paolosalvatori

**How to use Service Fabric to read telemetry data from an Event Hub and store it to Azure SQL Database using JSON functionalities**

This sample shows how to use a Service Fabric service hosting an **EventProcessorHost** listener to retrieve events from an **Event Hub** and store them in a batch mode to an **Azure SQL Database** using the [OPENJSON](https://msdn.microsoft.com/en-us/library/dn921885.aspx) function. The solution demonstrates how the use the following techniques:

* Send events to an [Event Hub](https://msdn.microsoft.com/en-us/library/azure/dn789973.aspx) using both AMQP and HTTPS transport protocols.
* Create an entity level shared access policy with only the Send claim. This key will be used to create SAS tokens, one for each publisher endpoint.
* Issue a SAS token to secure individual publisher endpoints.
* Use a SAS token to authenticate at a publisher endpoint level.
* Create an [EventProcessorHost](https://msdn.microsoft.com/en-us/library/microsoft.servicebus.messaging.eventprocessorhost(v=azure.95).aspx) listener for a Service Fabric stateless service to retrieve and process events from an event hub.
* Use the [OPENJSON](https://msdn.microsoft.com/en-us/library/dn921885.aspx) table-value function in a stored procedure to process a batch of rows.

**NOTE**: this article is not intended to provide an exhaustive analysis of the various batching techniques offered by Azure SQL Database. Relying on batching to optimize data ingestion is a topic by itself, if you’re interested in the details take a look at this dedicated article: [How to use batching to improve SQL Database application performance](https://azure.microsoft.com/en-us/documentation/articles/sql-database-use-batching-to-improve-performance/). Also look at [How to store Event Hub events to Azure SQL Database](https://code.msdn.microsoft.com/How-to-integrate-store-828769eb) for a version of the sample where the event processor uses a stored procedure with a [Table-Valued Parameter](https://msdn.microsoft.com/en-us/library/bb675163%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396) to store multiple events in a batch mode to a table on an Azure SQL database.

# Scenario

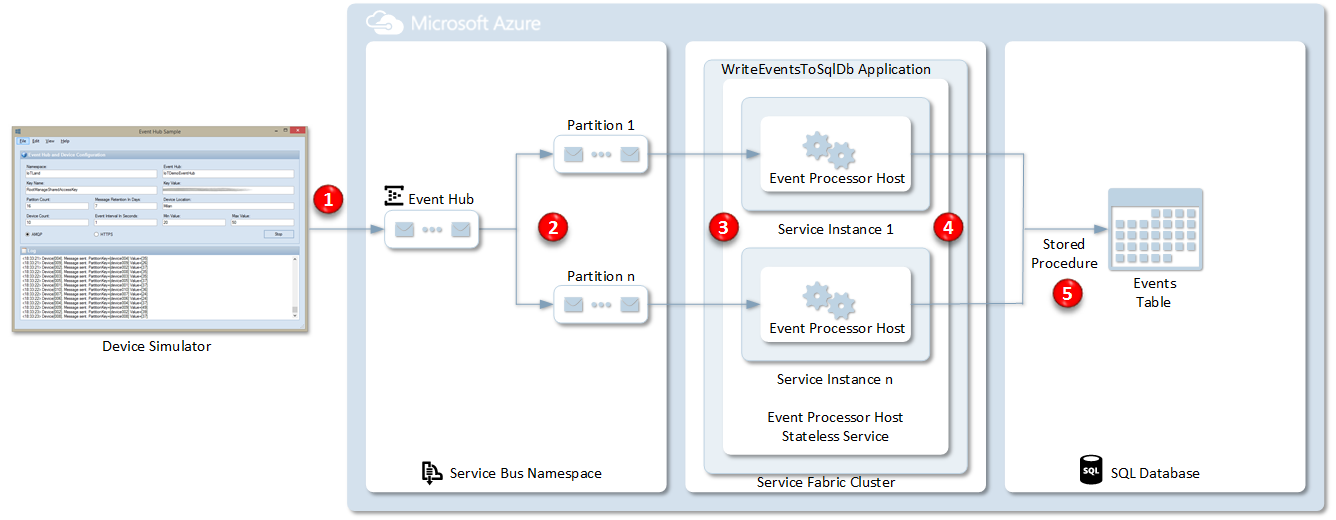
This solution simulates an Internet of Things (IoT) scenario where thousands of devices send events (e.g. sensor readings) to a backend system via a message broker. The backend system retrieves events from the messaging infrastructure and store them to a persistent repository in a scalable manner.

