portal
		`<appSettings>`
		`  <add key="ConfigM" value="[url]" />`
		`</appSettings>`

STEP 2	• Run the solution and update the DeviceM, ProfileM and Biometrics API manifests with URLs to their respective Public API and Admin API. • Include '/profile', '/device' and '/biometrics' at the end of the URLs • Note: the URLs for the services can be found in the Azure Portal	

STEP 3	• Using Visual Studio, create a new project called 'BiomaxSimulator' of type Console Application and save the solution in the HandsOnLabs\Code folder. • Add the DeviceMessage.cs and ConsoleSpinner.cs files to your project. They can be found in HandsOnLabs\Code folder

STEP 4	• Using the NuGet Package Manager Console, execute the following commands to load the Azure IoT Hub libraries:
	`> Install-Package Microsoft.Azure.Devices -Pre`
	`> Install-Package Microsoft.Azure.Devices.Client -Pre`

STEP 5	• Using the NuGet Package Manager, add references to the following NuGet packages from the local repository: ○ ConfigM Public SDK ○ ProfileM Public SDK ○ DeviceM Admin SDK 

STEP 6	• Add the ConfigM URI, IoT Hub URI and Connection String to the App.Config file
	`<appSettings>`
	`<add key="IoTHubUri" value="IoTHub URI" />`
	`<add key="IoTHubConnStr" value="IoT Hub Connection String" />`
	`<add key="ConfigM" value="ConfigM Public API" />`
	`</appSettings>`

STEP 7	• Add the following using statements at the top of the Program.cs file:
	`using System;`
	`using System.Collections.Generic;`
	`using System.Configuration;`
	`using System.Text;`
	`using System.Threading;`
	`using Microsoft.Azure.Devices.Client;`
	`using Microsoft.Azure.Devices;`
	`using Microsoft.Azure.Devices.Common.Exceptions;`
	`using LooksFamiliar.Microservices.Common.Wire;`
	`using LooksFamiliar.Microservices.Config.Models;`
	`using LooksFamiliar.Microservices.Config.Public.SDK;`
	`using LooksFamiliar.Microservices.Device.Admin.SDK;`
	`using LooksFamiliar.Microservices.Device.Models;`
	`using LooksFamiliar.Microservices.Profile.Models;`
	`using LooksFamiliar.Microservices.Profile.Public.SDK;`

STEP 8	• Add the following declarations at the top of the Program class definition:
	`class Program`
	`{`
	`static readonly DeviceClient[] DeviceClients = new DeviceClient[300];`
	`static RegistryManager _registryManager;`
	`static ConfigM _configM;`
	`static DeviceM _registryM;`
	`static ProfileM _profilesM;`
	`static Registrations _devices;`
	`static List<UserProfile> _profiles;`

STEP 9	• Add the following code at thte start of Main to provide a simple set of instructions to start the simulator.
	
	`static void Main(string[] args)`
	`{`
	`Console.WriteLine("************************************************************");`
	`Console.WriteLine("* B I O M A X S E N S O R E V E N T G E N E R A T O R *");`
	`Console.WriteLine("* *");`
	`Console.WriteLine("* I O T H U B E D I T I O N *");`
	`Console.WriteLine("************************************************************");`
	`Console.WriteLine();`
	`Console.WriteLine("Press Enter to start the generator.");`
	`Console.WriteLine("Press Ctrl-C to stop the generator.");`
	`Console.WriteLine();`
	`Console.ReadLine();`
	`Console.Write("Working....");`

STEP 10	• Add code to load microservice manifests dynamically using ConfigM and download the device registry and participant list
	• Stage the call SendDeviceToCloudMessagesAsync(), we will implement that next.
	
