IoT Pipeline Guide

Contents

File and Folder Descriptions

1) Folder: Hack Day

   a) Folder: **EventHubDashBoard:** C# App that reads incoming messages from an event hub in real-time.

   b) Folder: **TISenorTagEventHub_WindowsForms-master**: App that records data from a TI Sensor Tag and transmits it to an Event Hub.

   c) Folder: **Test_Data_And_Sample_Outputs**: this contains sample input data to test stream queries against. Also sample output data for building schemas.

       i) **SampleOutputHumidity.json**: a sample of what the output data would look like after a stream query.

       ii) **StreamQL Repository.sql**: A repository for T-SQL and StreamQL. Only to be used as a last resort.

       iii) **TestDataInput, Bruce Wayne.json**: Test data from Bruce Wayne's TI Sensor Tag.

       iv) **TestDataInput, Clark Kent.json**: Test data from Clark Kent's TI Sensor Tag.

       v) **TestDataInput.json**: Test data aggregated from both Clark Kent and Bruce Wayne's TI Sensor Tag.

   d) **Internet of Things.pdf:** Intro slides on the "internet of things" concept.

   e) **IoT Pipeline Guide**: A guide that gentle guides users through the creation of an IoT pipeline using TI Sensor Tags

datasciencedojo
unleash the data scientist in you

TI Sensor Tag

The sensor tag is a motherboard with 6-10 different sensors on it, depending on the model variations.

- Contactless IR temperature sensor (Texas Instruments TMP006)
- Humidity Sensor (Sensirion SHT21)
- Gyroscope (Invensense IMU-3000)
- Accelerometer (Kionix KXTJ9)
- Magnetometer (Freescale MAG3110)
- Barometric pressure sensor (Epcos T5400)
- On-chip temperature sensor (Built into the CC2541)
- Battery/voltage sensor (Built into the CC2541)
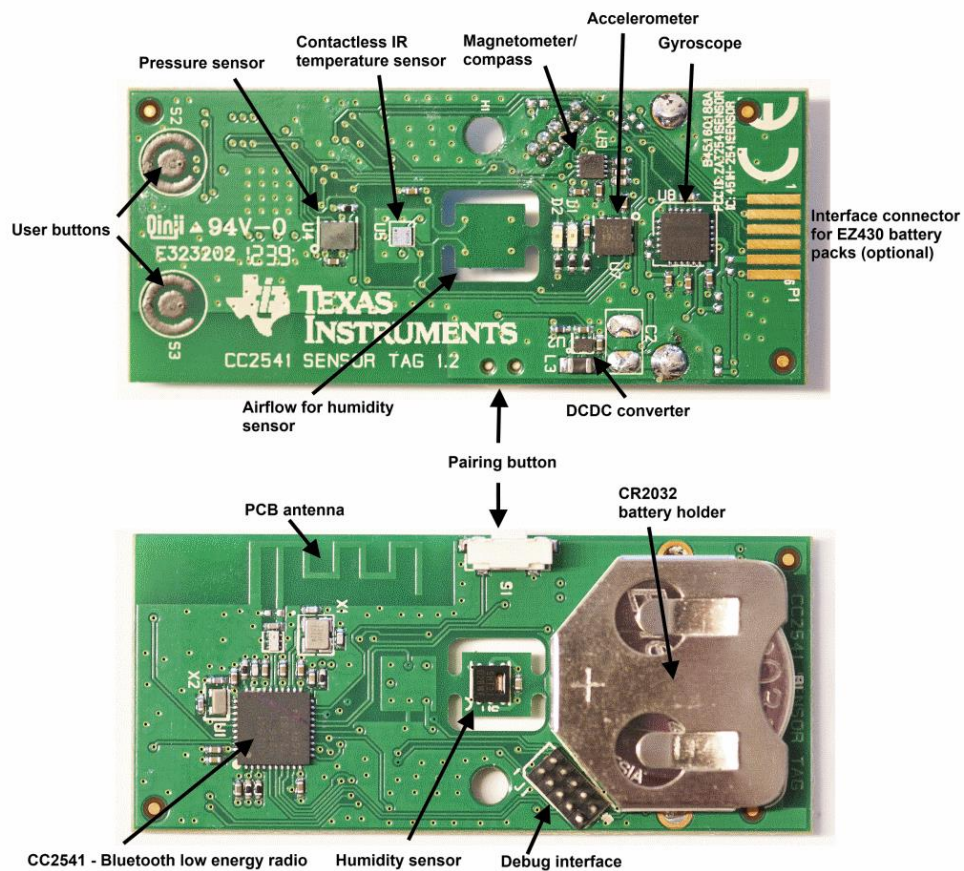
About the Sensors

Contactless IR Temperature Sensor

The IR temperature sensor can measure the temperature of objects in front of the SensorTag with typical +/- 1°C accuracy. The measurement distance depends on the size of the object to be measured. The datasheet recommends maximum distance of the radius/2. This means if the object has a radius of 1 meter, the distance from the object should be less than 50cm.
Note that polished and shiny metal surface objects cannot accurately be measured with the IR temperature sensor.

Accelerometer

The accelerometer measures acceleration in 3 axes with programmable resolution up to 14 bit. The accelerometer can also measure direction of gravity unlike a gyroscope that can only measure change of direction.

Humidity Sensor

The humidity sensor reports the relative humidity (%RH) and temperature. This is an integrated 12-bit relative humidity sensor with a 14-bit temperature sensor. In order to ensure the best performance of relative humidity and temperature measurement, there are some basic rules of design. The sensor requires direct contact with ambient air (like that provided by a hole in the enclosure), while at the same time the sensor should be airtight towards the inner volume of the device.

Magnetometer/Compass

The magnetometer measures the magnetic field in 3 axes. The data from the magnetometer is usually combined with an accelerometer to make a compass. Magnetometers can be used to measure the earth's magnetic field but also to measure local magnetic fields created by electronics.

