Sensor nettverk

# Utstyr som trengs

* MQ-7
* MH-Z19
* DHT11
* Photoresistor
* PIR
* Microsoft Account
* Microsoft Azure
* Visual Studio 2015
* Particle Local IDE

Før du i det heletatt begynner å koble opp hele systemet og laste opp kode, kan det lønne seg å bare koble hvert komponent hver for seg og teste hver for seg.

# Kobling

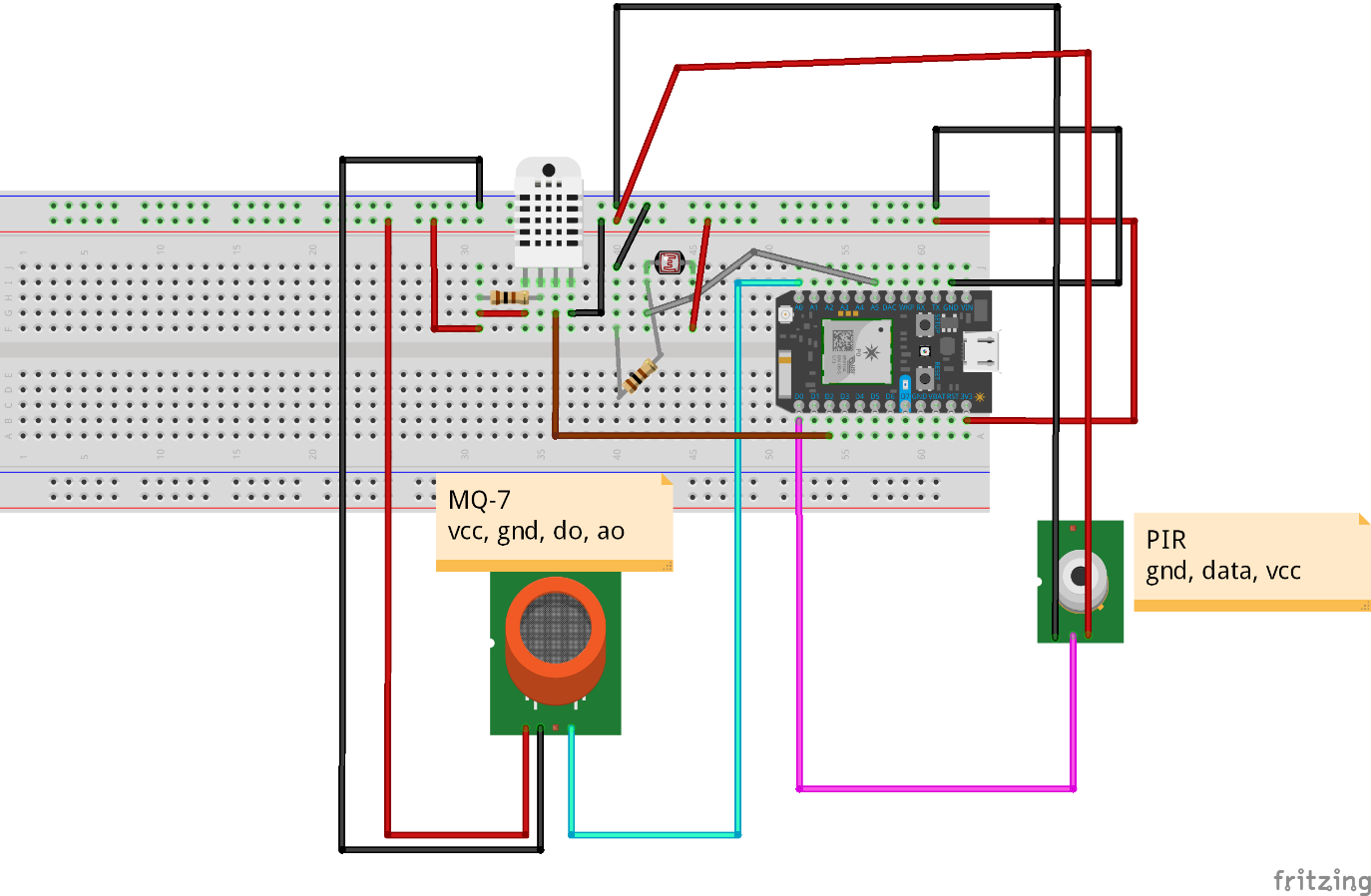
|  |  |
| --- | --- |
| PIR | Particle Photon |
| VCC | VCC |
| Data | D0 |
| GND | GND |

|  |  |
| --- | --- |
| MQ-7 | Particle Photon |
| VCC | VCC |
| GND | GND |
| D-OUT | ingen |
| A-OUT | A0 |

|  |  |
| --- | --- |
| DHT11 | Particle Photon |
| VCC | VCC |
| GND | GND |
| D-OUT | D2 |
| A-OUT | Ingen |

|  |  |
| --- | --- |
| Photosensor | Particle Photon |
| A-OUT | A5 |
| GND | GND |

|  |  |
| --- | --- |
| MH-Z19 | Particle Photon |
| PWM | D1 |
| GND | GND |
| VCC | VCC |



# Particle Photon

Til dette prosjektet lastet jeg ned local IDE for å kunne jobbe med filer. Fordelen med dette er når man har så mange sensorer som nå, blir det lettere å lage objekter av sensorene. IDE kan lastes ned her: <https://www.particle.io/products/development-tools/particle-local-ide>

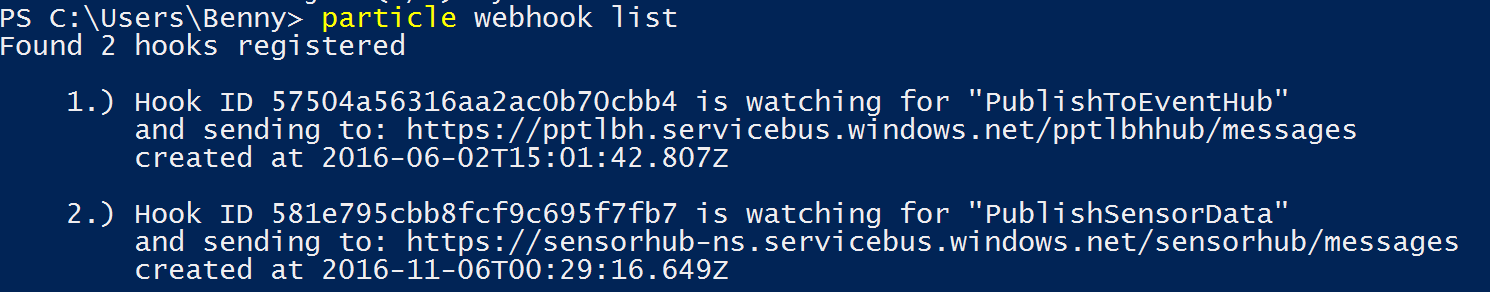
Ulempen med IDE-et er at man ikke kan inkludere biblioteker på like lett måte som online IDE-et. Dette må du selv laste ned og legge til i prosjektet. En løsning på dette (som jeg gjorde) var å gå inn på online IDE, finne det biblioteket jeg ville ha og replikere filene i local IDE.

Deretter, for at alle .h filer skal fungere må du inkludere #include "application.h" i respektive .filer. Dette er for å kunne bruke Particle sitt bibliotek.



Dette er JSON payload som blir sendt til eventhuben i Azure. For å generere denne bruker vi «particle webhook» i CLI.

Ved å skrive **particle webhook list** kan du se liste over dine webhooks.



Når du skal legge til/ endre på webhooken så må du først slette den (hvis den allerede eksisterer) ved å skrive **particle webhook delete <ID>** etterfulgt med **particle webhook create .\webhook.json**

# Azure

I azure bruker vi disse:

|  |
| --- |
| Apper |
| Service Bus |
| Stream Analytics |
| SQL Databases |
| WebApps |

Service bus er ikke støttet i den nye portalen enda. Derfor forholder vi oss mest til den gamle portalen: [www.manage.windowsazure.com](http://www.manage.windowsazure.com)

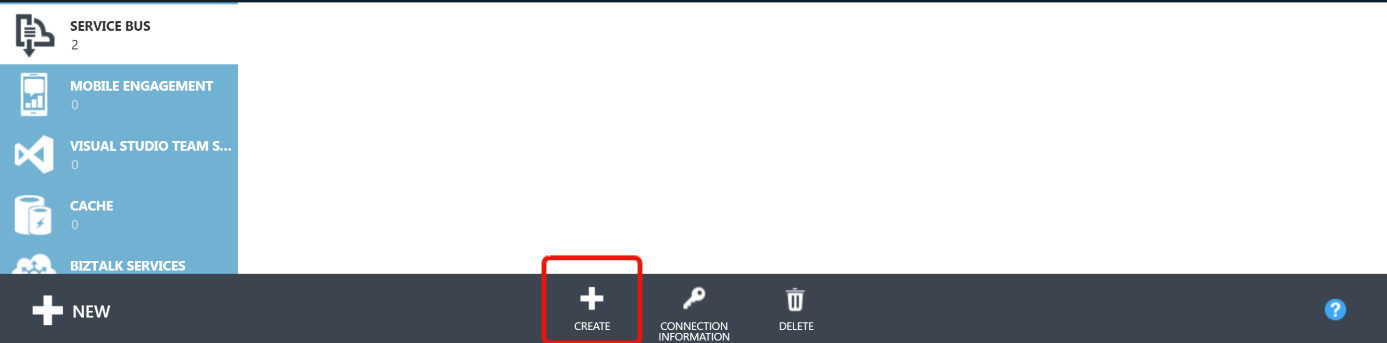
## Service Bus

Generell info om service bus: <https://azure.microsoft.com/nb-no/services/service-bus/>

