Bau dir deinen eigenen Serverless EventStore

Azure Functions + Custom Bindings

Jan(ek) Fellien | @janekf | Inclusify AG

Part 1 - Was ist Event Sourcing

Kurze Einleitung, was EventSourcing eigentlich ist
Alle Teilnehmenden nennen aus ihrer jeweiligen Domain ein zwei Events

HISTORICAL

Event Store

Change

Append only

"Customer Created"

STANDARD

Part 2 - Bauen wir uns einen serverless EventStore

Systemvoraussetzungen

- Visual Studio 2017 oder 2019 mit Azure Cloud Erweiterung
- oder Visual Studio Code mit Azure Functions Erweiterung
- oder JetBrains Rider >= 2018.1
- Node.Js 10.x
- > npm install azure-functions-core-tools
- dotNet Core 2.2

Part 2 - Bauen wir uns einen serverless EventStore

Vorbereitung

Gehe zu einem Ordner, wo Du deine Azure Function App entwickeln möchtest
Lege dir dort einen Ordner an

```
> cd [YOUR FUNCTION PATH HERE]
> npm install -g azure-functions-core-tools
> func init
```

Wähle dotnet als Host Environment

Part 2 - Bauen wir uns einen serverless EventStore

Installieren der Extension für den EventStore

Die Erweiterung wir über einen NugetFeed mit Hilfe des Befehls `func` in der Konsole installiert

```
> func extensions install
  --package SiaConsulting.Azure.WebJobs.Extensions.EventStoreExtension.Streams
  --version 0.1.2
```

Part 2 - Bauen wir uns einen serverless EventStore

Neue Funktion anlegen

In der Konsole lässt sich, wieder über Funktion `func`, eine neue Funktion anlegen

```
> func new
```

Wähle nun `HttpTrigger` aus und benenne die Funktion `StoreEvent`
Wir lassen die Funktion erstmal, wie sie ist

Part 2 - Bauen wir uns einen serverless EventStore

Einrichten des EventStore Servers

Mit Hilfe von docker ist es am einfachsten

```
> docker pull eventstore/eventstore
> docker run --name eventstore-node -it
  -p 2113:2113 -p 1113:1113 eventstore/eventstore
```

Falls der Container nicht gestartete ist, dann bitte mit
`docker start eventstore-node` starten

Part 2 - Bauen wir uns einen serverless EventStore

Einrichten des EventStore Servers

Die beiden Ports bedeuten, dass die Kommunikation auf zwei Wegen hergestellt wird

Auf dem Port **2113** wird mit dem Webportal kommuniziert und auf dem Port **1113** werden die Daten ausgetauscht

Öffnen des Webportals erfolgt über `http://127.0.0.1:2113`. Der Standardlogin lautet `admin:changeit`

Part 2 - Bauen wir uns einen serverless EventStore

Laden der Setup Werte

Ich habe für den Workshop eine Function App eingerichtet, mit der ich Daten bzgl. Connections und ähnliches bereitstelle

Unter
https://its-gone-so-far-:)

kann jeder Teilnehmer diese Daten direkt im Browser anfragen

Part 2 - Bauen wir uns einen serverless EventStore

Konfigurieren der Function App

In der `local.settings.json` Datei kann man für die lokale Entwicklung die Werte ablegen, die später in der Function App unter den Settings eingerichtet werden

Nimm die Werte aus der o.g. API und trage sie in Deine local.settings ein

Part 2 - Bauen wir uns einen serverless EventStore

Funktion mit vernünftigem Code befüllen

1.) ein neuer Parameter im Funktionsaufruf, für das OUT Binding

```
[EventStoreStreams(ConnectionStringSetting="EventStoreConnection")]
IAsyncCollector<EventStoreData> data
```

`IAsyncCollector<T>` nimmt die Daten zum Speichern entgegen
`EventStoreData` ist der Container, der ein oder mehrere Events enthalten kann

Part 2 - Bauen wir uns einen serverless EventStore

Funktion mit vernünftigem Code befüllen

2.) ein bisschen Meta

```
byte[] metaData = null;
var eventID = Guid.NewGuid();
```

`metaData` ist etwas Spezielles für den EventStore und wird aktuell nicht benötigt
`eventID` ist die eindeutige ID eines Events

Part 2 - Bauen wir uns einen serverless EventStore

Funktion mit vernünftigem Code befüllen

3.) noch mehr Meta

```
var eventStream          = req.Headers["x-related-event-stream"].ToString();
var eventType            = req.Headers["x-related-event-type"].ToString();
var isJson               = req.ContentType == "application/json";
var requestBodyAsString = await req.Body.AsString();
var requestBodyAsBytes   = requestBodyAsString.AsBytes();
```