# Architecture

The sample is structured as follows:

1. A client application can be used to create an event hub and an entity level shared access policy with only the **Send** access right. The same application can be used to simulate a configurable amount of devices that send readings into the event hub. Each device uses a separate publisher endpoint to send data to the underlying event hub and a separate SAS token to authenticate with the **Service Bus** namespace.
2. An **Event Hub** is used to ingest device events.
3. A **Service Fabric** stateless service with multiple instances uses an [EventProcessorHost](https://msdn.microsoft.com/en-us/library/microsoft.servicebus.messaging.eventprocessorhost(v=azure.95).aspx) to read and process messages from the partitions of the event hub.
4. The custom [EventProcessor](https://msdn.microsoft.com/en-us/library/microsoft.servicebus.messaging.ieventprocessor.aspx) class inserts a collection of events into a table of a **SQL Database** in a batch mode by invoking a stored procedure.
5. The stored procedure uses the [OPENJSON](https://msdn.microsoft.com/en-us/library/dn921885.aspx) table-value function and the [MERGE](https://msdn.microsoft.com/en-us/library/bb510625.aspx) statement to implement an **UPSERT** mechanism.

The following picture shows the architecture of the solution:



# References

JSON Functionalities of Azure SQL Database

* [JSON in SQL Server 2016: Part 1 of 4](https://blogs.technet.microsoft.com/dataplatforminsider/2016/01/05/json-in-sql-server-2016-part-1-of-4/)
* [Channel9 Video: SQL Server 2016 and JSON Support](https://channel9.msdn.com/Shows/Data-Exposed/SQL-Server-2016-and-JSON-Support)
* [Reference Documentation](https://msdn.microsoft.com/en-us/library/dn921897.aspx)

Event Hubs

* [Event Hubs](http://azure.microsoft.com/en-us/services/event-hubs/)
* [Get started with Event Hubs](http://azure.microsoft.com/en-us/documentation/articles/service-bus-event-hubs-csharp-ephcs-getstarted/)
* [Event Hubs Programming Guide](https://msdn.microsoft.com/en-us/library/azure/dn789972.aspx)
* [Service Bus Event Hubs Getting Started](https://code.msdn.microsoft.com/windowsazure/Service-Bus-Event-Hub-286fd097)
* [Event Hubs Authentication and Security Model Overview](https://msdn.microsoft.com/en-us/library/azure/dn789974.aspx)
* [Service Bus Event Hubs Large Scale Secure Publishing](https://code.msdn.microsoft.com/windowsazure/Service-Bus-Event-Hub-99ce67ab)
* [Service Bus Event Hubs Direct Receivers](https://code.msdn.microsoft.com/windowsazure/Event-Hub-Direct-Receivers-13fa95c6)
* [Service Bus Explorer](https://code.msdn.microsoft.com/windowsapps/Service-Bus-Explorer-f2abca5a)
* [Episode 160: Event Hubs with Elio Damaggio](http://channel9.msdn.com/Shows/Cloud+Cover/Episode-160-Event-Hubs-with-Elio-Damaggio) (video)
* [Telemetry and Data Flow at Hyper-Scale: Azure Event Hub](http://channel9.msdn.com/Events/TechEd/Europe/2014/CDP-B307) (video)
* [Data Pipeline Guidance](https://github.com/mspnp/data-pipeline)  (Patterns & Practices solution)
* [Event Processor Host Best Practices Part 1](http://blogs.msdn.com/b/servicebus/archive/2015/01/16/event-processor-host-best-practices-part-1.aspx)
* [Event Processor Host Best Practices Part 2](http://blogs.msdn.com/b/servicebus/archive/2015/01/21/event-processor-host-best-practices-part-2.aspx)
* [How to create a Service Bus Namespace and an Event Hub using a PowerShell script](http://blogs.msdn.com/b/paolos/archive/2014/12/01/how-to-create-a-service-bus-namespace-and-an-event-hub-using-a-powershell-script.aspx)