Barometric Pressure Sensor

The pressure sensor measures the barometric air pressure. This is a 16-bit pressure sensor with 16-bit temperature reading.

Gyroscope

The gyroscope measures the rate of rotation in all 3 axes (X, Y, Z) with up to 16-bit resolution. Readings from the gyroscope can be combined with readings from the accelerometer to determine the orientation of the SensorTag in 3-D space.

Assembly

Play around with the sensor tag and assemble it together. Follow the quick start guide that came with your device. An electronic pdf version can also be found here:

http://www.ti.com/lit/ml/swru324b/swru324b.pdf

Fully assembled:
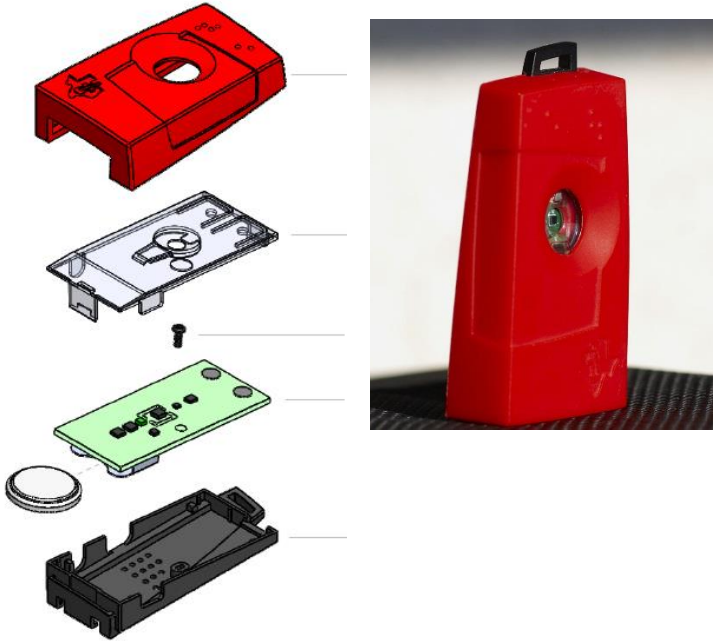
Syncing and Compatibility

Connect and sync the TI Sensor Tag to your device. Enable Bluetooth for your device (laptop/tablet/smart phone), then press the side button to begin transmission.

 The pairing password is '0'.

Supported iOS Devices

To use the SensorTag, a Bluetooth Smart (4.0 or newer) device is required, and the API of the device must support the Bluetooth low energy API. Currently the following iOS devices are supported:

- iPhone 4S and newer
- iPad(3) and newer
- iPad mini
- iPod Touch (5th generation)

If You Are Using **OSX**

Newer Macs should support Bluetooth LE. If you Mac supports Bluetooth 4.0, then chances are it supports Bluetooth LE as well. If you are using OSX, you need to make one configuration change so your Mac can connect to the Sensor Tag. Run the following command in a terminal window and restart your machine.

$ sudo nvram bluetoothHostControllerSwitchBehavior="never"

Source: https://github.com/IBM-Bluemix/iot-sensor-tag/tree/master/publish

If You Are Using Linux

If you are using Linux then you need to install some software packages in order to connect to the Sensor Tag. Run the following commands in a terminal window:

```
$ apt-get install bluez
$ apt-get install libbluetooth-dev
```

datasciencedojo
unleash the data scientist in you

Supported Android Devices

To use the SensorTag, a Bluetooth Smart (4.0 or newer) device is required, and the Android API 18 (Android 4.3) must be supported. The SensorTag app has been tested on the following Android devices:

- Nexus 4 (JWR66V)
- Nexus 7 (2012) (JWR66V)
- Nexus 7 (2013) (JSS15J)
- Nexus 5 (KTU84M)
- Nexus 9
- Samsung S4 (JWR66V.S11.130708)
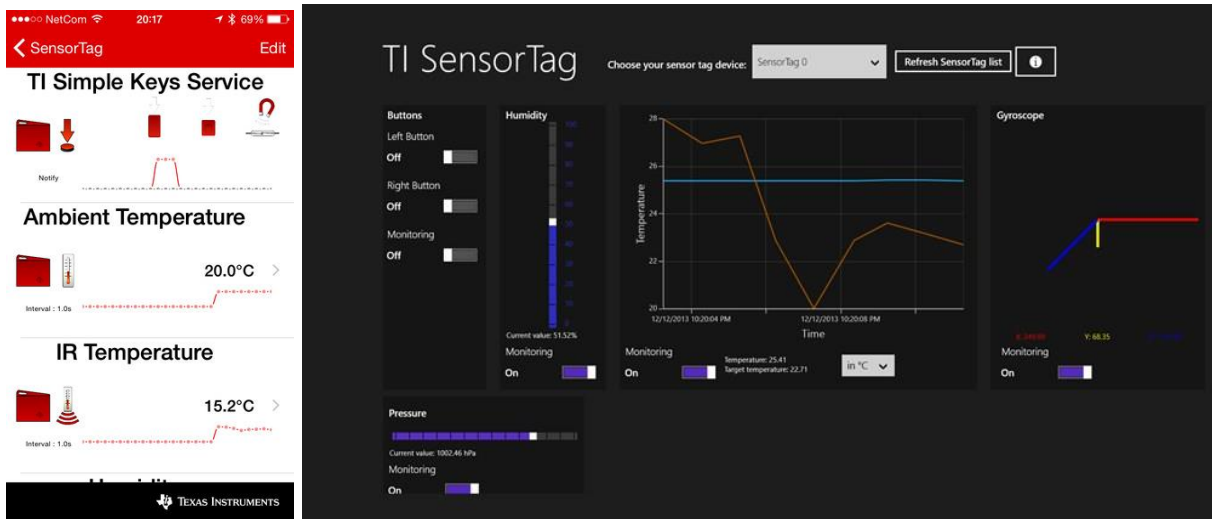- HTC One (M7), HTC One M8, HTC One M9
- Sony Xperia E1

TI Sensor Tag App

Sync your Sensor Tag to your smart device (iPhone/tablet/laptop) and download the app. There is an app for iSO, Windows 8, and Android devices. A simple Google search for your device or operating system followed by "TI Sensor Tag" will also yield open source results.