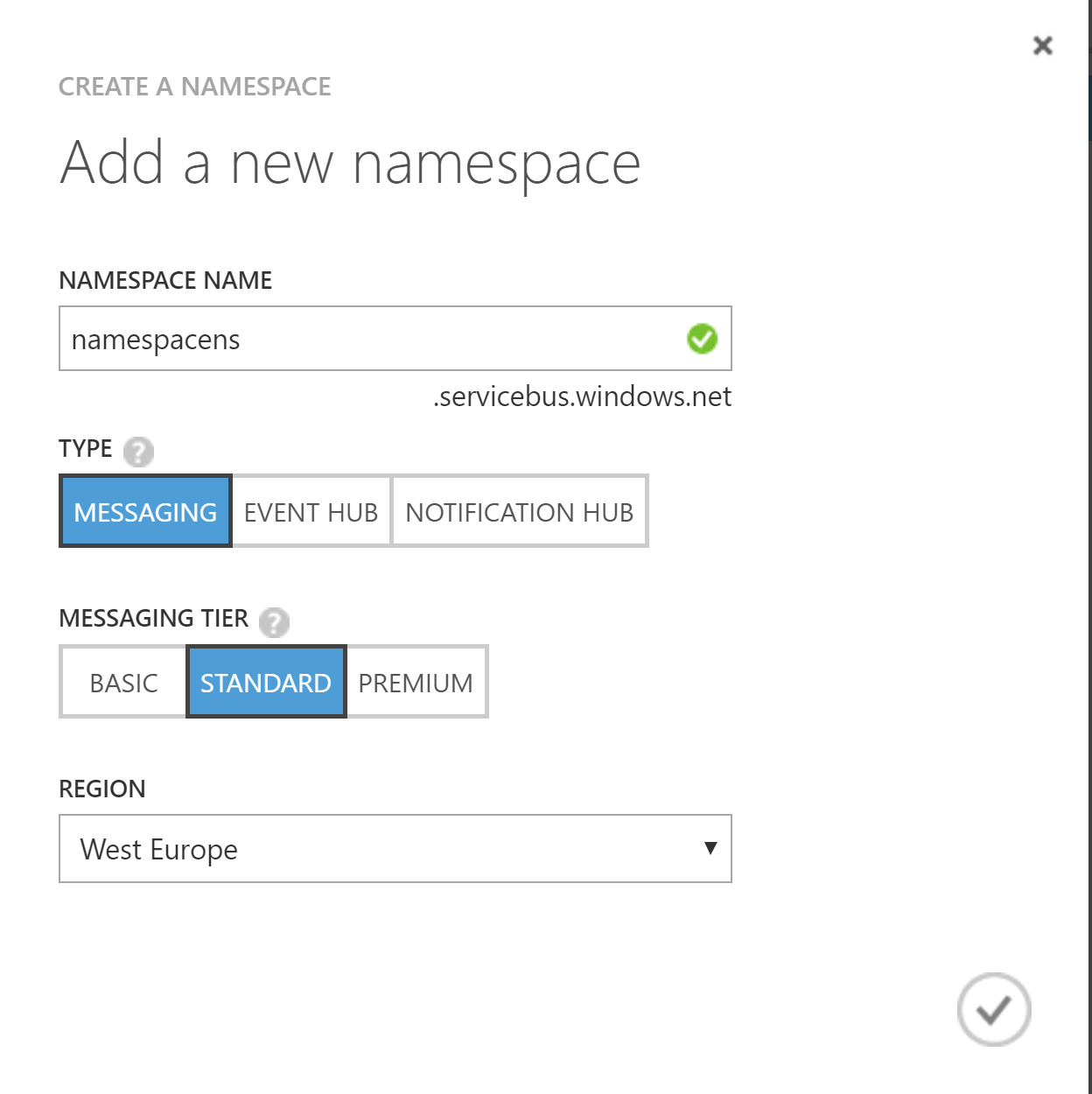
Fra Particle sender vi meldinger til Azure ved hjelp av Eventhubs som ligger i service bus. For å opprette en service bus må du klikke på dette ikonet. Dette finner du i sidebaren.



Klikk deretter på Create som befinner seg i footeren



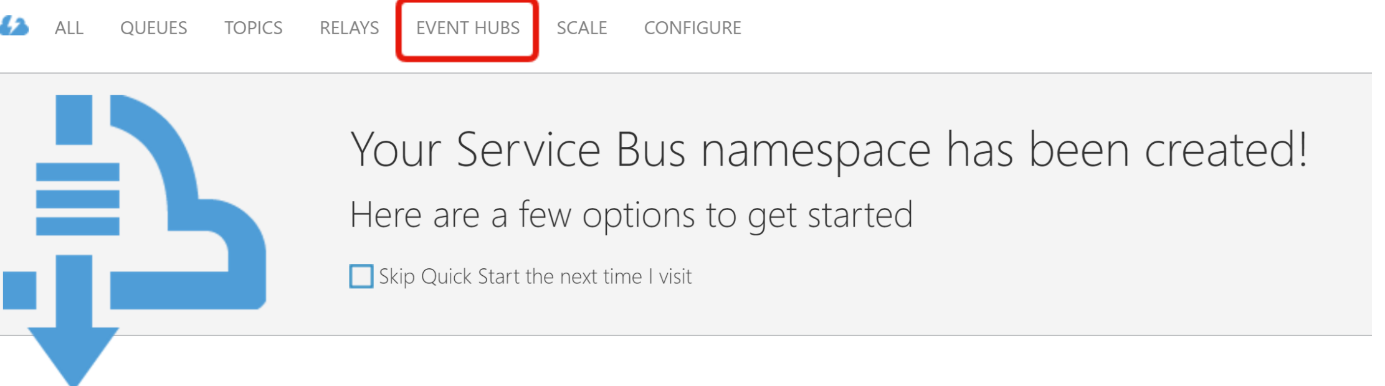
Deretter vil du få en dialogboks hvor du configurerer service bus. Allerede her er det viktig å ha logiske navnkonvensjoner. Det er helt opp til deg hvordan du gjør det. Jeg har lagt til prefikser på slutten av hvert navn. Eks: ns, hub, job.



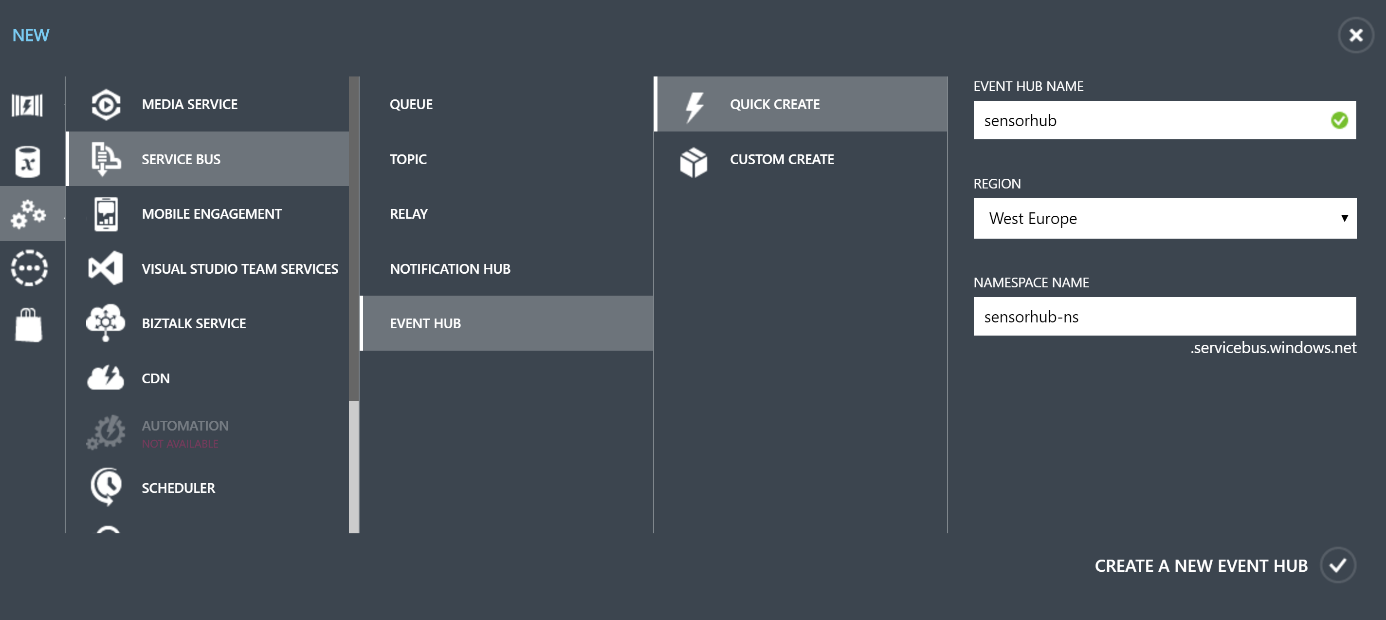
Velg instillingene som er vist over. Det vil ta noen minutter før Service bus er klar for bruk.

Når Azure er ferdig med å fyre opp Service Bus vil du få beskjed om det i notification baren nederst.

Inne på Service Busen din kan du gå inn i Event Hub som ligger i øverste navigasjonsmeny.

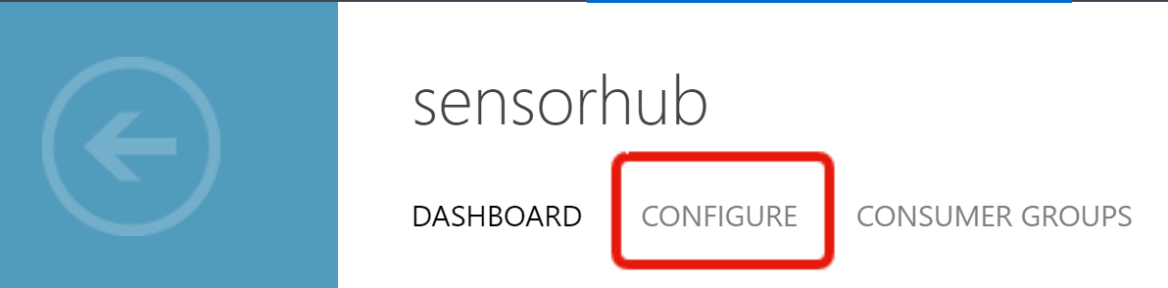


Trykk deretter på new nederst til venstre > quick create.