# Visual Studio Solution

The Visual Studio solution includes the following projects:

* **CreateIoTDbWithMerge.sql**: this script can be used to create the **SQL Database** used to store device events.
* **Entities**: this library contains the **Payload** class. This class defines the structure and content of the [EventData](https://msdn.microsoft.com/en-us/library/microsoft.servicebus.messaging.eventdata.aspx?f=255&MSPPError=-2147217396) message body.
* **EventProcessorHostService**: this library defines the stateless service used to handle the events from the event hub.
* **DeviceSimulator**: this **Windows Forms** application can be used to create the **Event Hub** used by the sample and simulate a configurable amount of devices sending telemetry events to the IoT application.
* **WriteEventsToAzureSqlDatabase**: this project defines the **Service Fabric** application used to handle the events from the event hub.

**NOTE**: To reduce the size of the zip file, I deleted the NuGet packages. To repair the solution, make sure to right click the solution and select **Enable NuGet Package Restore**. For more information on this topic, see the following [post](http://blogs.4ward.it/enable-nuget-package-restore-in-visual-studio-and-tfs-2012-rc-to-building-windows-8-metro-apps/).

# Solution

This section briefly describes the individual components of the solution.

## SQL Azure Database

Run the **CreateIoTDbWithMerge.sql** script to create the database used by the solution. In particular, the script create the following artifacts:

* The **Events** table used to store events.
* The **sp\_InsertJsonEvents** stored procedure used to store events.

The stored procedure receives a single input parameter of type **nvarchar(max)** which contains the events to store in **JSON** format and uses the [MERGE](https://msdn.microsoft.com/en-us/library/bb510625.aspx) statement to implement an **UPSERT** mechanism. This technique is commonly used to implement idempotency: if an row already exists in the table with the a given EventId, the store procedure updates its columns, otherwise a new record is created. The stored procedure uses the [OPENJSON](https://msdn.microsoft.com/en-us/library/dn921885.aspx) table-value function that parses JSON text and returns objects and properties in JSON as rows and columns. [OPENJSON](https://msdn.microsoft.com/en-us/library/dn921885.aspx) provides a rowset view over a JSON document, with the ability to explicitly specify the columns in the rowset and the property paths to use to populate the columns. Since OPENJSON returns a set of rows, you can use [OPENJSON](https://msdn.microsoft.com/en-us/library/dn921885.aspx) function in FROM clause of Transact-SQL statements like any other table, view, or table-value function. The [OPENJSON](https://msdn.microsoft.com/en-us/library/dn921885.aspx) function is available only under compatibility level 130. If your database compatibility level is lower than 130, SQL Server will not be able to find and execute OPENJSON function. Other JSON functions are available at all compatibility levels. You can check compatibility level in sys.databases view or in database properties. You can change a compatibility level of database using the following command: **ALTER DATABASE DatabaseName SET COMPATIBILITY\_LEVEL = 130**. Note that compatibility level 120 might be default even in new Azure SQL Databases. For more information on the new JSON support in Azure SQL Database, see [JSON functionalities in Azure SQL Database](https://azure.microsoft.com/en-us/blog/json-functionalities-in-azure-sql-database-public-preview/).

USE IoTDB

GO

-- Drop sp\_InsertJsonEvents stored procedure

DROP PROCEDURE IF EXISTS [dbo].[sp\_InsertJsonEvents]

GO

-- Drop Events table

DROP TABLE IF EXISTS [dbo].[Events]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

-- Create Events table

CREATE TABLE [dbo].[Events](

[EventId] [int] NOT NULL,

[DeviceId] [int] NOT NULL,

[Value] [int] NOT NULL,

[Timestamp] [datetime2](7) NULL,

PRIMARY KEY CLUSTERED

(

[EventId] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

-- Create sp\_InsertEvents stored procedure

CREATE PROCEDURE dbo.sp\_InsertJsonEvents

@Events NVARCHAR(MAX)

AS

BEGIN

MERGE INTO dbo.Events AS A

USING (SELECT \*

FROM OPENJSON(@Events)