Apple: https://itunes.apple.com/app/ti-ble-sensortag/id552918064?l=nb&mt=8

Droid: https://play.google.com/store/apps/details?id=com.ti.ble.sensortag

Windows 8: http://apps.microsoft.com/windows/en-us/app/ti-sensor-tag-reader/c5218f45-f779-41d9-a5ca-4624df94613d
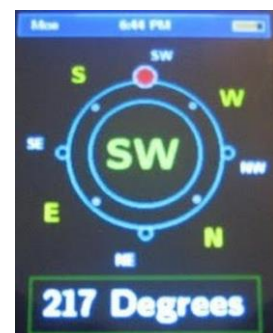

Spend some time and play around with the TI Sensor Tag. Blow on it to see humidity spike or wave it around to affect the gyroscope and accelerometer.

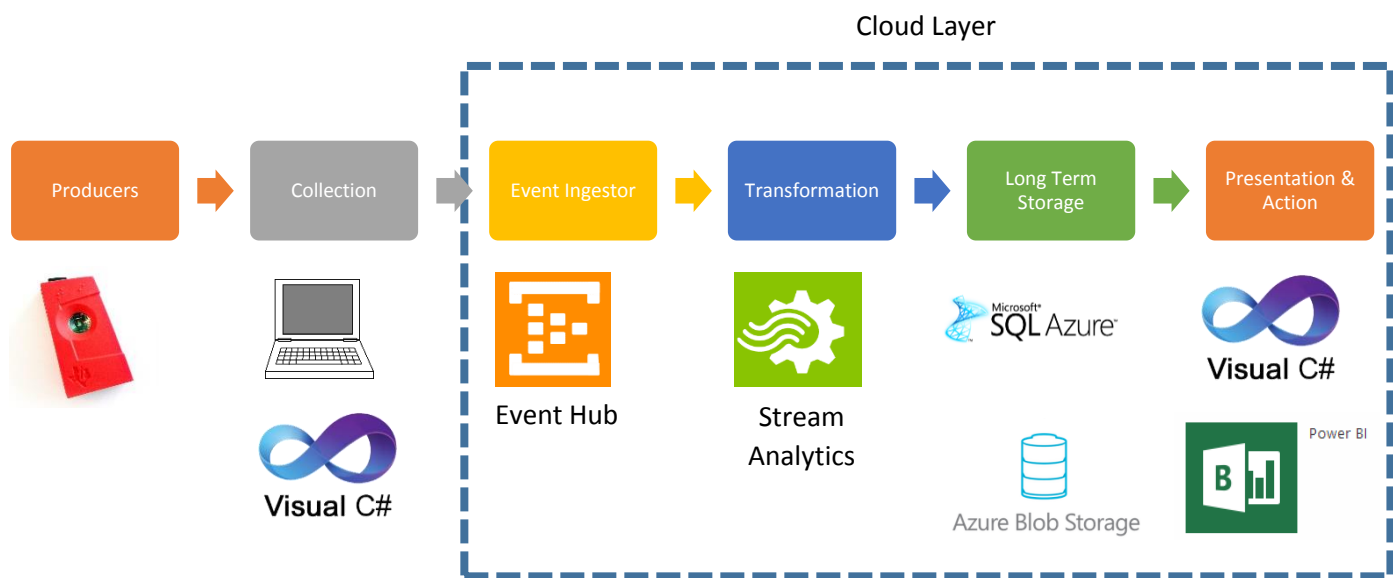Put it near your coffee to see if temperatures rise.

Accelerometer used as a construction leveler.

Magnetometer + Gyroscope = Compass

Building an IoT Pipeline in the Cloud

Cloud Layer

Producers	Collection	Event Ingestor	Transformation	Long Term Storage	Presentation & Action

Visual C#

Event Hub

Stream Analytics

Microsoft SQL Azure

Visual C#

Azure Blob Storage

Power BI

     The internet of things (IoT) are where objects transfer data through the internet. Literally any object with software and network capabilities can become linked to the cloud. This creates a future where factories, houses, cars, planes, supply chains, power grids, and more can all become "smart". IoT is created through the orchestration of multiple sensors streaming to the cloud in real time and are hooked up to software and dashboards that will enable live automation or live analytics.