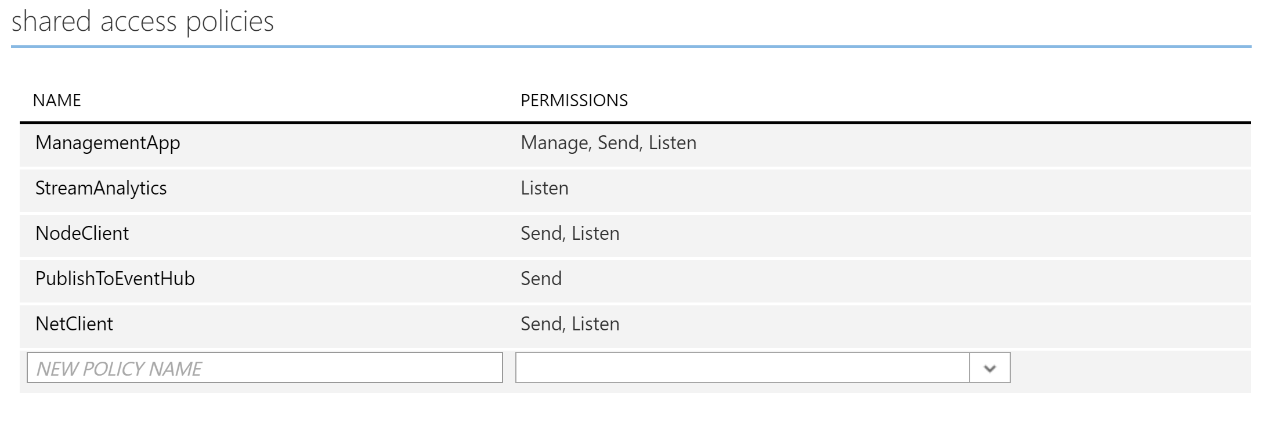


Kan hende at man kan lage eventhub direkte når man trykker på create i Service Bus, men for sikkerhetsskyld gjorde jeg dette for å få til en fungerende løsning. Denne prossessen var det jeg gjorde i forrige innlevering så jeg forholdt meg til det. Det kan være en minor update som har skjedd i mellom tiden. Men jeg ser allerede her at det blir opprettet to namespaces. Det kan være at siden forrige gang så må man ha med -ns på slutten av name-spacet. Dette har jeg enda ikke prøvd.

For å kunne få Particle Photon til å sende meldinger til Eventhuben må vi gjøre noen konfigureringer først. Naviger inn til configure for å sette instillinger på eventhuben.



Med tanke på sikkerhet kan man konfigurere shared access policy. Her kan vi bestemme hva slags rettigheter de ulike policiene kan ha. Dette er viktig med tanke på sikkerhet. Jeg har satt opp en del policies for fremtidlig bruk.



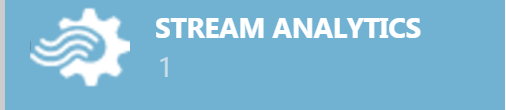
Det er også her du finner shared access key til particle webhook. Det er på denne måten webhooken kan klare å gjøre HTTP-POST request til Event hub.

## Stream Analytics

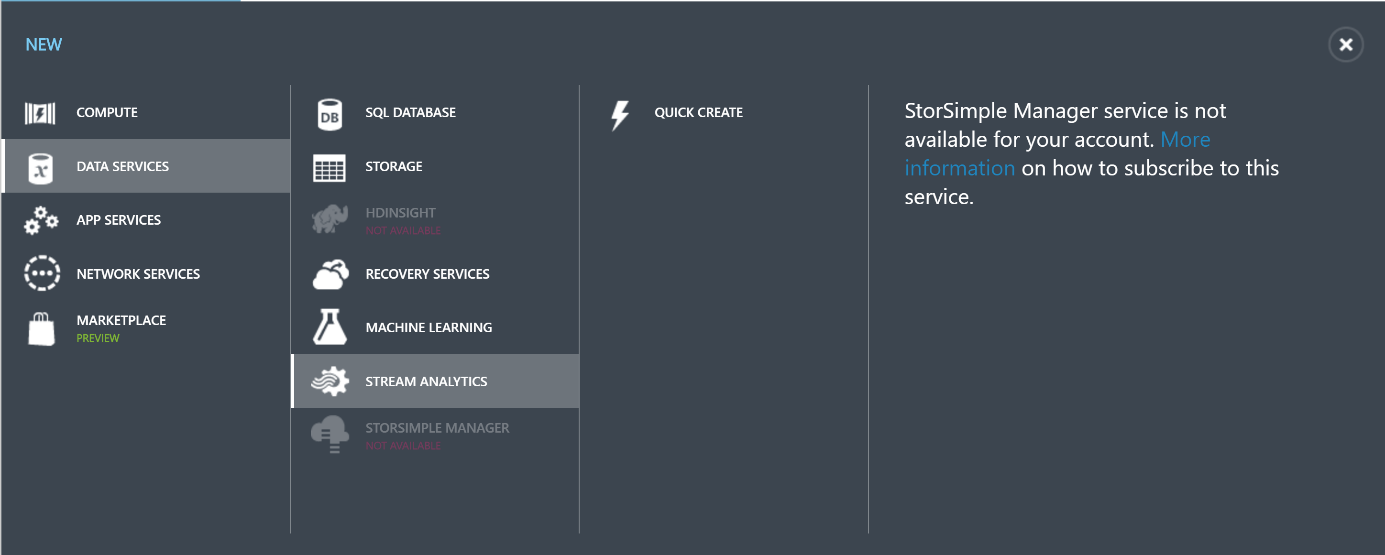
Generell info om Stream Analytics: <https://azure.microsoft.com/nb-no/services/stream-analytics/>

I vårt prosjekt vil Stream Analytics være den tjenesten som tar input fra Event hub og output til en SQL-Database.

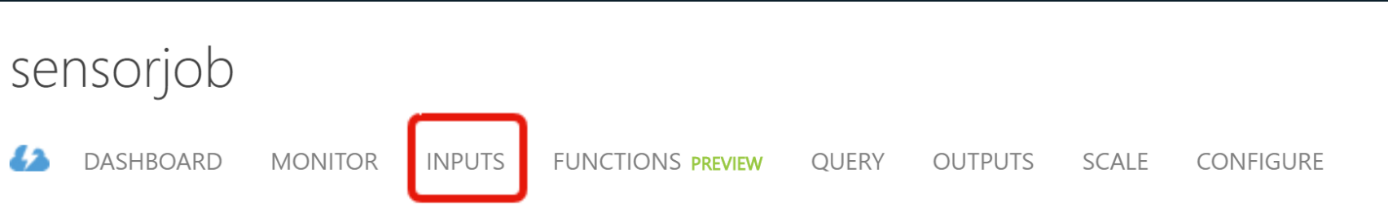
På samme måte som Service bus leter du etter en lignende ikon bare for stream analytics



Jo mer du bruker tjenesten jo mer vant blir du til å opprette ting da det er mye av det samme.



Vi vil først gå inn på inputs for å si til stream analytics hvor datakildene våres kommer i fra. Dette finner du i topmenyen

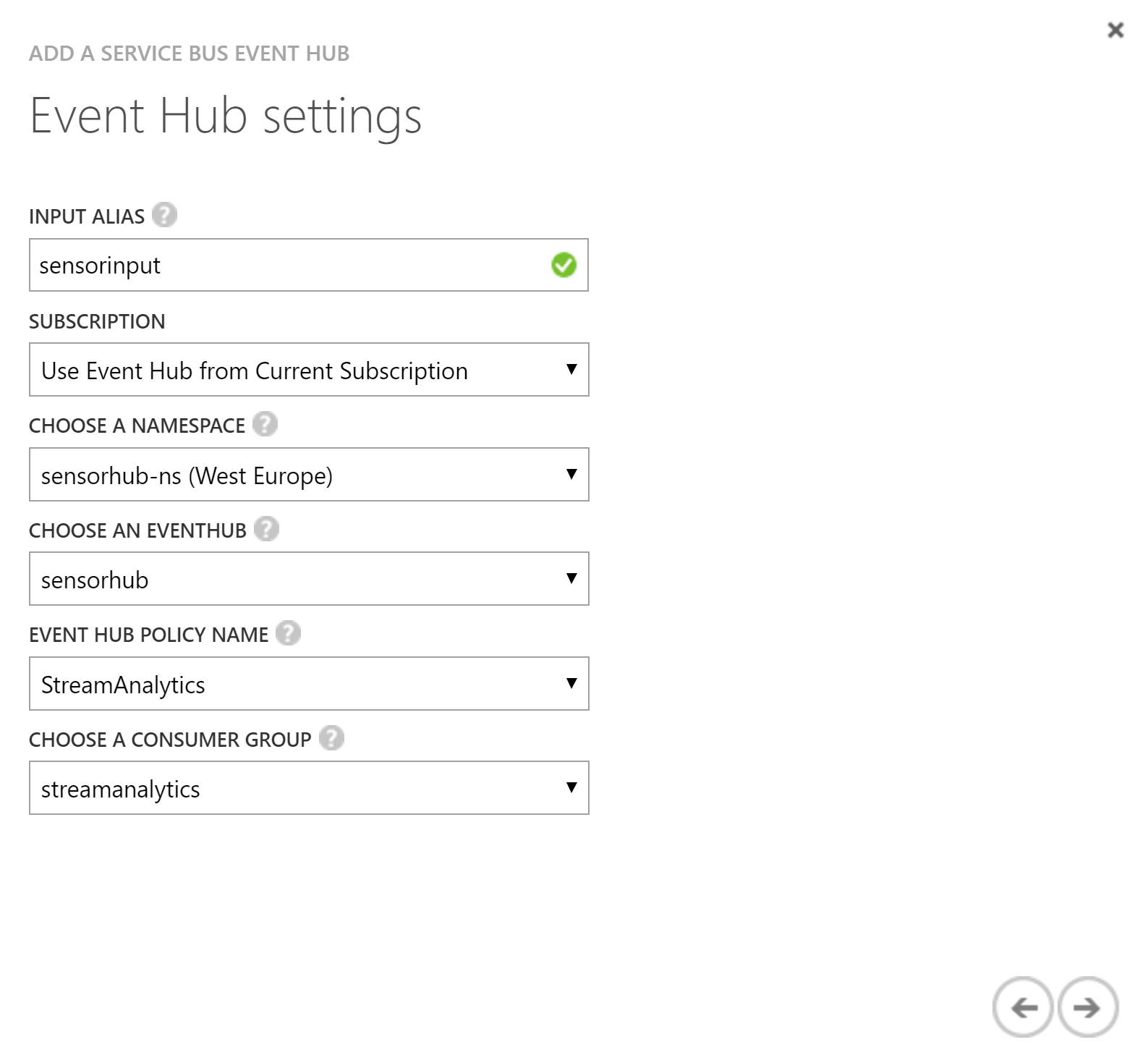


Trykk deretter på add input som befinner seg i footeren. 