WITH ([eventId] int, [deviceId] int, [value] int, [timestamp] datetime2(7))) B

ON (A.EventId = B.EventId)

WHEN MATCHED THEN

UPDATE SET A.DeviceId = B.DeviceId,

A.Value = B.Value,

A.Timestamp = B.Timestamp

WHEN NOT MATCHED THEN

INSERT (EventId, DeviceId, Value, Timestamp)

VALUES(B.EventId, B.DeviceId, B.Value, B.Timestamp);

END

GO

## Entities

The following table contains the code of the **Payload** class. This class is used to define the body of the messages sent to the event hub. Note that the properties of the class are decorated with the **JsonPropertyAttribute**. In fact, the code of both the client application and Service Fabric service use [Json.NET](http://www.newtonsoft.com/json) to serialize and deserialize the message content in JSON format.

#region Using Directives  
 using System;  
 using Newtonsoft.Json;   
 #endregion  
   
 namespace Microsoft.AzureCat.Samples.Entities  
 {  
 [Serializable]  
 public class Payload  
 {  
 /// <summary>  
 /// Gets or sets the device id.  
 /// </summary>  
 [JsonProperty(PropertyName = "eventId", Order = 1)]  
 public int EventId { get; set; }  
   
 /// <summary>  
 /// Gets or sets the device id.  
 /// </summary>  
 [JsonProperty(PropertyName = "deviceId", Order = 2)]  
 public int DeviceId { get; set; }  
   
 /// <summary>  
 /// Gets or sets the device value.  
 /// </summary>  
 [JsonProperty(PropertyName = "value", Order = 3)]  
 public int Value { get; set; }  
   
 /// <summary>  
 /// Gets or sets the event timestamp.  
 /// </summary>  
 [JsonProperty(PropertyName = "timestamp", Order = 4)]  
 public DateTime Timestamp { get; set; }  
 }  
 }

## EventProcessor

The following table contains the code of the **EventProcessor** class. In particular, the **ProcessEventsAsync** method writes events to an **Azure SQL Database** in a batch mode by invoking a stored procedure. Note how the code first extracts the payload from the **EventData** objects contained in the **events** collection and then serializes the resulting **List** collection into a JSON array using the **JsonConvert.SerializeObject** method. The string returned by this call is used as value of **@Events** parameter of the **sp\_InsertJsonEvents** stored procedure.

#region Using Directives

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Microsoft.ServiceBus.Messaging;

#endregion

namespace Microsoft.AzureCat.Samples.EventProcessorHostService

{

public class EventProcessor : IEventProcessor

{

#region Private Fields

private readonly string sqlDatabaseConnectionString;

private readonly string insertStoredProcedure;

private readonly int checkpointCount;

private int messageCount;

#endregion

#region Public Constructors

public EventProcessor(string sqlDatabaseConnectionString,

string insertStoredProcedure,

int checkpointCount)

{

if (string.IsNullOrWhiteSpace(sqlDatabaseConnectionString))

{

throw new ArgumentNullException($"{nameof(sqlDatabaseConnectionString)} parameter cannot be null");

}

if (string.IsNullOrWhiteSpace(insertStoredProcedure))

{

throw new ArgumentNullException($"{nameof(insertStoredProcedure)} parameter cannot be null");

}

this.sqlDatabaseConnectionString = sqlDatabaseConnectionString;

this.insertStoredProcedure = insertStoredProcedure;

this.checkpointCount = checkpointCount > 0 ? checkpointCount : 1;

}

#endregion

#region IEventProcessor Methods

public Task OpenAsync(PartitionContext context)

{

try

{

// Trace Open Partition

ServiceEventSource.Current.OpenPartition(context.EventHubPath,

context.ConsumerGroupName,

context.Lease.PartitionId);

}

catch (Exception ex)

{

// Trace Exception

ServiceEventSource.Current.Message(ex.Message);

}

return Task.FromResult<object>(null);

}

public async Task ProcessEventsAsync(PartitionContext context, IEnumerable<EventData> events)

{

try

{

if (events == null)

{

return;

}

var eventList = events.Select(e => Encoding.UTF8.GetString(e.GetBytes())).ToList();

if (!eventList.Any())