     The hack day will gently guide you through the creation of an IoT pipeline from start to finish. You will apply everything you learned in this bootcamp to achieve the end result. The above diagram highlights which technologies will be used at each stage of the big data pipeline. Everyone should work together to achieve this goal.
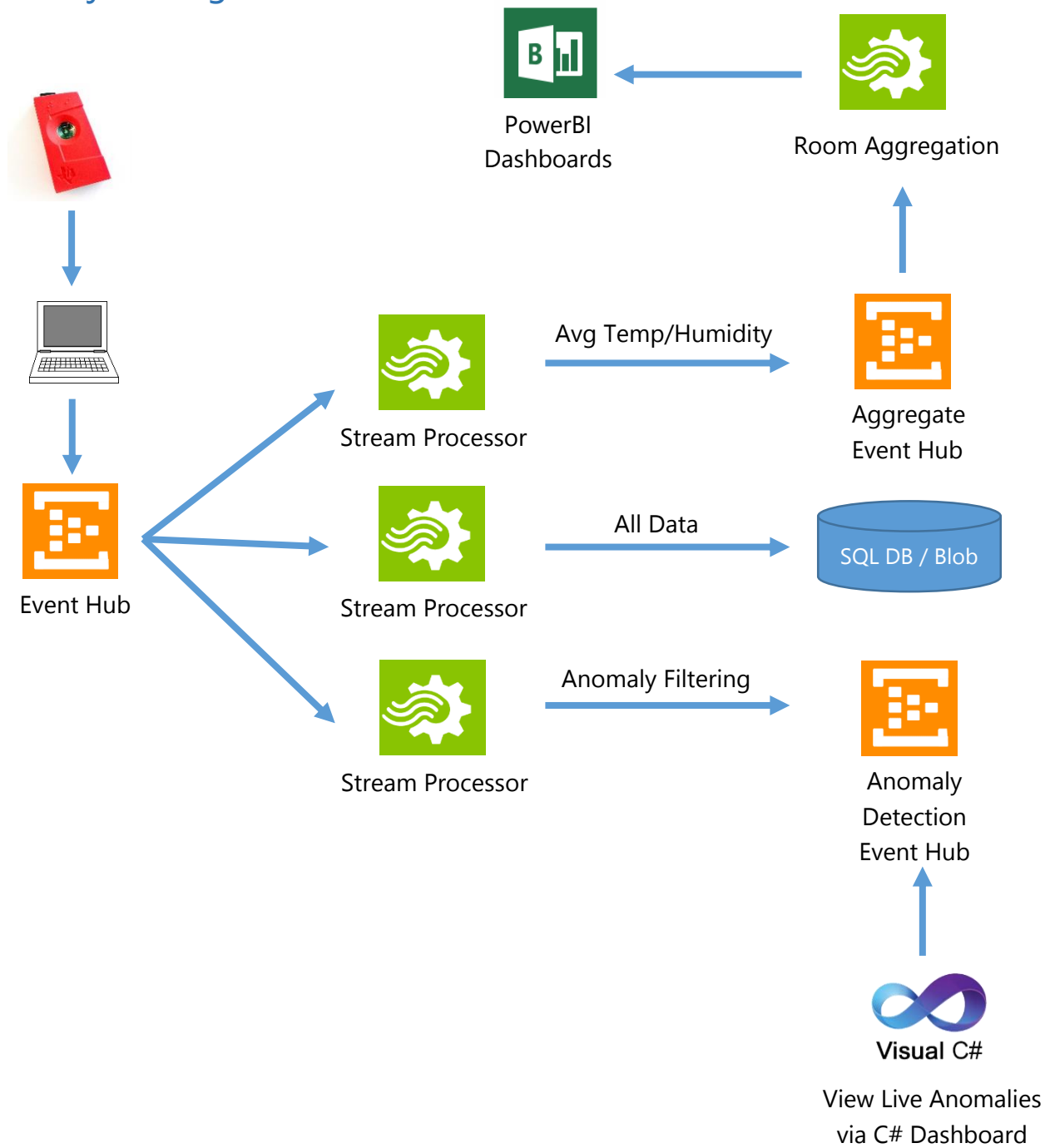
datasciencedojo
unleash the data scientist in you

Project Prerequisites

- Laptop:
  - Windows Operating System (or Windows Virtual Machine)
  - Bluetooth 4.0 enabled (hack day)
    - http://www.wikihow.com/Check-if-Your-Computer-Has-Bluetooth
  - Laptop must have full administrative rights
  - Company laptops often have firewalls: beware that this may block some developer tools
- Text Editor (also called IDE; choose **ONE** of the following recommendations below)
  - Notepad++: http://notepad-plus-plus.org/download/v6.7.5.html
  - Sublime Text 2: http://www.sublimetext.com/2
- Visual Studio (Free Trial, or Community Edition also works)
  - 90-day trial: https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx
  - .NET 4.5.2: http://www.microsoft.com/en-us/download/details.aspx?id=42642
  - Visual Studio 2010 or below may not work in a few instances
- An Azure 30-day free trial account, or an Azure account with administrative access rights
  - Free Trial: http://azure.microsoft.com/en-us/pricing/free-trial/
- *Optional*: Bluetooth enabled Android/iOS smart device (for the hack day IoT project)

Big Picture Overview of Project:

1) TI Sensor Tag measures humidity and temperature
2) C# App records and transmits readings to an Event Hub as a data stream
3) Event Hub collects and consolidates the data to be sent out
    a) Use C# dashboard to see Event Hub ingresses
4) Stream Analytics
    a) Filter and aggregate streams from Event Hub
5) Long term storage and data exploration:
    a) Output 1: Azure Blob Storage and AzureSQL DB
6) Room aggregations from all Sensor Tags
    a) Output 2a: Data Science Dojo Event Hub
        i) All bootcamp attendees will output to the Data Science Dojo Event Hub for mass aggregation of all TI Sensor Tags
    b) Output 2b: Data Science Dojo Stream Processors
        i) Stream processor will filter out individual device streams
        ii) Stream processor will mass aggregate all sensor tags
    c) Output 2c: Power BI dashboard for live updates
7) Build a cloud-based anomaly detection dashboard
    a) Output 3a: Anomaly Detection Stream Processor
        i) Use stream analytics to write an anomaly detection query
    b) Output 3b:
        i) Output the stream query results to a separate Event Hub that is dedicated for anomalies
        ii) Use C# dashboard to view live and incoming anomalies