	`// initialize the ConfigM microservice client sdk`
	`_configM = new ConfigM {ApiUrl = ConfigurationManager.AppSettings["ConfigM"]};`
	
	`// lookup the manifests for the devie registry and user profile microservices`
	`var deviceManifest = _configM.GetByName("DeviceM");`
	`var profileManifest = _configM.GetByName("ProfileM");`
	
	`// initialize the DeviceM microservice client sdk`
	`_registryM = new DeviceM`
	`{ ApiUrl = deviceManifest.lineitems[LineitemsKey.AdminAPI] };`
	
	`// initialize the ProfileM microservice client sdk`
	`_profilesM = new ProfileM`
	`{ ApiUrl = profileManifest.lineitems[LineitemsKey.PublicAPI] };`
	
	`// get the device registry from the device microservice`
	`_devices = _registryM.GetAll();`
	
	`// get all the participants in the study`
	`_profiles = _profilesM.GetAllByType("Participant");`
	
	`// send simulated messages from the device collection`
	`SendDeviceToCloudMessagesAsync();`
	
	`Console.ReadLine();`

STEP 11	• Implement the SendDeviceCloudMessageAsync() method
	
	`private static async void SendDeviceToCloudMessagesAsync()`
	`{`
	`var random = new Random();`
	`var spin = new ConsoleSpiner();`
	
	`while (true)`
	`{`
	`spin.Turn();`
	
	`Thread.Sleep(1000);`
	`}`
	
	`}`

STEP 12	• Add a try/catch block after the call to spin.Turn(). This will encapsulate the code that gathers sensor readings and sends those readings to IoT Hub
	
	`try`
	`{`
	
	`}`
	`catch (Exception exception)`
	`{`
	`  Console.ForegroundColor = ConsoleColor.Red;`
	`  Console.WriteLine("{0} > Exception: {1}", DateTime.Now, exception.Message);`
	`  Console.ResetColor();`
	`}`

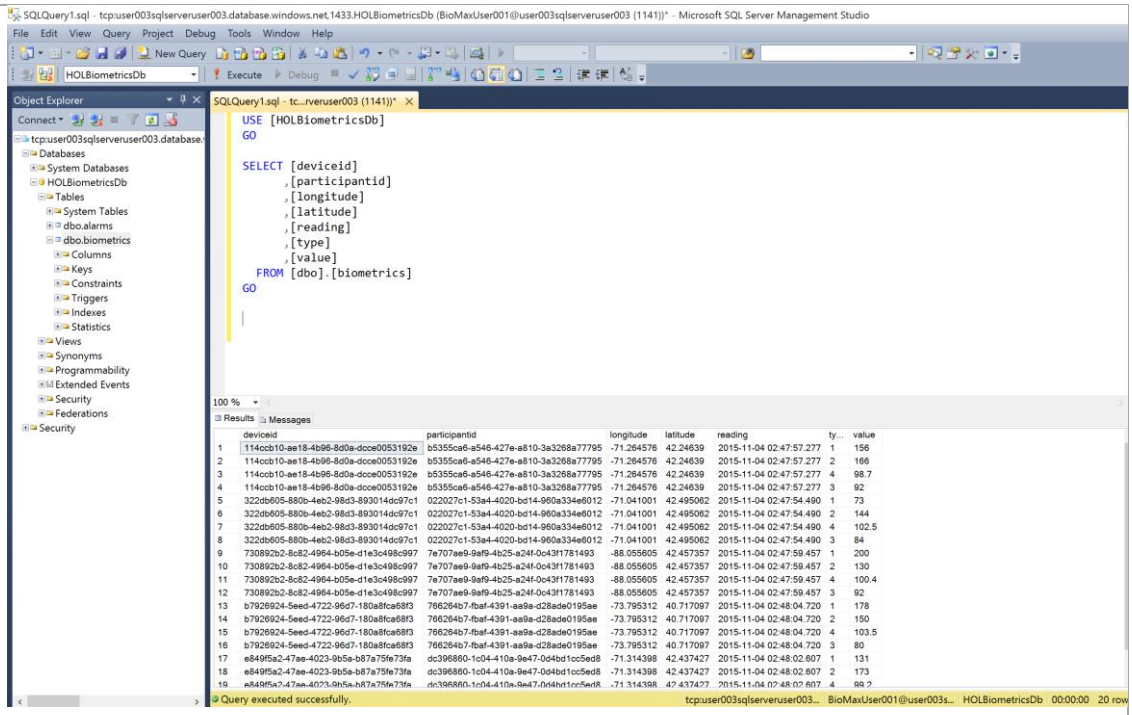
STEP 13	• Inside the try block, add the code that randomly selects a device from our list of 300, look up the participant that is associated with this device
	
	`// randomly select a device`
	`var index = random.Next(0, _devices.list.Count - 1);`
	`var deviceManifest = _devices.list[index];`
	
	`// lookup the participant associated with this device`
	`var participant = _profiles.Find(p => p.id == deviceManifest.participantid);`

STEP 14	• Create a unique device id from the combination of the model name and the device guid
	• Check to see if you have already connected to IoT Hub for this device and if not, request the asymmetric key from IoT Hub for this device and then use that along with the unique Id to create a connection.
	