{

return;

}

// Trace Process Events

ServiceEventSource.Current.ProcessEvents(context.EventHubPath,

context.ConsumerGroupName,

context.Lease.PartitionId,

eventList.Count);

using (var sqlConnection = new SqlConnection(sqlDatabaseConnectionString))

{

await sqlConnection.OpenAsync();

// Create command

var sqlCommand = new SqlCommand(insertStoredProcedure, sqlConnection)

{

CommandType = CommandType.StoredProcedure

};

// Add table-valued parameter

sqlCommand.Parameters.Add(new SqlParameter

{

ParameterName = "@Events",

SqlDbType = SqlDbType.NVarChar,

Size = -1,

Value = GetJsonArray(eventList)

});

// Execute the query

await sqlCommand.ExecuteNonQueryAsync().ConfigureAwait(false);

}

// Increase messageCount

messageCount += eventList.Count;

// Invoke CheckpointAsync when messageCount => checkpointCount

if (messageCount < checkpointCount)

{

return;

}

await context.CheckpointAsync();

messageCount = 0;

}

catch (LeaseLostException ex)

{

// Trace Exception as message

ServiceEventSource.Current.Message(ex.Message);

}

catch (AggregateException ex)

{

// Trace Exception

foreach (var exception in ex.InnerExceptions)

{

ServiceEventSource.Current.Message(exception.Message);

}

}

catch (Exception ex)

{

// Trace Exception

ServiceEventSource.Current.Message(ex.Message);

}

}

public async Task CloseAsync(PartitionContext context, CloseReason reason)

{

try

{

// Trace Open Partition

ServiceEventSource.Current.ClosePartition(context.EventHubPath,

context.ConsumerGroupName,

context.Lease.PartitionId,

reason.ToString());

if (reason == CloseReason.Shutdown)

{

await context.CheckpointAsync();

}

}

catch (Exception ex)

{

// Trace Exception

ServiceEventSource.Current.Message(ex.Message);

}

}

#endregion

#region Private Methods

private string GetJsonArray(IReadOnlyList<string> list)

{

if (list == null || list.Count == 0)

{

return null;

}

var builder = new StringBuilder("[");

for (var i = 0; i < list.Count; i++)

{

builder.Append(i == 0 ? list[0] : $",{list[i]}");

}

builder.Append("]");

return builder.ToString();

}

#endregion

}

}

Application Configuration

Make sure to replace the following placeholders in the project files below before deploying and testing the application on the local development Service Fabric cluster or before deploying the application to your Service Fabric cluster on Microsoft Azure.

## Placeholders

This list contains the placeholders that need to be replaced before deploying and running the application:

* **[SqlDatabaseConnectionString]**: defines the connection string of the **SQL Database** containing the **Events** table.
* **[InsertStoredProcedure]**: contains the name of the **Stored Procedure** used to insert the events into the **SQL Database** table.
* **[ServiceBusConnectionString]**: defines the connection string of the **Service Bus** namespace that contains the **Event Hub** and **Queue** used by the solution.
* **[StorageAccountConnectionString]**: contains the connection string of the **Storage Account** used by the **EventProcessorHost** to store partition lease information when reading data from the input **Event Hub**.
* **[EventHubName]**: contains the name of the input **Event Hub**.
* **[ConsumerGroupName]**: contains the name of the **Consumer Group** used by the **EventProcessorHost** to read data from the input **Event Hub**.
* **[ContainerName]**: defines the name of the **Storage Container** where the **EventProcessorHost** writes **Append Blobs**.
* **[QueueName]**: contains the name of the **Service Bus Queue** used by the **EventProcessorHost** send a message to the external hot path analytics system when user session completes.
* **[CheckpointCount]**: this number defines after how many messages the **EventProcessorHost** invokes the **ChechpointAsync** method.
* **[EventHubClientNumber]**: this number defines how many [EventHubClient](https://msdn.microsoft.com/library/azure/microsoft.servicebus.messaging.eventhubclient.aspx) objects are contained in the connection pool of the **PageViewWebService**.