Project Diagram

PowerBI Dashboards

Room Aggregation

Stream Processor — Avg Temp/Humidity → Aggregate Event Hub

Event Hub

Stream Processor — All Data → SQL DB / Blob

Stream Processor — Anomaly Filtering → Anomaly Detection Event Hub

Visual C#

View Live Anomalies via C# Dashboard

datasciencedojo
unleash the data scientist in you

Event Hub Hints & Questions

- **CREATE ALL AZURE RESOURCES IN WEST EUROPE**
  - Stream analytics is currently offered only in West EU and Central US. So that all the streams are in the same region, please provision all blob storages, SQL DBs, Event Hubs, and Stream Processors in **West Europe**.
- You will need an Event Hub that is in charge of ingesting raw data from the TI Sensor Tag
  - Why is an Event Hub required here?
  - Could this pipeline be built without this event hub?
  - Why are we not sending the device data directly into a SQL database?
  - What are the benefits of having an event ingestor?
- Event Hub will require an access port for receiving (listening) data, also called a "Shared Access Policy"
  - Why does each Shared Access Policy come with 2 policy keys?
  - Why are Shared Access Policies necessary?
  - Let's say your client was streaming data from your Event Hub and now you and your client have moved on and parted ways. How do you stop that client from accessing your Event Hub's data?
  - How many days will the Event Hub store your data?
  - What happens if the output of your Event Hub goes down? How much data will be lost?

datasciencedojo
unleash the data scientist in you

Transmitting to Your Event Hub

Now that we have an Event Hub ready to receive data, let's transmit to it. Do this by syncing the TI Senor Tag to your computer and utilizing an app to read and transmit the data.

- This app will require an Azure Blob storage as an output, create an Azure Blob storage with a container in West Europe
- We have compiled a C# app that will read from the TI Sensor Tag and output to an Event Hub
- Folder: **bootcamp > Hack Day > TISensorToEventHub_WindowsForm-master**
  - o The source code is available if you want to continue this project after the bootcamp
  - o Other languages such as Java, Python, Node.js, C, and JavaScript can be found in the "additional readings" portion of this guide
  - o App: **DeploymentFiles > SensorTagToEventHub.exe**
- Sync the TI Sensor Tag and fill in the credentials to your Event Hub
  - o **SensorName**: input your name as a string. This is how we will differentiate each other's TI Sensor Tags when we do full attendee aggregations.
    - ▪ (ex, Bruce Wayne → Bruce WayneTag)
  - o Tip #1: If you get tired of filing out your credentials every time, edit the following file with your credentials instead:
    - ▪ **DeploymentFiles > SensorTagToEventHub.exe.config**
  - o Tip #2: Create a shortcut of SensorTagToEventHub.exe for quick re-launch
  - o What format is the data being sent in? (CSV? TSV? JSON? XML?)

Event Hub Desktop Dashboard

The Azure Management Portal has a dashboard to view activities of Event Hubs, however there is a 15 minute delay between data received and the dashboard display. It also lacks the ability to peer at messages being received from the Event Hub; it only keeps a count of events coming in, in the form of "messages". Luckily, we can easily build an app that will display information from the Event Hub.

1. In Visual Studio, create a new Visual C# Desktop App project using the **Console Application** project template. Name the project **Receiver**.
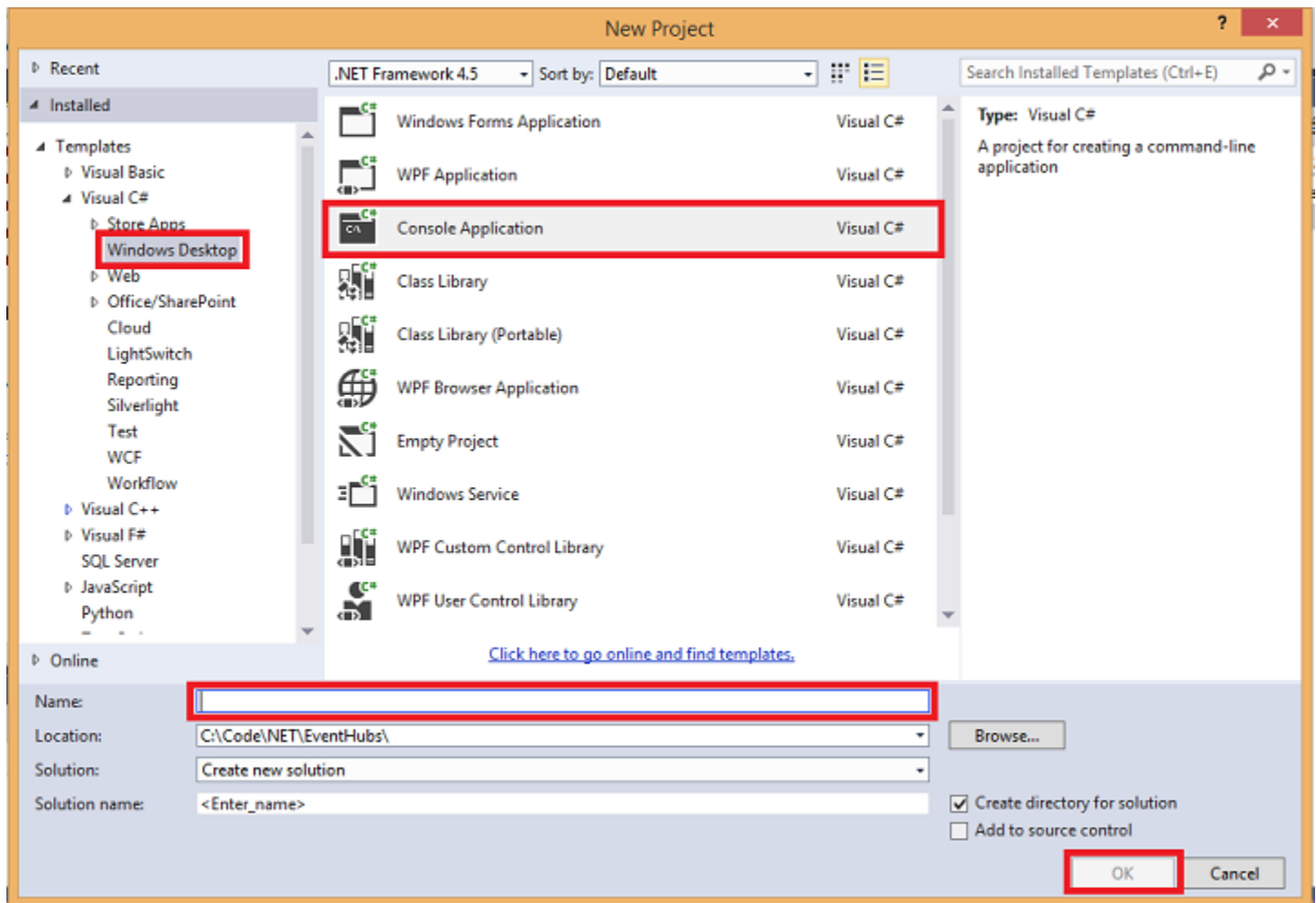
2. In Solution Explorer, right-click the solution, and then click **Manage NuGet Packages**.