## Input

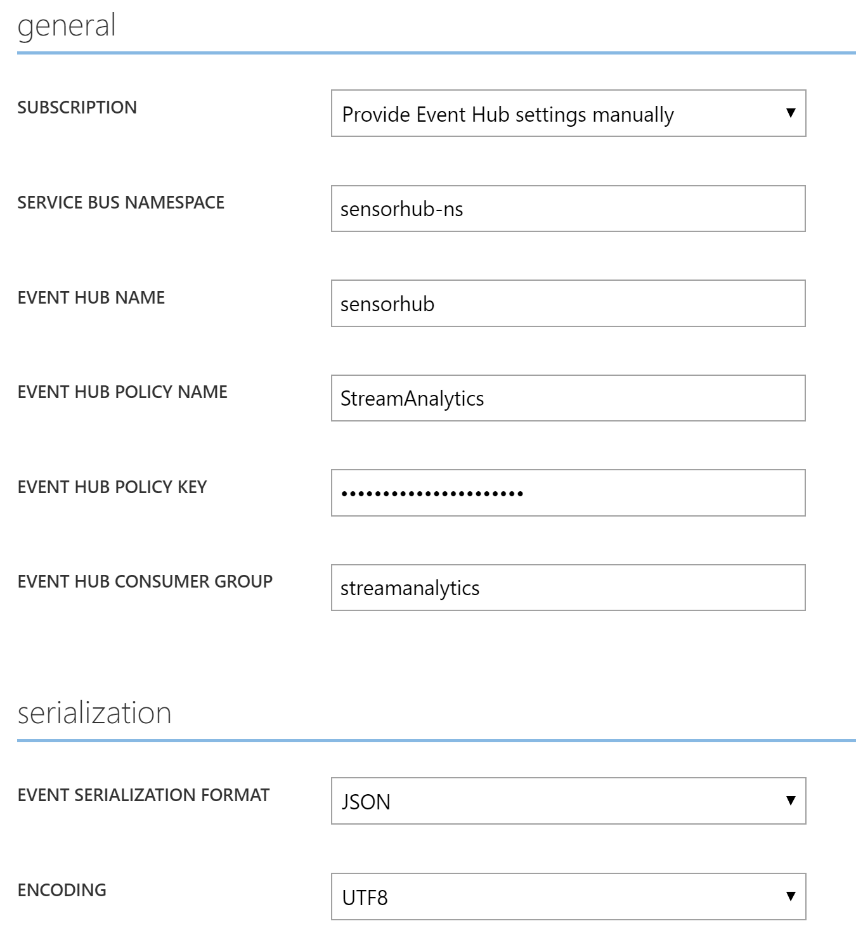
Step 1. Velg data stream

Step 2. Velg Event Hub

Step 3. 

Her har vi allerede konfigurert det som trengs i eventhuben. Der ser man også fordelen med å ha riktig navn allerede fra starten av. Hadde man gått for noe som test-1 test-2 test-3... hadde vi allerede sett her at det blir vanskelig.

Instillingene i input ser slik ut:

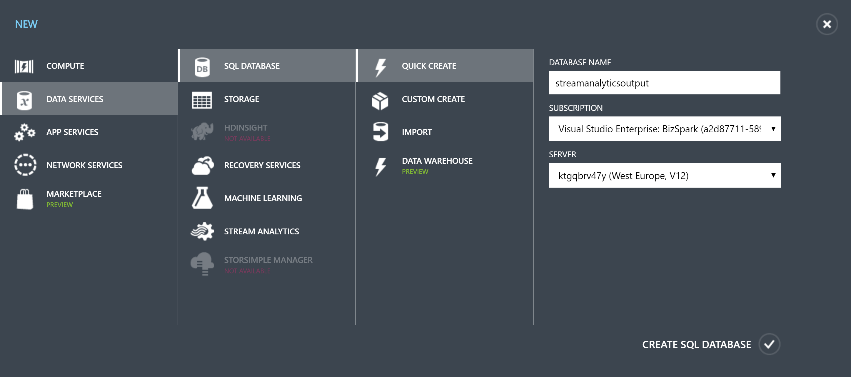


## Output

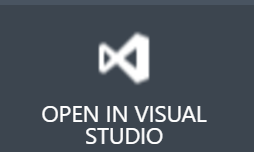
Før vi går videre i output må vi lage en database først.

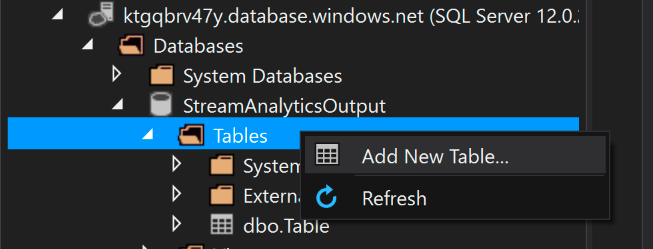


Naviger deg inn hit og trykk igjenn på new.



Når databasen er ferdig opprettet må vi legge til tabeller og attributter. For å gjøre dette lett, klikk inn på **Open in Visual Studio**



Let etter SQL serveren som databasen ligger i og opprett ny tabell. 

Det går ann å bruke Databasedesign vinduet til å legge til i databasen, men lim inn dette for å spare tid.

CREATE TABLE [dbo].[Table] (

[Id] INT IDENTITY (1, 1) NOT NULL,

[CO] INT NULL,

[temperature] FLOAT (53) NULL,

[humidity] FLOAT (53) NULL,

[light] INT NULL,

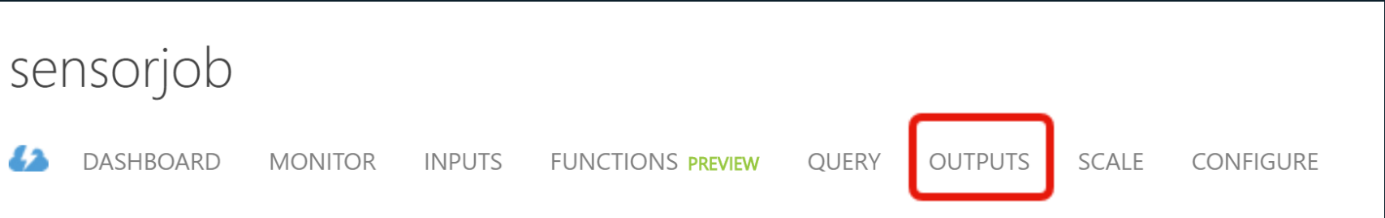
[motion] NVARCHAR (MAX) NULL,

[timecreated] DATETIME NULL,

[guid] NVARCHAR (MAX) NULL,

PRIMARY KEY CLUSTERED ([Id] ASC)

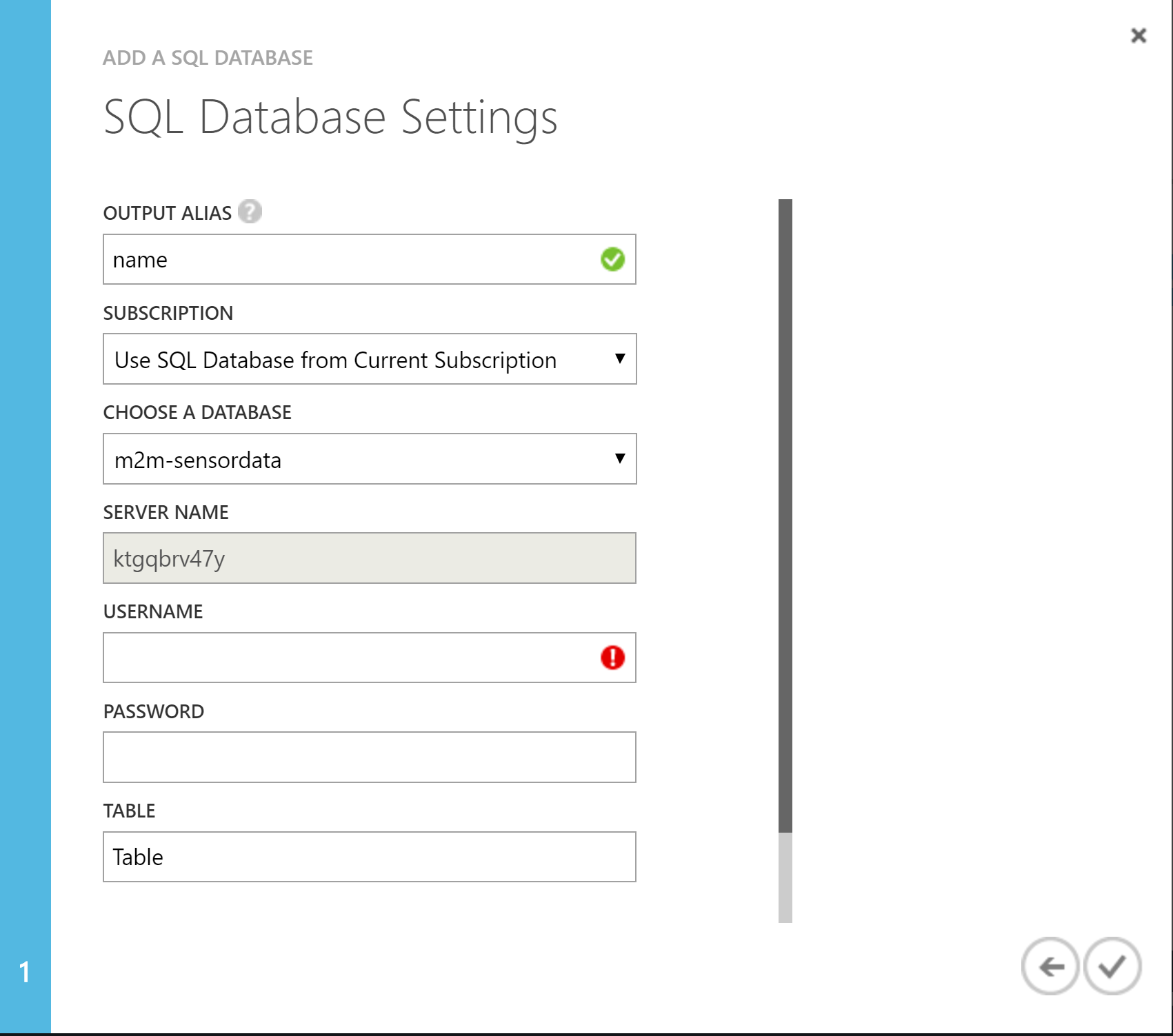
);

Nå kan vi endelig å videre til Output. For å lage Output navigerer vi til output i top menyen. 