## Configuration Files

**ApplicationParameters.xml** file in the **PageViewTracer** project:

<?xml version="1.0" encoding="utf-8"?>  
 <Application xmlns:xsd="http://www.w3.org/2001/XMLSchema"   
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   
 Name="fabric:/PageViewTracer"   
 xmlns="http://schemas.microsoft.com/2011/01/fabric">  
 <Parameters>  
 <Parameter Name="EventProcessorHostService\_InstanceCount"   
 Value="1" />  
 <Parameter Name="EventProcessorHostService\_SqlDatabaseConnectionString"   
 Value="[SqlDatabaseConnectionString]" />  
 <Parameter Name="EventProcessorHostService\_InsertStoredProcedure"   
 Value="[InsertStoredProcedure]" />  
 <Parameter Name="EventProcessorHostService\_StorageAccountConnectionString"   
 Value="[StorageAccountConnectionString]" />  
 <Parameter Name="EventProcessorHostService\_ServiceBusConnectionString"   
 Value="[ServiceBusConnectionString]" />  
 <Parameter Name="EventProcessorHostService\_EventHubName"   
 Value="[EventHubName]" />  
 <Parameter Name="EventProcessorHostService\_ConsumerGroupName"   
 Value="[ConsumerGroupName]" />  
 <Parameter Name="EventProcessorHostService\_MaxRetryCount"   
 Value="3" />  
 <Parameter Name="EventProcessorHostService\_CheckpointCount"   
 Value="[CheckpointCount]" />  
 <Parameter Name="EventProcessorHostService\_BackoffDelay"   
 Value="1" />  
 </Parameters>  
 </Application>

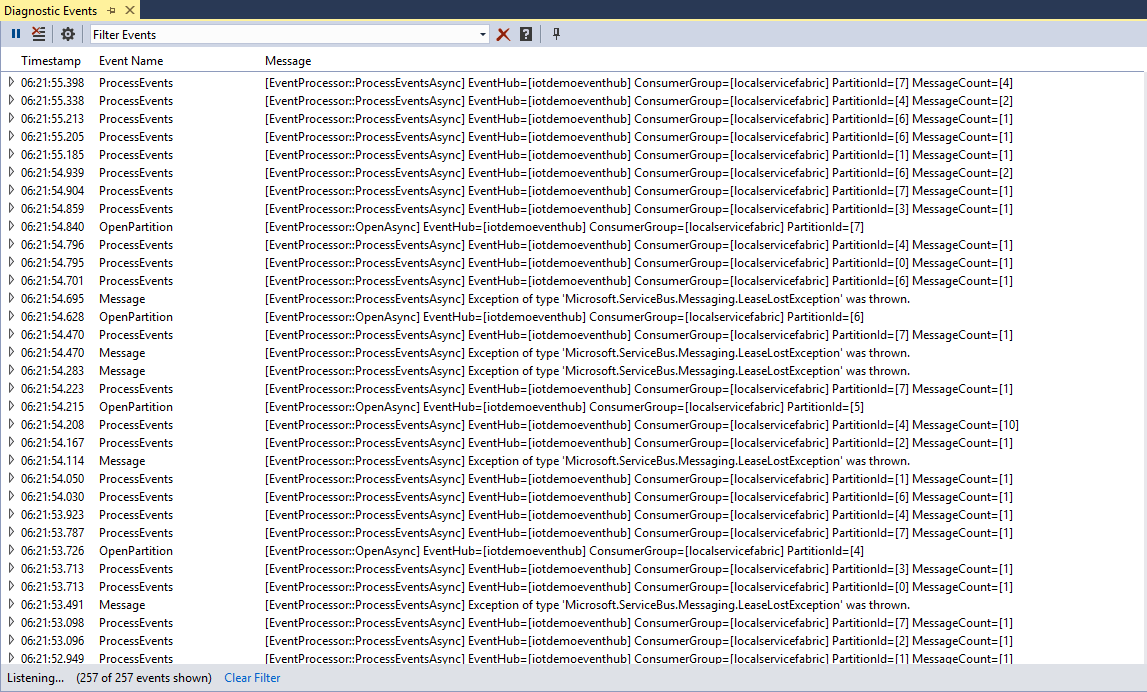
**ApplicationParameters.xml** file in the **PageViewTracer** project:

<?xml version="1.0" encoding="utf-8"?>  
 <Application xmlns:xsd="http://www.w3.org/2001/XMLSchema"   
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   
 Name="fabric:/PageViewTracer"   
 xmlns="http://schemas.microsoft.com/2011/01/fabric">  
 <Parameters>  
 <Parameter Name="EventProcessorHostService\_InstanceCount"   
 Value="1" />  
 <Parameter Name="EventProcessorHostService\_SqlDatabaseConnectionString"   
 Value="[SqlDatabaseConnectionString]" />  
 <Parameter Name="EventProcessorHostService\_InsertStoredProcedure"   
 Value="[InsertStoredProcedure]" />  
 <Parameter Name="EventProcessorHostService\_StorageAccountConnectionString"   
 Value="[StorageAccountConnectionString]" />  
 <Parameter Name="EventProcessorHostService\_ServiceBusConnectionString"   
 Value="[ServiceBusConnectionString]" />  
 <Parameter Name="EventProcessorHostService\_EventHubName"   
 Value="[EventHubName]" />  
 <Parameter Name="EventProcessorHostService\_ConsumerGroupName"   
 Value="[ConsumerGroupName]" />  
 <Parameter Name="EventProcessorHostService\_MaxRetryCount"   
 Value="3" />  
 <Parameter Name="EventProcessorHostService\_CheckpointCount"   
 Value="[CheckpointCount]" />  
 <Parameter Name="EventProcessorHostService\_BackoffDelay"   
 Value="1" />  
 </Parameters>  
 </Application>