   The **Manage NuGet Packages** dialog box appears.

datasciencedojo
unleash the data scientist in you

3. Search for Microsoft Azure Service Bus Event Hub – EventProcessorHost, click **Install**, and accept the terms of use.
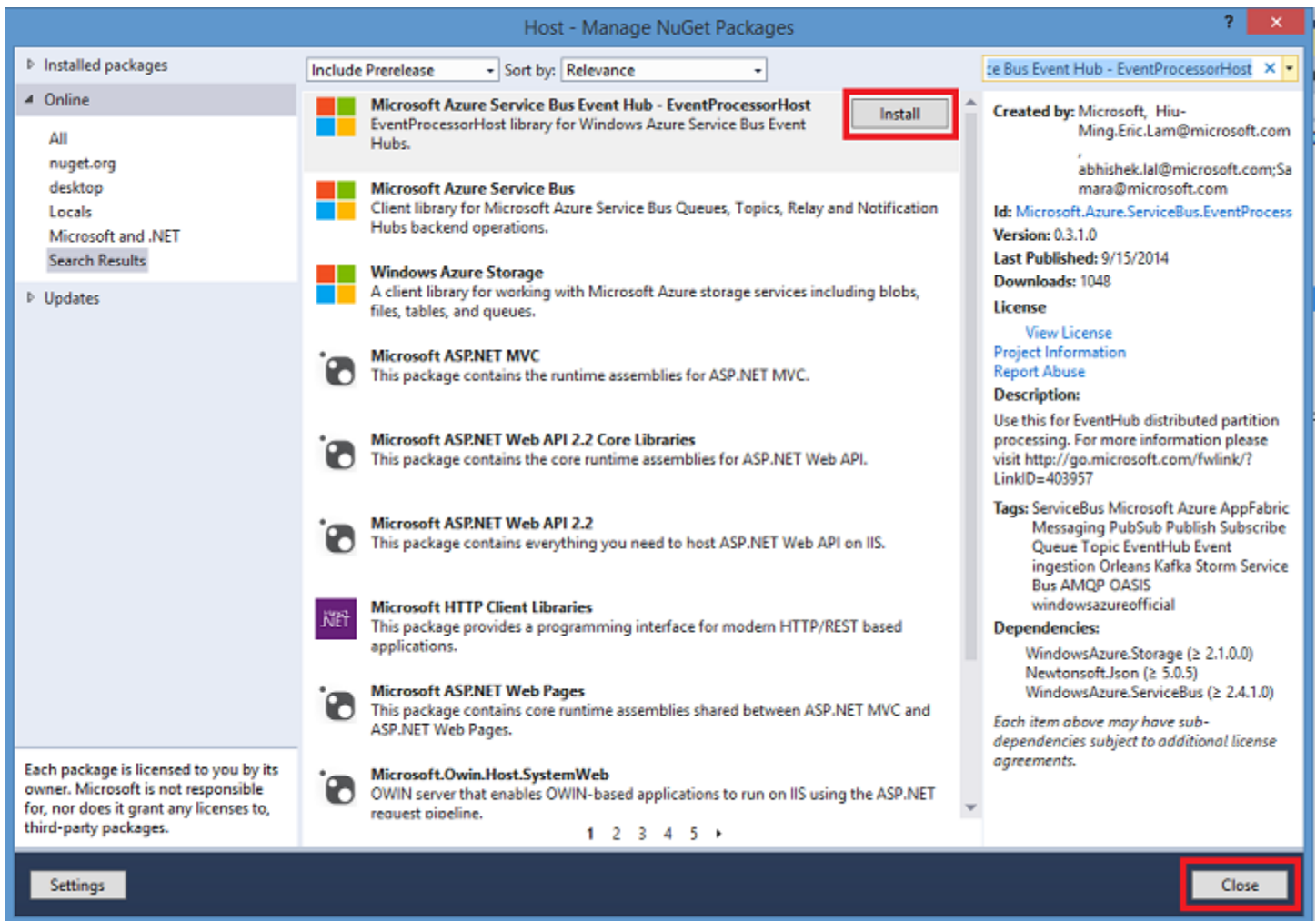
This downloads, installs, and adds a reference to the Azure Service Bus Event Hub - EventProcessorHost NuGet package, with all its dependencies.