Klikk på Add output og deretter velg SQL-Database



Fyll inn feltene med det du har configurert databasen din med. Her er det viktig at du referer til samme tabell. I mitt tilfelle kalte jeg bare min for Table.



Videre går vi til Query som er i samme meny som Input og Output. Her vil vi kunne kjøre spørringer slik at det som kommer fra input datastrømmen vil bli lagt inn i output som er vår database. Legg så inn dette scriptet i vinduet

**SELECT**

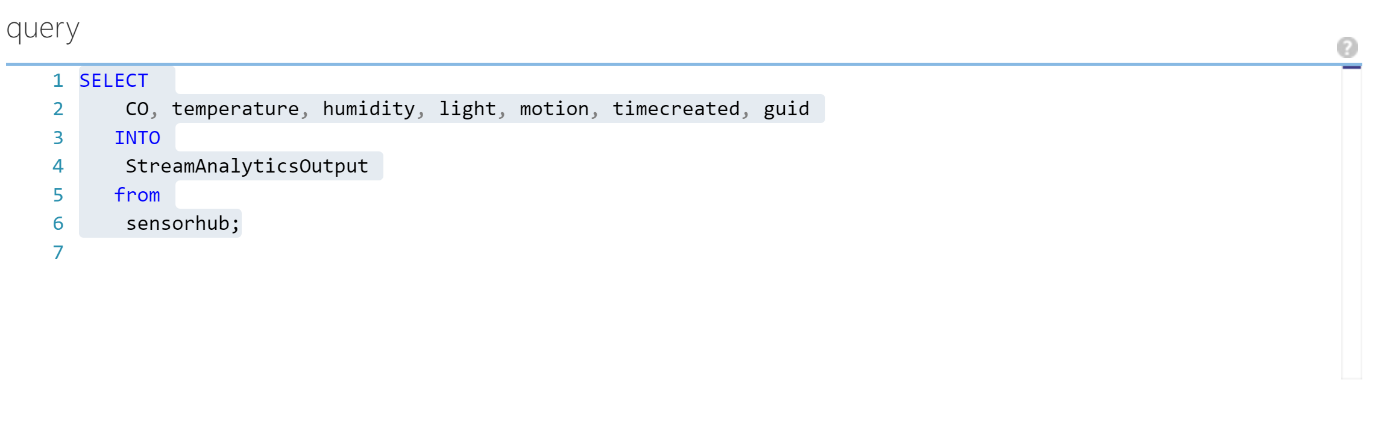
CO, temperature, humidity, light, motion, timecreated, guid

**INTO**

StreamAnalyticsOutput

**from**

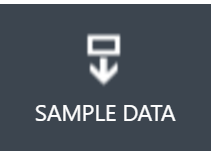
sensorhub;



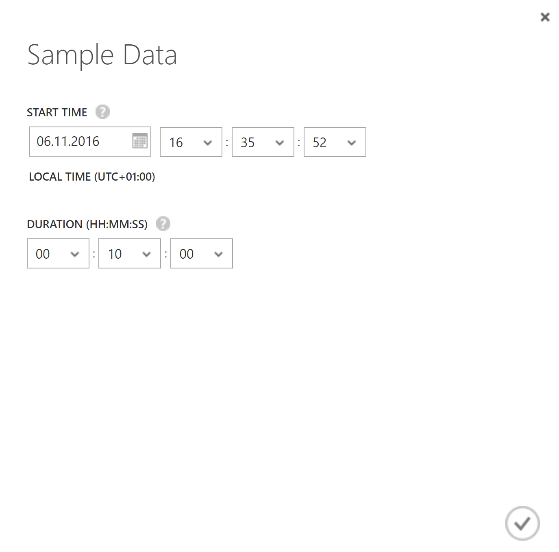
## Test runs

Før du spinner opp jobben kan det lønne seg å kjøre test på dataene. Dette kan spare deg for masse tid da oppstart av jobben tar lang tid og aggregeringen av dataene tar lang tid. Jeg brant meg veldig på dette under prosjektet da webhooken min hadde feil i JSON, men skjønte ikke hva som var problemet til jeg testet dataene.

Gå først inn på input for å laste ned sample data.



Her kan du velge hva slags intervall dataene dine skal komme fra. Hvis det var en tidsperiode der din particle ikke fungerte, kan du lett filtrere bort dette.

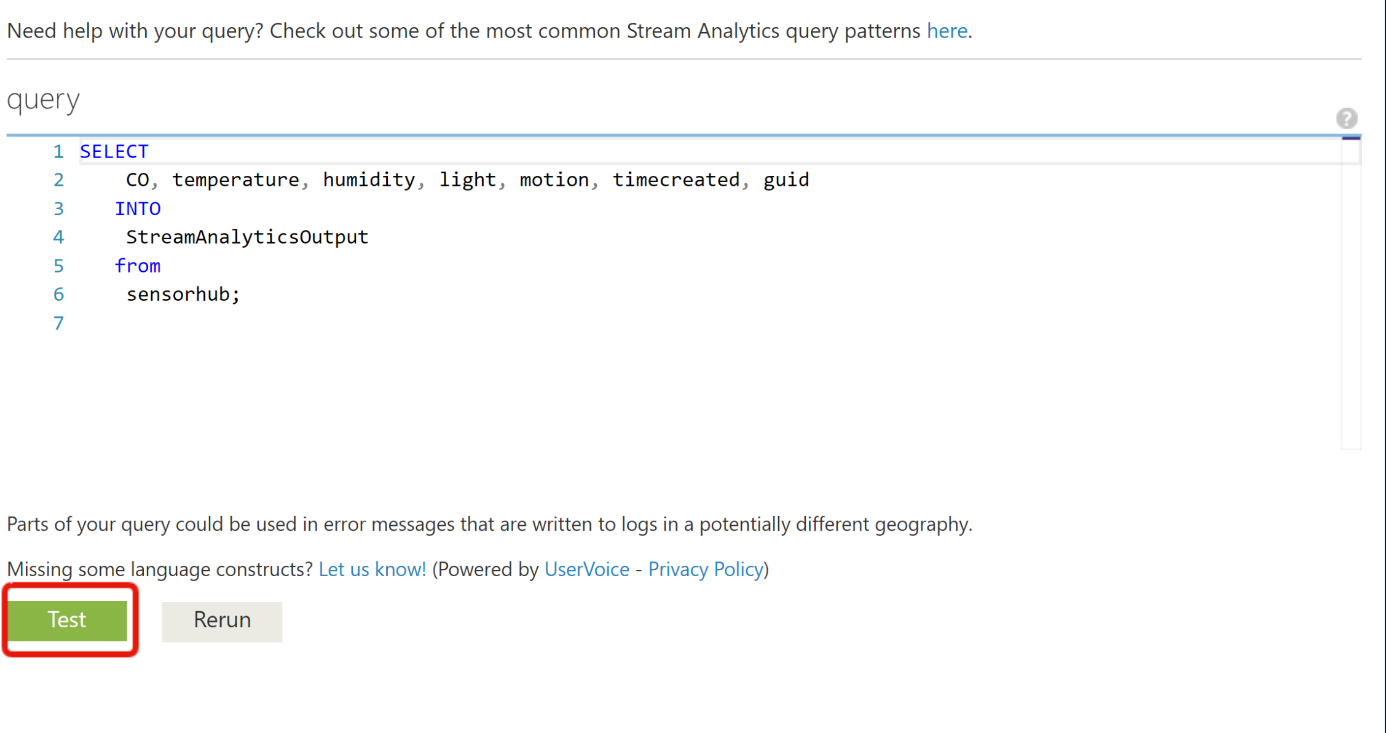


Når den er ferdig lastet trykk på details for å laste ned dataene.

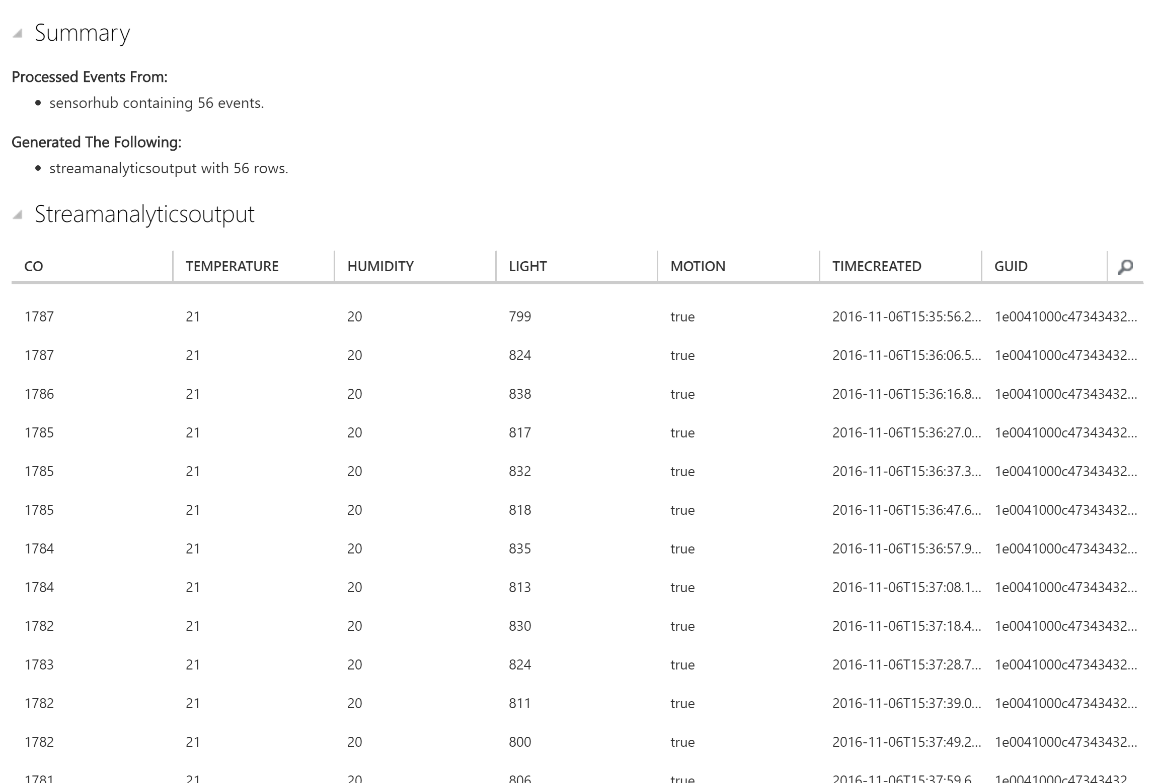


Bruk: <https://jsonformatter.curiousconcept.com> til å formatere JSON slik at det blir lettere å se dataene. Dette gjør du ved å åpne JSON filen, klipper ut innholdet og limer den inn i vinduet. Hvis alt her viser seg å være fint, betyr det at alle dataene som blir sendt fra din particle photon er safe og det er ikke noe feil der som må rettes opp.