**ApplicationManifest.xml** file in the **PageViewTracer** project:

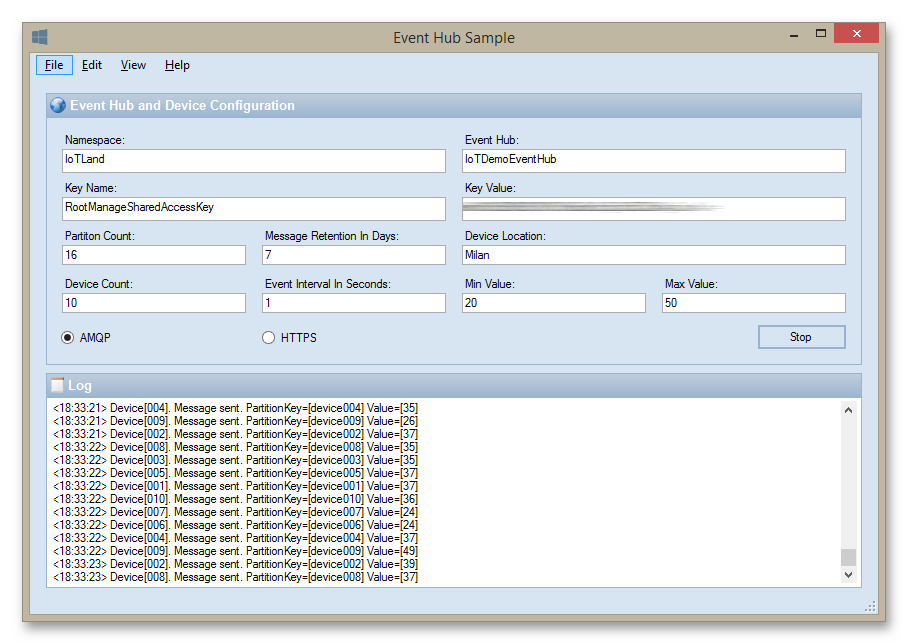
<?xml version="1.0" encoding="utf-8"?>  
 <ApplicationManifest xmlns:xsd="http://www.w3.org/2001/XMLSchema"   
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   
 ApplicationTypeName="PageViewTracerType"   
 ApplicationTypeVersion="1.0.1"   
 xmlns="http://schemas.microsoft.com/2011/01/fabric">  
 <Parameters>  
 <Parameter Name="EventProcessorHostService\_InstanceCount" DefaultValue="-1" />  
 <Parameter Name="EventProcessorHostService\_SqlDatabaseConnectionString" DefaultValue="" />  
 <Parameter Name="EventProcessorHostService\_InsertStoredProcedure" DefaultValue="sp\_InsertJsonEvents" />  
 <Parameter Name="EventProcessorHostService\_StorageAccountConnectionString" DefaultValue="" />  
 <Parameter Name="EventProcessorHostService\_ServiceBusConnectionString" DefaultValue="" />  
 <Parameter Name="EventProcessorHostService\_EventHubName" DefaultValue="" />  
 <Parameter Name="EventProcessorHostService\_ConsumerGroupName" DefaultValue="" />  
 <Parameter Name="EventProcessorHostService\_ContainerName" DefaultValue="usersessions" />  
 <Parameter Name="EventProcessorHostService\_QueueName" DefaultValue="usersessions" />  
 <Parameter Name="EventProcessorHostService\_MaxRetryCount" DefaultValue="3" />  
 <Parameter Name="EventProcessorHostService\_CheckpointCount" DefaultValue="100" />  
 <Parameter Name="EventProcessorHostService\_BackoffDelay" DefaultValue="1" />  
 </Parameters>  
 <ServiceManifestImport>  
 <ServiceManifestRef ServiceManifestName="EventProcessorHostServicePkg"   
 ServiceManifestVersion="1.0.0" />  
 <ConfigOverrides>  
 <ConfigOverride Name="Config">  
 <Settings>  
 <Section Name="EventProcessorHostConfig">  
 <Parameter Name="SqlDatabaseConnectionString"   
 Value="[EventProcessorHostService\_SqlDatabaseConnectionString]" />  
 <Parameter Name="InsertStoredProcedure"   
 Value="[EventProcessorHostService\_InsertStoredProcedure]" />  
 <Parameter Name="StorageAccountConnectionString"   
 Value="[EventProcessorHostService\_StorageAccountConnectionString]" />  
 <Parameter Name="ServiceBusConnectionString"   
 Value="[EventProcessorHostService\_ServiceBusConnectionString]" />  
 <Parameter Name="EventHubName"   
 Value="[EventProcessorHostService\_EventHubName]" />  
 <Parameter Name="ConsumerGroupName"   
 Value="[EventProcessorHostService\_ConsumerGroupName]" />  
 <Parameter Name="CheckpointCount"   
 Value="[EventProcessorHostService\_CheckpointCount]" />  
 <Parameter Name="MaxRetryCount"   
 Value="[EventProcessorHostService\_MaxRetryCount]" />  
 <Parameter Name="BackoffDelay"   
 Value="[EventProcessorHostService\_BackoffDelay]" />  
 </Section>  
 </Settings>  
 </ConfigOverride>  
 </ConfigOverrides>  
 </ServiceManifestImport>  
 <DefaultServices>  
 <Service Name="EventProcessorHostService">  
 <StatelessService ServiceTypeName="EventProcessorHostServiceType"   
 InstanceCount="[EventProcessorHostService\_InstanceCount]">  
 <SingletonPartition />  
 </StatelessService>  
 </Service>  
 </DefaultServices>  
 </ApplicationManifest>

The **EventProcessor** class uses the **ServiceEventSource** class to generate ETW events. You can use Service Fabric tooling in Visual Studio to see streaming traces, as shown in the picture below.



## Device Simulator

This application can be used to provision the **Event Hub** and simulate a configurable amount of devices.



The following table shows the configuration file of the application. Make sure to substitute the placeholders with the expected information before running the application.

<?xml version="1.0" encoding="utf-8"?>  
 <configuration>  
 <appSettings>  
 <add key="namespace" value="[SERVICE BUS NAMESPACE]"/>  
 <add key="keyName" value="[NAMESPACE LEVEL SAS KEY NAME]"/>  
 <add key="keyValue" value="[NAMESPACE LEVEL SAS KEY VALUE]"/>  
 <add key="eventHub" value="[EVENT HUB NAME]"/>  
 <add key="partitionCount" value="16"/>  
 <add key="retentionDays" value="7"/>  
 <add key="location" value="Milan"/>  
 <add key="deviceCount" value="10"/>  
 <add key="eventInterval" value="1"/>  
 <add key="minValue" value="20"/>  
 <add key="maxValue" value="50"/>  
 </appSettings>  
 ...  
 </configuration>