4. Create a new class called **SimpleEventProcessor**. Add the following statements at the top of the file:

```
using Microsoft.ServiceBus.Messaging;
using System.Diagnostics;
using System.Threading.Tasks;
```

Then, insert the following code as the body of the class:

```
class SimpleEventProcessor : IEventProcessor{
    Stopwatch checkpointStopWatch;

    async Task IEventProcessor.CloseAsync(PartitionContext context, CloseReason
reason){
        Console.WriteLine(string.Format("Processor Shuting Down.  Partition '{0}',
Reason: '{1}'.", context.Lease.PartitionId, reason.ToString()));
        if (reason == CloseReason.Shutdown){
            await context.CheckpointAsync();
        }
    }

    Task IEventProcessor.OpenAsync(PartitionContext context){
        Console.WriteLine(string.Format("SimpleEventProcessor initialize.  Partition:
'{0}', Offset: '{1}'", context.Lease.PartitionId, context.Lease.Offset));
        this.checkpointStopWatch = new Stopwatch();
        this.checkpointStopWatch.Start();
        return Task.FromResult<object>(null);
    }

    async Task IEventProcessor.ProcessEventsAsync(PartitionContext context,
IEnumerable<EventData> messages){
        foreach (EventData eventData in messages){
            string data = Encoding.UTF8.GetString(eventData.GetBytes());

            Console.WriteLine(string.Format("Message received.  Partition: '{0}', Data:
'{1}'",
                context.Lease.PartitionId, data));
        }

        //Call checkpoint every 5 minutes, so that worker can resume processing from
the 5 minutes back if it restarts.
        if (this.checkpointStopWatch.Elapsed > TimeSpan.FromMinutes(5)){
            await context.CheckpointAsync();
            this.checkpointStopWatch.Restart();
        }
    }
}
```

This class will be called by the **EventProcessorHost** to process events received from the Event Hub. Note that the SimpleEventProcessor class uses a stopwatch to periodically call the checkpoint method on the **EventProcessorHost** context. This ensures that, if the receiver is restarted, it will lose no more than five minutes of processing work.

5. In the **Program** class, add the following using statements at the top:

```
using Microsoft.ServiceBus.Messaging;
using System.Threading.Tasks;
```

Then, add the following code in the **Main** method, substituting the Event Hub name and connection string, and the storage account and key that you copied in the previous sections:
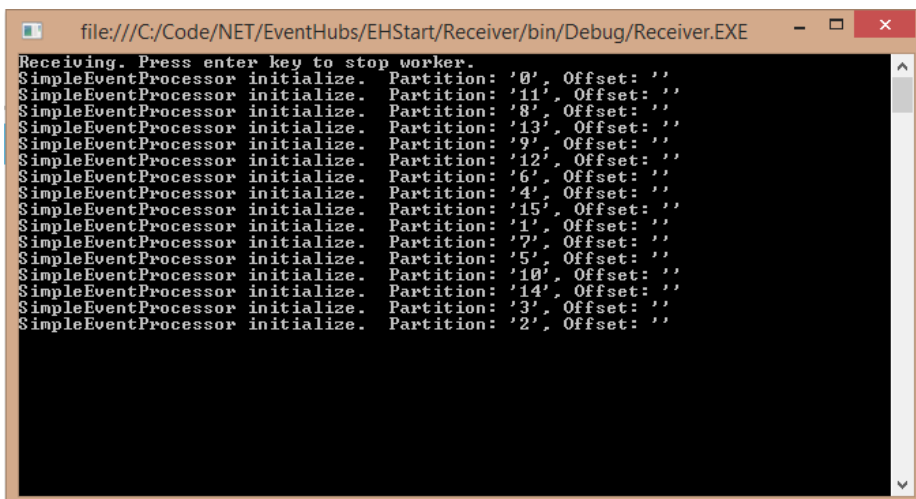
```
string eventHubConnectionString = "{event hub connection string}";
string eventHubName = "{event hub name}";
string storageAccountName = "{storage account name}";
string storageAccountKey = "{storage account key}";
string storageConnectionString =
string.Format("DefaultEndpointsProtocol=https;AccountName={0};AccountKey={1}",
        storageAccountName, storageAccountKey);

string eventProcessorHostName = Guid.NewGuid().ToString();
EventProcessorHost eventProcessorHost = new EventProcessorHost(eventProcessorHostName,
eventHubName, EventHubConsumerGroup.DefaultGroupName, eventHubConnectionString,
storageConnectionString);
eventProcessorHost.RegisterEventProcessorAsync<SimpleEventProcessor>().Wait();

Console.WriteLine("Receiving. Press enter key to stop worker.");
Console.ReadLine();
```

Now you are ready to run the applications.

From within Visual Studio, run the **Receiver** project, then wait for it to start the receivers for all the partitions.

datasciencedojo
unleash the data scientist in you

Stream Processor Hints

Our data is in the cloud, so we can utilize the full power of the cloud to aggregate, filter, and process it live and incoming. We will use Stream Analytics for this project.

- Review the Stream Analytics slides
- Remember that each query is associated with a stream processor, also called a "job"
- Each Job may only have 1 output, but multiple inputs
- You can have multiple stream processors
- Always test your query first before you fire off a stream job
    - Stream analytics won't tell you why it's not working
- You can download sample data from your stream processor's input. This step is **HIGHLY** recommended!
    - Why might sample data be important in testing query logic?
- Azure Stream Query Language Skeleton (order of operations matters!)

```
SELECT
    System.Timestamp AS AttributeName
INTO
    YourOutput
FROM
    YourStreams
JOIN
ON
TIMESTAMP BY
    YourTimePK
WHERE
GROUP BY
    Window/Attribute
HAVING
```

- List of AzureQL Aggregation Functions: https://msdn.microsoft.com/en-us/library/azure/dn931787.aspx
- Questions:
    - What's the difference between "where" and "having"?
    - What's the difference between hoping, sliding, and tumbling windows?
    - What happens if you don't set a window for your query?
    - What happens if you don't set a "Timestamp by"? What are the implications? Does it matter?
    - What's so special about the "having" clause within Azure Stream Query language?
    - If you wrote a stream query that aggregated and counted events during a window, and no events occurred during the window, what would happen? Why do you think it was designed this way?
    - Look at your test data. What is the difference between "time", "EventProcessedUtcTime", and "EventEnqueuedUtcTime"? Also, what is "PartitionID"?
    - What should the "job start time" be set to?