Naviger deg videre inn til query og trykk på Test knappen.



Last opp JSON filen du fikk fra input. Da vil du også kunne få ut hvordan det vil se ut i databasen.

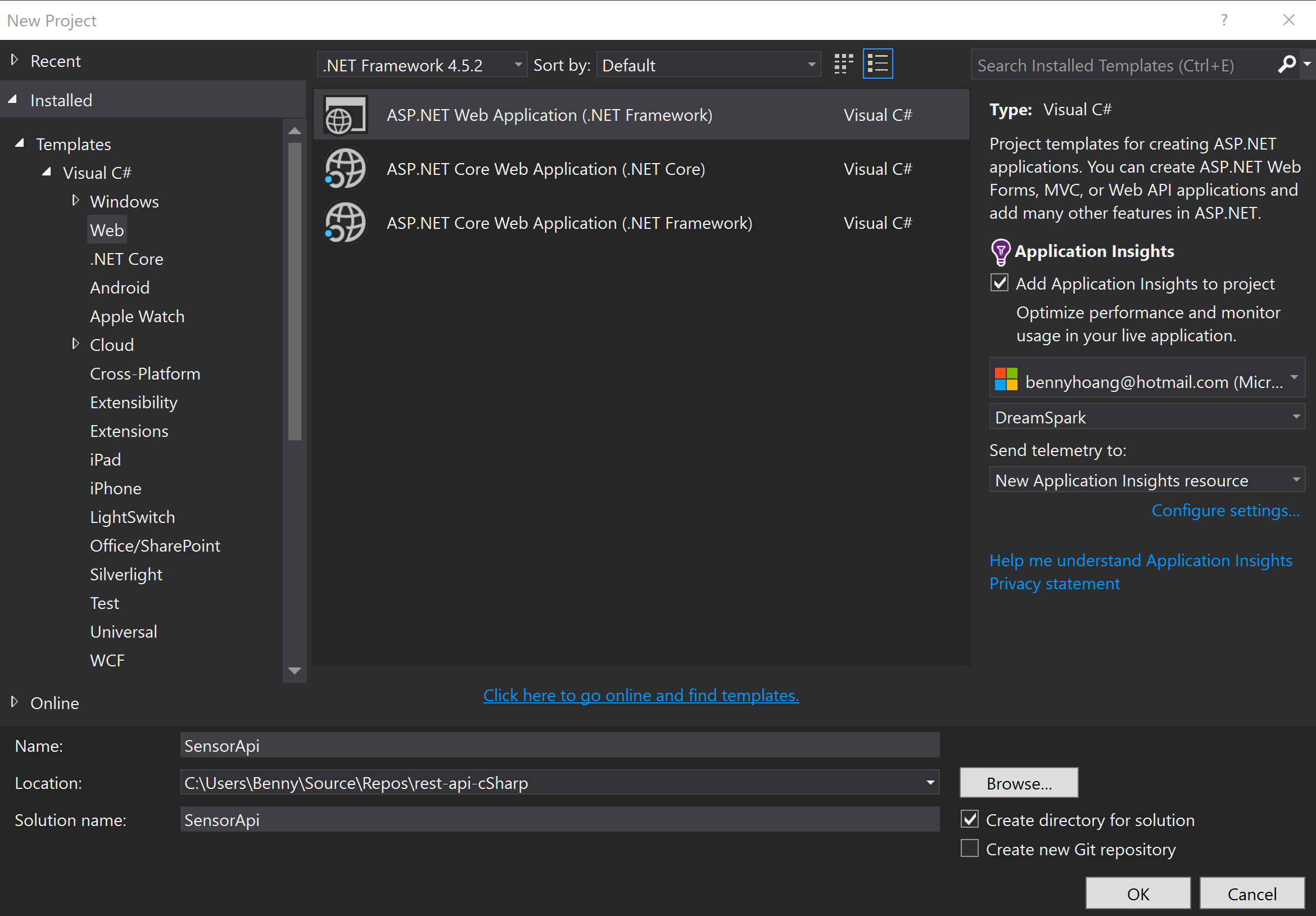


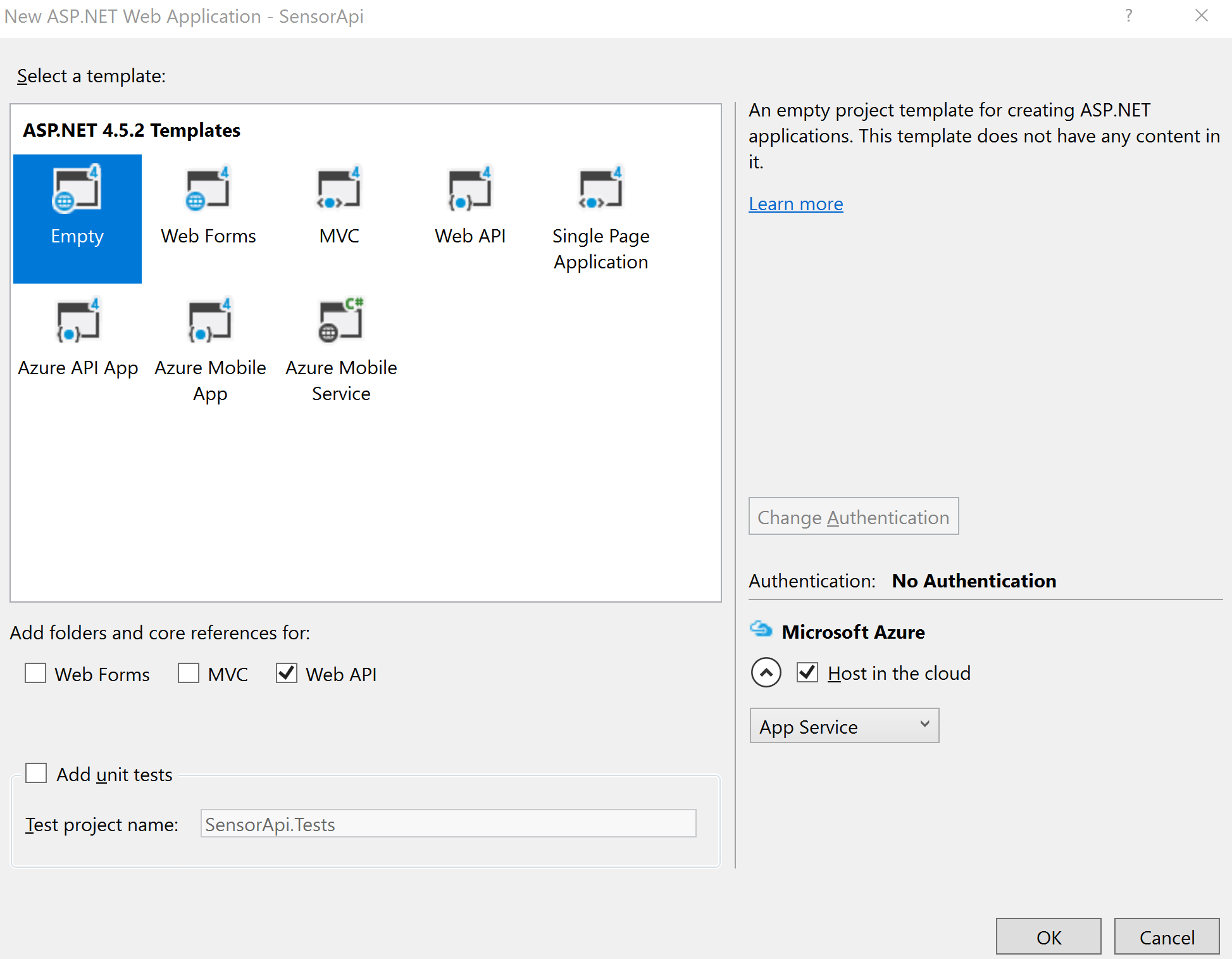
Hvis alt ser ut som det skal fungere, kan du bare starte jobben og samle inn sensor data.