	`// create a unique identifier for this device`
	`var deviceId = deviceManifest.model + "-" + deviceManifest.id;`
	
	`// create an IoT Hub client for this device if necessary`
	`if (DeviceClients[index] == null)`
	`{`
	`  Device device;`
	
	`  // initialize the IoT Hub registration manager`
	`  _registryManager = RegistryManager.CreateFromConnectionString(`
	`  ConfigurationManager.AppSettings["IoTHubConnStr"]);`
	
	`  // register or lookup the device registration from IoT Hub`
	`  try`
	`  {`
	`  device = await _registryManager.AddDeviceAsync(new Device(deviceId));`
	`  }`
	`  catch (DeviceAlreadyExistsException)`
	`  {`
	`  device = await _registryManager.GetDeviceAsync(deviceId);`
	`  }`
	
	`  // connect to the IoT Hub using the unique device`
	`  // registration settings (deviceid, devicekey)`
	`  DeviceClients[index] = DeviceClient.Create(`
	`  ConfigurationManager.AppSettings["IoTHubUri"],`
	`  new DeviceAuthenticationWithRegistrySymmetricKey(`
	`  deviceId,`
	`  device.Authentication.SymmetricKey.PrimaryKey));`
	`}`

STEP 15	• Create a DeviceMessage object and populate with the data from the device and participant registries.
	• Create simulated readings for glucose, heartrate, blood oxygen and temperature
	• Add those sensor readings to the sensor reading array
	• Set a reading date/time
	
	`// initialize a device message`
	`var deviceReading = new DeviceMessage();`
	
	`// begin to create the simulated device message`
	`deviceReading.deviceid = deviceManifest.id;`
	`deviceReading.participantid = participant.id;`
	`deviceReading.location.latitude = participant.location.latitude;`
	`deviceReading.location.longitude = participant.location.longitude;`
	
	`// generate simulated sensor reaings`
	`var glucose = new SensorReading`
	`{`
	`type = SensorType.Glucose,`
	`value = random.Next(70, 210)`
	`};`
	
	`var heartrate = new SensorReading`
	`{`
	`type = SensorType.Heartrate,`
	`value = random.Next(60, 180)`
	`};`
	
	`var temperature = new SensorReading`
	`{`
	`type = SensorType.Temperature,`
	`value = random.Next(98, 105) + (.1 * random.Next(0, 9))`
	`};`
	
	`var bloodoxygen = new SensorReading`
	`{`
	`type = SensorType.Bloodoxygen,`
	`value = random.Next(80, 100)`
	`};`
	
	`deviceReading.sensors.Add(glucose);`
	`deviceReading.sensors.Add(heartrate);`
	`deviceReading.sensors.Add(temperature);`
	`deviceReading.sensors.Add(bloodoxygen);`
	
	`deviceReading.reading = DateTime.Now;`

STEP 16	• Convert the Device Message to JSON and send asynchronously to IoT Hub
	
	`// serialize the message to JSON`
	`var json = ModelManager.ModelToJson<DeviceMessage>(deviceReading);`
	
	`// send the message to EventHub`
	`DeviceClients[index].SendEventAsync(`
	`new Microsoft.Azure.Devices.Client.Message(`
	`Encoding.ASCII.GetBytes(json))).Wait();`

STEP 17	• Compile and run to validate that you can connect to the microservices, IoT and that messages are being delivered to SQL Server and DocumentDb • You can use Visual Studio or SQL Server Management Studio to connect to your SQL Database and check to see if data is flowing in through IoT Hub and Stream Analytics



- Navigate to your DocumentDb instance and select the Biometrics database use the Document Explorer tool to investigate the contents of the Alarms collection