Practicing Stream Queries

Try writing the following queries on your test data or the test data provided in the Hack Day folder:

- Return all the contents of the Event Hub **once**.
- Return the average temperature every 3 seconds, from past and future readings indefinitely.
- Return the average temperature for the past 3 seconds, but open a window up every 1 second, from past and future readings indefinitely.
- Return descriptive statistics for temperature every 3 seconds from past and future readings indefinitely. Description statistics being (average, minimum, number of readings, max temperature, and standard deviation).
- Return descriptive statistics for humidity every 3 seconds from past and future readings indefinitely. Description statistics being (average, minimum, number of readings, max temperature, and standard deviation).
- Combine the last two queries about descriptive statistics together into one query.
- Return min temperature, max temperature, and the change in temperature between the min and max temperature of 3 second tumbling windows.
- Write a query that returns the min and max time for events inside of 3 second tumbling windows and report the difference between the min and max time in seconds. Hint: datediff()

Output to a Blob Storage & Azure SQL Database

Fire off your stream processor after you have tested your query logic against sample data. Output to an Azure Blob Storage. View the results.

Outputting to an Azure SQL Database will be a bit trickier because it will require some T-SQL to create some blank tables. There are a few templates inside of the StreamQL repository on how to create a table.

3 Methods of Create a Table:

- Visual Studio -> Tools -> Connect to a Database
- Azure ML Reader Module
  - Please note that when creating a table, the module displays as fail, but the tables will still be created

Use the CREATE TABLE command. Insert your schema and table name. A list of data types for T-SQL can be found here:

https://msdn.microsoft.com/en-us/library/ms187752.aspx

You will then need to create a clustered index for your table. In this case, let your WindowEnd timestamps be your clustered keys.

Creating a table skeleton and a clustered Index:

```
CREATE TABLE YourTableName(

        ColName1 DATETIME2,

        ColName2 decimal(5,2),

        ColName3 float,

        ColName4 bigint,

        ColName5 nvarchar(max)

)

IF (NOT EXISTS (SELECT * FROM SYS.INDEXES

        WHERE NAME = 'IX_YourTable_WindowEnd))

BEGIN

CREATE CLUSTERED INDEX [IX_YourTable_WindowEnd]

   ON [dbo].[YourTable]([WindowEnd] ASC);

END
```

Group Event Hub

Partner up with 2-5 people. Share your Event Hub information with one another and add their Event Hubs as additional stream inputs.

- Write queries to vertically join (UNION) two queries from two separate streams.
- Write queries that would detect when both sensors return different readings from one another.

Real World Scenarios, Group Exercise

- You own a fleet of dairy delivery trucks whose internal freezer temperatures need to be maintained at 22°-24°. How would you write a query that returns the status update only for freezers who deviated from this range?

- You're in charge of 12 temperature and humidity controlled bio-domes in Eastern Washington that grow pineapples. Pineapples need humidity between 50% and 70% and temperature between 68°F and 86°F. If temperatures go below 68°F, plant growth will slow. Write a query that would notify you every time either the humidity or temperature deviated out of the optimal range. Hint: OR clause.

- You're a NOAA technician in charge of an array of storm detection buoys. Sudden drops in temperature and sudden spikes in humidity are a good indicator that a storm is forming. Likewise, sudden rising temperatures and spikes in humidity are a sign of a forming thunderstorm. Write queries to address these two scenarios.

- Static electricity build up in data centers often causes spontaneous shutdown of computers when discharge occurs, often resulting in irreversible damage. Likewise, too high of humidity will result in condensation, hardware corrosion, and early system or component failures. Operating computer equipment at high temperatures for extended periods of time also greatly reduce reliability and longevity of components. It is best practice to maintain a temperature of 68° to 75°F operating temperature. Computer equipment should not be operated if temperatures exceed 85°F. Early warning alerts should also occur if humidity deviates from 40-60%, and critical alerts be sent out if humidity deviates from 30-70% range. You're in charge of maintaining that your data centers have 99.9% uptime. What do you do?

Building an Anomaly Detection App

Hints:

- Write a stream query that would open up a window when humidity spikes.
- Link the output to a separate event hub that is dedicated to just receiving anomalies.
- Clone the C# Event Hub receiver app to read from the new Event Hub.
- Blow into the TI Sensor Tag to receive activities in the dashboard.
- Optional: rewrite the C# app so that it pops up a message box alerting you of 'critical' anomalies.
    - Hint: MessageBox.Show("My Message");
- After the bootcamp:
    - Reconfigure the C# app to send someone an email when there's a critical anomaly.
    - Guide to sending email via C#: http://csharp.net-informations.com/communications/csharp-smtp-mail.htm

datasciencedojo
unleash the data scientist in you

Room Aggregations and PowerBI

We will consolidate all sensors together in the same room and report them to PowerBI.

- Create a Stream Processor that will output to the Data Science Dojo Event Hub.
  - Service Bus Namespace: **sensortaghub-ns**
  - Event Hub Name: **aggregatesensorhub**
  - Shared Access Policy Name: **masterkey**
  - Shared Access Key: **Iw9aXCTiOC8fQaxsqkpUzFswCKNUj4aPrLqadt4gWRY=**
- Send your stream into the aggregate event hub with a stream processor, using the following query:
  -

```
SELECT
    *
From
    MySensorStream
  TIMESTAMP BY
        time
```

Additional Readings

For information on how to program each of these sensors, refer to the Wikipedia article written below (**C/Java**): http://processors.wiki.ti.com/index.php/SensorTag_User_Guide

The sensor tag also supports Node.js, Python, JavaScript, and C#.

Building iPhone/iPad electronic projects book: http://shop.oreilly.com/product/0636920029281.do?cmp=af-code-book-product_cj_9781449363505_6760607

Locationing with Apple iBeacon app: http://itunes.apple.com/app/locationing-with-ibeacon/id852315723?mt=8&uo=4

datasciencedojo
unleash the data scientist in you