# Web Api i C#

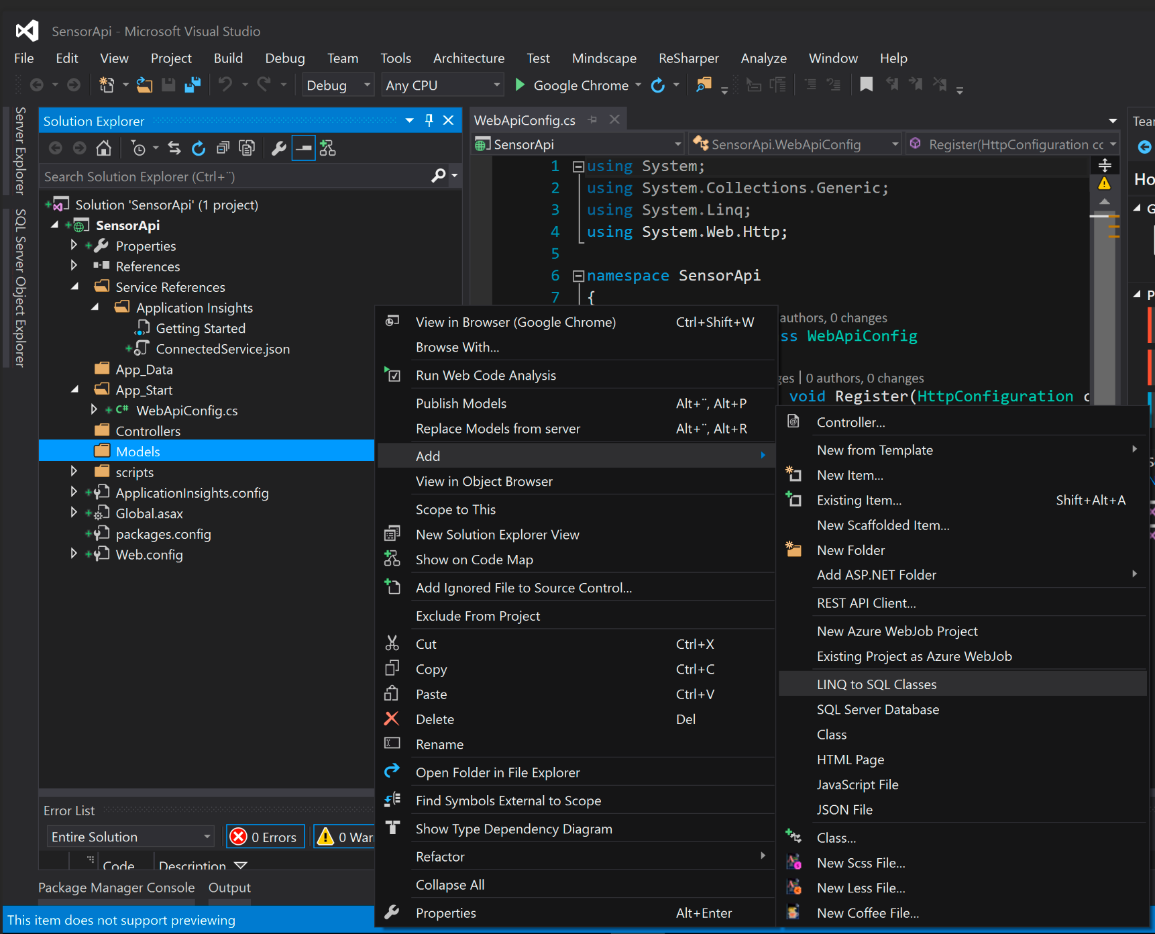
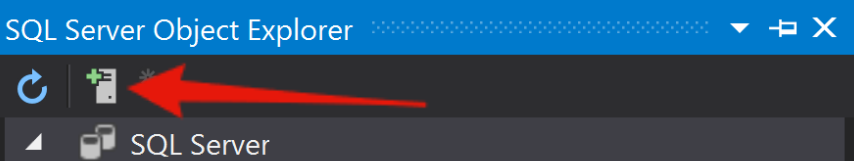
## Oppsett av prosjektet

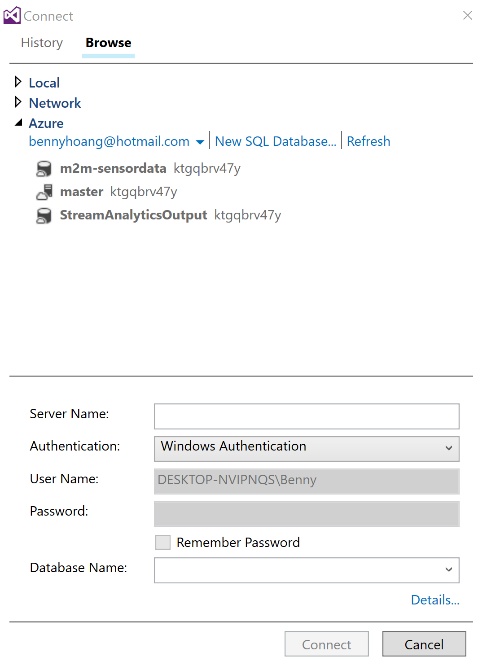
* Start med å lage ett nytt prosjekt ved å gå inn på **file** > **new** > **Project ...**
* Velg **ASP.Net Web application (.NET framework)**



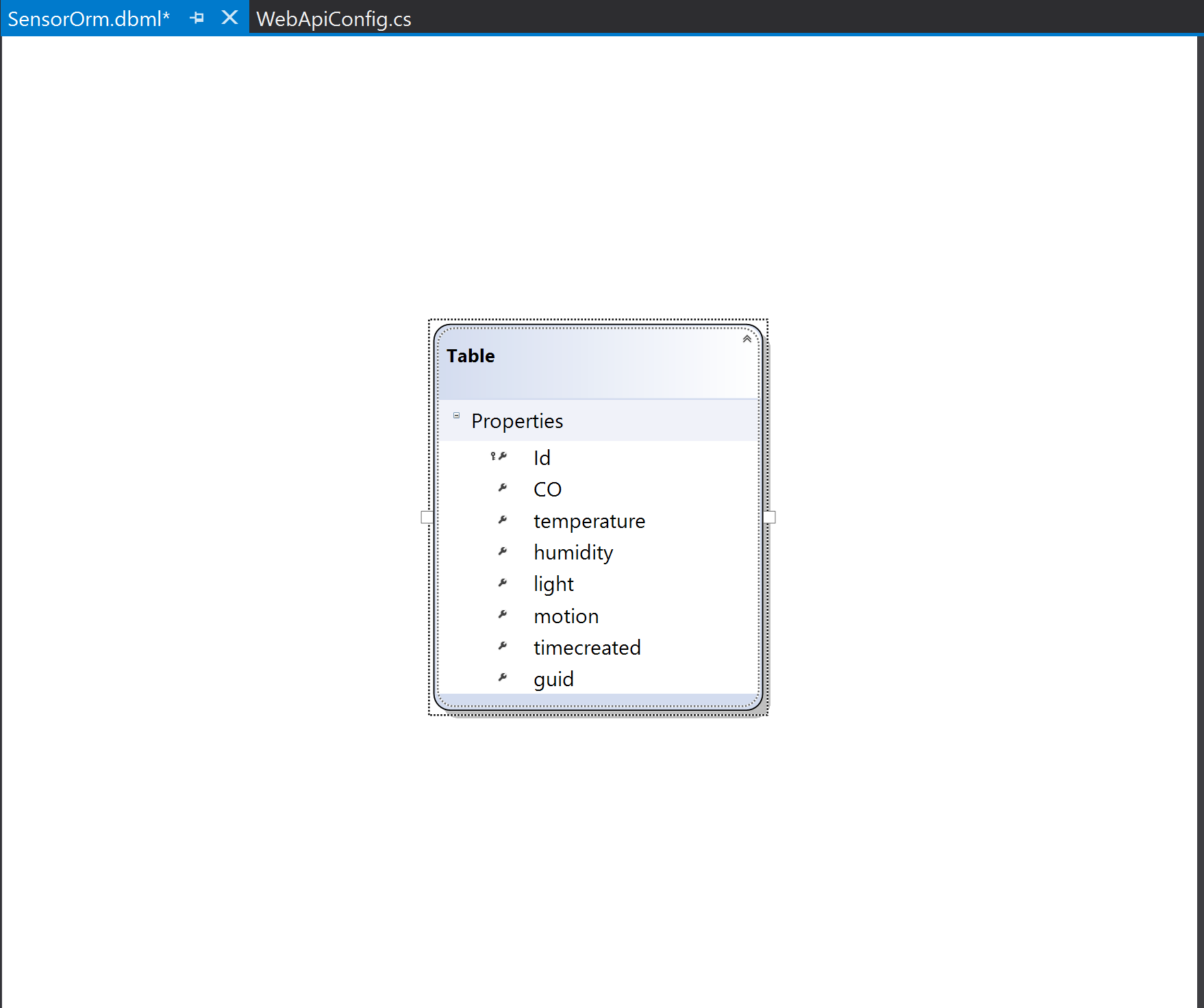
* Velg **Empty** og huk av på **Web API**. Det er også mulig å publisere api-et til azure hvis det er ønsket. 

## Lag Modell

* Legg til LINQ to SQL Classer ved å høyreklikke på Models. 
* Gå inn på **SQL Server Object Explorer** i venstre tab og velg din SQL server fra Azure. Hvis det ikke er allerede opprettet sql server kan du klikke på add **sql server** ikonet. 
* Du vil deretter få dialogboks om hvilken server du skal koble deg opp mot. Velg riktig database og deretter fyll inn feltene med riktig info.