`eventStream` ist der Name des Streams, zu dem der Event gehört --> siehe Part 1
`eventType` hat zwei Zwecke; zum Einen, für die Deserialisierung und zum Anderen beschreibt der Type den Grund, warum es den Event gibt --> siehe Part 1
`isJson` wird vom EventStore benötigt
`requestBodyAsString` der gesendete Body als String
`requestBodyAsBytes` der EventStore speichert nur Bytes, darum wird der gesendete Body in Bytes umgewandelt

Part 2 - Bauen wir uns einen serverless EventStore

Funktion mit vernünftigem Code befüllen

4.) den gesendeten Event in einen EventStoreEvent umwandeln

```
var eventData
    = new EventData(eventID, eventType, isJson, requestBodyAsBytes, metaData);
```

`eventData` stellt einen einzelnen Event für den EventStore dar

Part 2 - Bauen wir uns einen serverless EventStore

Funktion mit vernünftigem Code befüllen

5.) Daten an das Binding übergeben

```
await data.AddAsync(new EventStoreData{
       StreamName = eventStream,
       Events = new List<EventData>{ eventData }
});
```

`EventStoreData` stellt eine Collection von EventStore Events dar

Done so far?

Part 2 - Bauen wir uns einen serverless EventStore

Lesen von Events

Anlegen einer zweiten Funktion

```
> func new
```

Wir wählen `HttpTrigger` aus

Part 2 - Bauen wir uns einen serverless EventStore

Befüllen der neuen Funktion mit vernünftigem Code

Anlegen einer zweiten Funktion

```
> func new
```

Wir wählen HttpTrigger aus

Part 2 - Bauen wir uns einen serverless EventStore

Funktion mit vernünftigem Code befüllen

1.) ein neuer Parameter im Funktionsaufruf, für das IN Binding

```
[HttpTrigger(AuthorizationLevel.Function, "get",
Route = "retrieve/{eventStream}")]
HttpRequest req,
[EventStoreStreams(ConnectionStringSetting="EventStoreConnection",
StreamName="{eventStream}", StreamReadDirection=StreamReadDirection.Forward)]
IList<ResolvedEvent> data
```

`{eventStream}` gilt als Platzhalter in unserer Route
`data` enthält alle Events vom angegebenen `{eventStream}`

Part 2 - Bauen wir uns einen serverless EventStore

Funktion mit vernünftigem Code befüllen

2.) Events aus dem Stream deserialisieren

```
var events = data.Select( e => {
       var eventData = JsonConvert.DeserializeObject<dynamic>(
              Encoding.UTF8.GetString(e.Event.Data));
       eventData.EventType = e.Event.EventType;
       return eventData;
}).ToList();
```

`{eventStream}` gilt als Platzhalter in unserer Route
`data` enthält alle Events vom angegebenen `{eventStream}`

Bau dir
dein eigenes
Custom Binding

Jan(ek) Fellien | @janekf | Inclusify AG

Part 3 – My own Custom Binding

“Exakt” drei Schritte zum Glück

1.) Eine Klasse, die `IAsyncCollector<T>` implementiert
2.) Ein Attribut markiert als `[Binding]`
3.) Eine Klasse, die `IExtensionConfigProvider` implementiert

Hidden Secret
0.) Ein dotnet Core 2.2 C# Projekt mit dem neuesten Microsoft.Azure.Functions.Extensions NuGet Package
4.) Eine Startup Klasse, die `IWebJobsStartup` implementiert
5.) Eine AssemblyInfo.cs

Part 3 – My own Custom Binding

IAsyncCollector<T>

Nimmt die Werte der Function entgegen und verarbeitet sie einzeln

```
internal class MyOwnCollector : IAsyncCollector<string> {
   public async Task AddAsync(string myValue, CancellationToken …){…}
   public Task FlushAsync(CancellationToken …) {…}
}
```

`AddAsync` verarbeitet jeden einzelnen Wert aus der IAsyncCollector Liste

Part 3 – My own Custom Binding

Attribut mit Binding

Das Attribut markiert einen Parameter der Funktion der für das Binding verantwortlich ist.

```
[Binding]
[AttributeUsage(AttributeTargets.Parameter | AttributeTargets.ReturnValue)]
public class MyBindingAttribute : Attribute {

   [AutoResolve]
   public string MyAttributeValue{ get; set;}
}
```

`AutoResolve` erlaubt das Verwenden von Platzhaltern im Trigger

Part 3 – My own Custom Binding

IExtensionConfigProvider

Hier werden Attribut und Collector zusammengefügt

```
[Extension(“MySuperDuperExtension")]
public class MySuperDuperExtension : IExtensionConfigProvider {

   public void Initialize(ExtensionConfigContext context) {…}
}
```

`Extension` markiert die Klasse als die Definition der entsprechenden Extension aka dem Custom Binding

Link zum GitHub Repo

https://github.com/jfellien/serverless-eventstore-hands-on