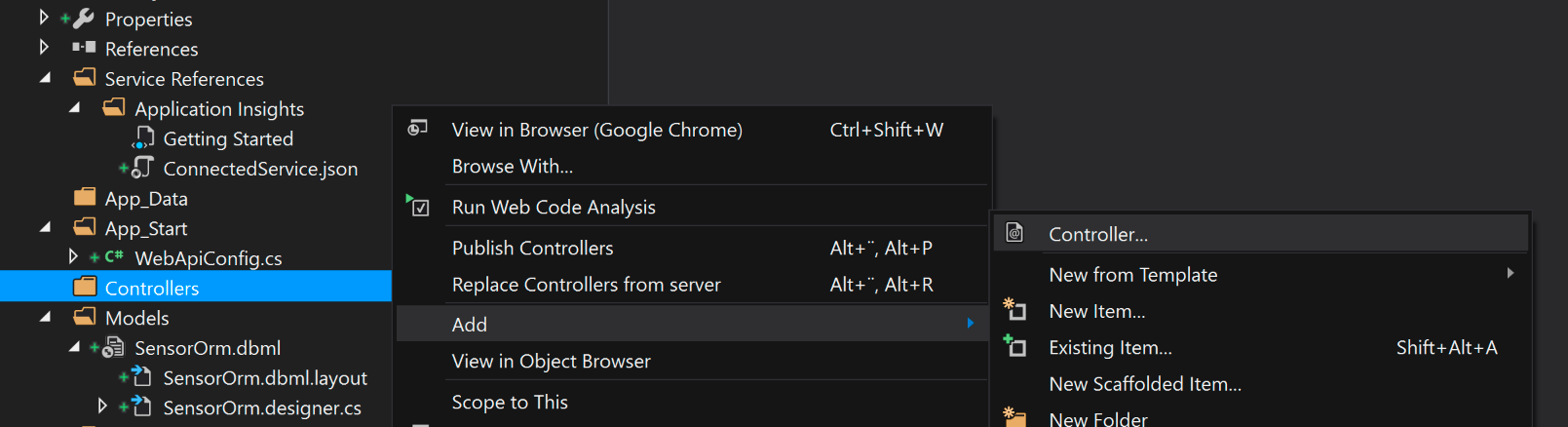
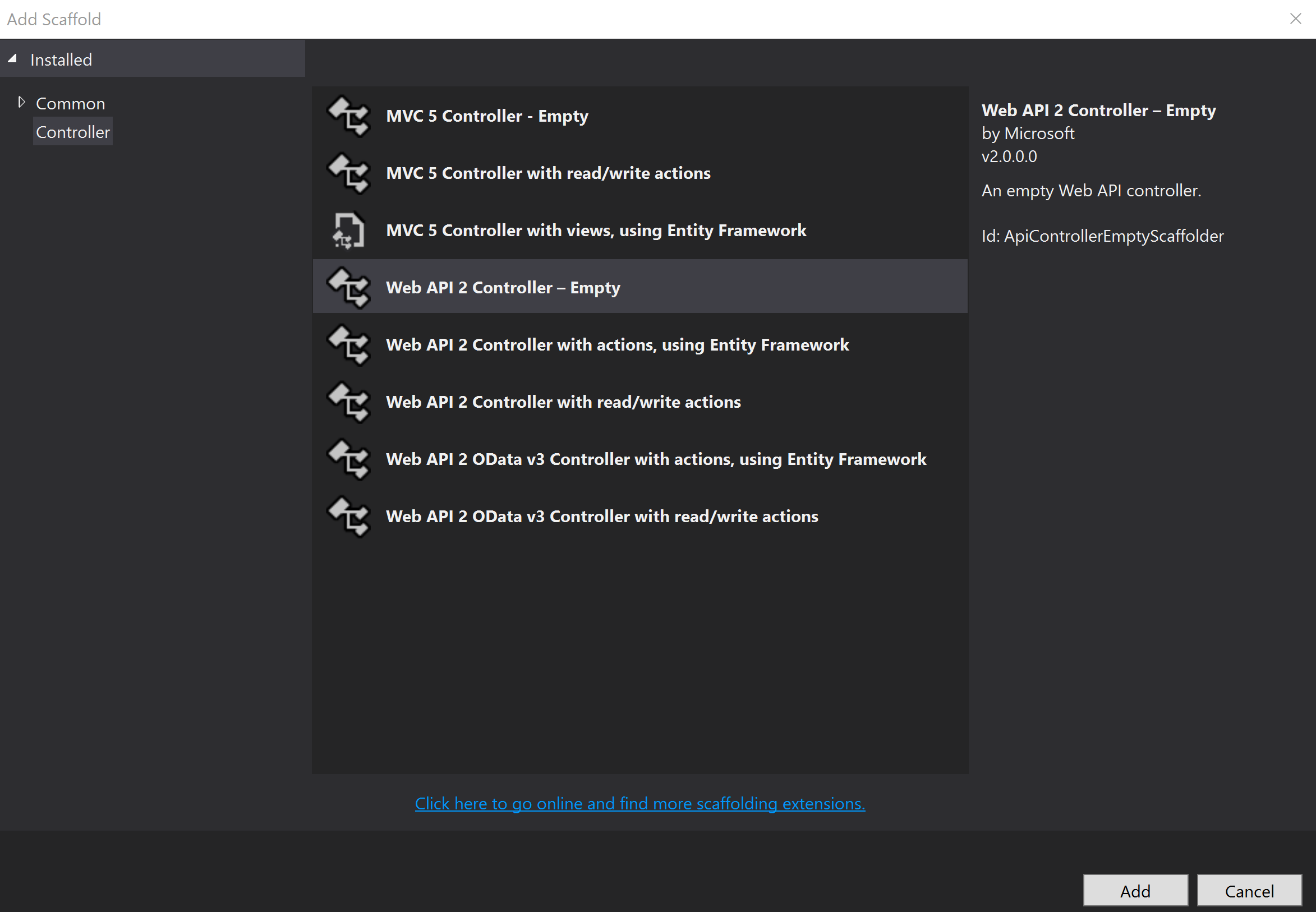


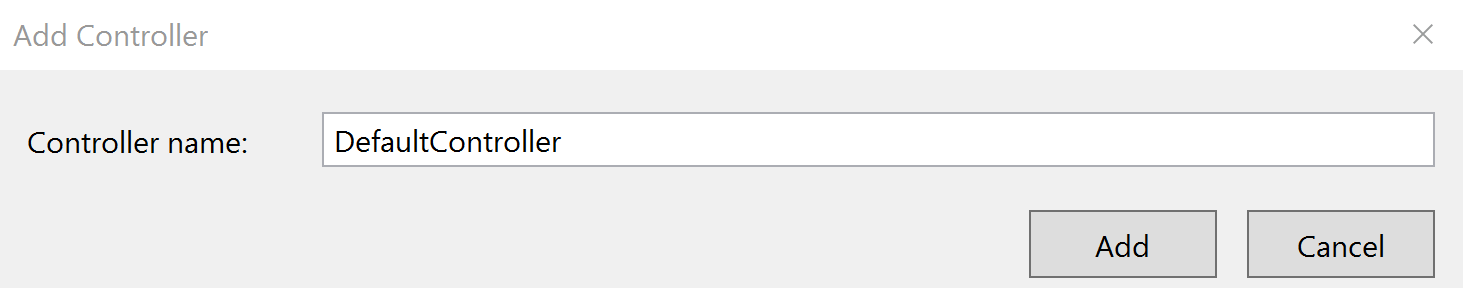
* Etter å ha åpnet sql serveren din kan du finne tabellen hvor sensordataene ligger. Deretter dra hele tabellen inn i det hvite vinduet i dbml filen. I mitt tilfelle heter tabellen min **dbo.Table**



For å gjøre det lettere i kode gir jeg tabellen et mer logisk og spesefikt navn slik at det blir lettere å holde tungen rett i munnen. Dette gjør du ved å dobbelklikke på tittelen, deretter lagre filen. På denne måten lager vi et objekt som representerer data i web-apiet. ASP.NET Web API kan automatisk serialisere modellen din til formater som **JSON** eller **XML.**

## Lag Controller

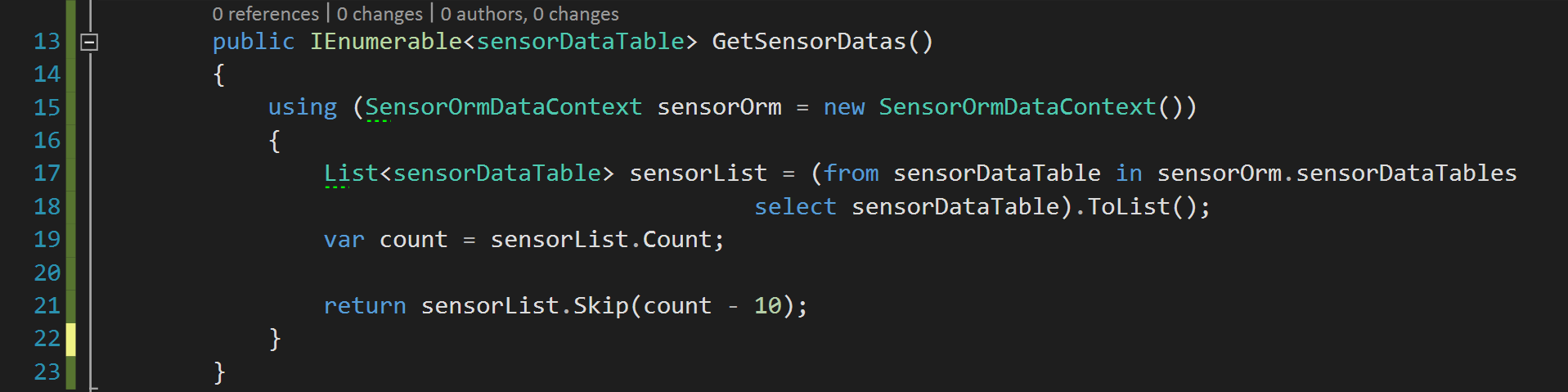
* Høyreklikk på **Controller** > **Add** > **Controller...** 
* I **Scaffold** dialogboksen velg **Web API 2 Controller – Empty**
* Gi deretter logisk navn til kontrolleren. Det er viktig å ha med prefixen «**Controller**»på slutten av kontrolleren din slik at applikasjonen kan vite at dette er en kontroller. Navnet må også følge PascalCase for at det skal fungere.



* Vi lager en metode som henter alt fra databasen og populerer den til en liste. For at vi skal kunne få tilgang til modellene som er opprettet må du inkludere modellene. Dette gjør du ved å skrive:

using <*NAMESPACE*>.Models;

Namespace er det prosjektet ditt heter. I mitt tilfelle skriver jeg **using SensorApi.Models;**

* Legg til følgene kode 

Denne metoden returner de 10 siste loggede sensor data. Dette er for å lett kunne se hva som ble logget sist istedenfor å bla igjennom en hel historikk med data.

Kjør koden og gå inn på ~/api/sensor så vil du kunne se loggen som er blitt publisert fra eventhuben.

Apiet mitt ligger publisert her: <http://sensorapi2030394.azurewebsites.net/api/sensor>

# Next steps

## Webside

Nå som Web-Api er på plass, skal det bli lettere å lage en nettside som henter sensordata. Akkurat nå har ikke løsningen noe brukerverdi. Ingen vanlige folk liker å lese JSON format (enda).

## Optimalisere Sensordata

Sensordataene som sendes inn kan kanskje forbedres mye mer. Jeg har vurdert om ikke alle data skal sendes med en gang eller ikke. Det har har jeg desverre ikke fått så mye tid til å titte på. Hovedfokuset var å få disse sensordataene til å kunne flowe gjennom stream analytics til en database. Eksempler på dete kan være at den leser av 10 verdier og tar gjennomsnittet av det for å så returnere en sensor data.

# Kilder

<https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf>

<https://www.hackster.io/falafel-software/pir-sensor-with-a-photon-particle-cloud-and-a-uwa-97e79b>

<https://cdn-learn.adafruit.com/downloads/pdf/pir-passive-infrared-proximity-motion-sensor.pdf>

<https://www.asp.net/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api>

<https://www.linkedin.com/pulse/getting-started-azure-iot-services-stream-analytics-rob-tiffany>

<https://github.com/BennyHoang/webutvikling_innlevering_2>