dotnet Cologne 2013

Einführung in die Windows Azure Mobil Services

03.05.2013

Stefan Lange
empira Software GmbH
Stefan.Lange@empira.de

Agenda

- Warum Azure Mobile Services?
- Überblick Features
- Beispielanwendung
- Einsatzszenarien
- Fazit

Fragen / Diskussion

Was sind die Azure Mobile Services?

Automatisch generierter REST-basierter Web-Services mit

- Zugriff auf strukturierten Speicher (MSSQL Server)
- Zugriffskontrolle
- Server-seitiges Scripting
- Authentifizierung
- Notifications
- Diagnose und Logging
- Skalierbar

Alles was man als Backend für seine App braucht

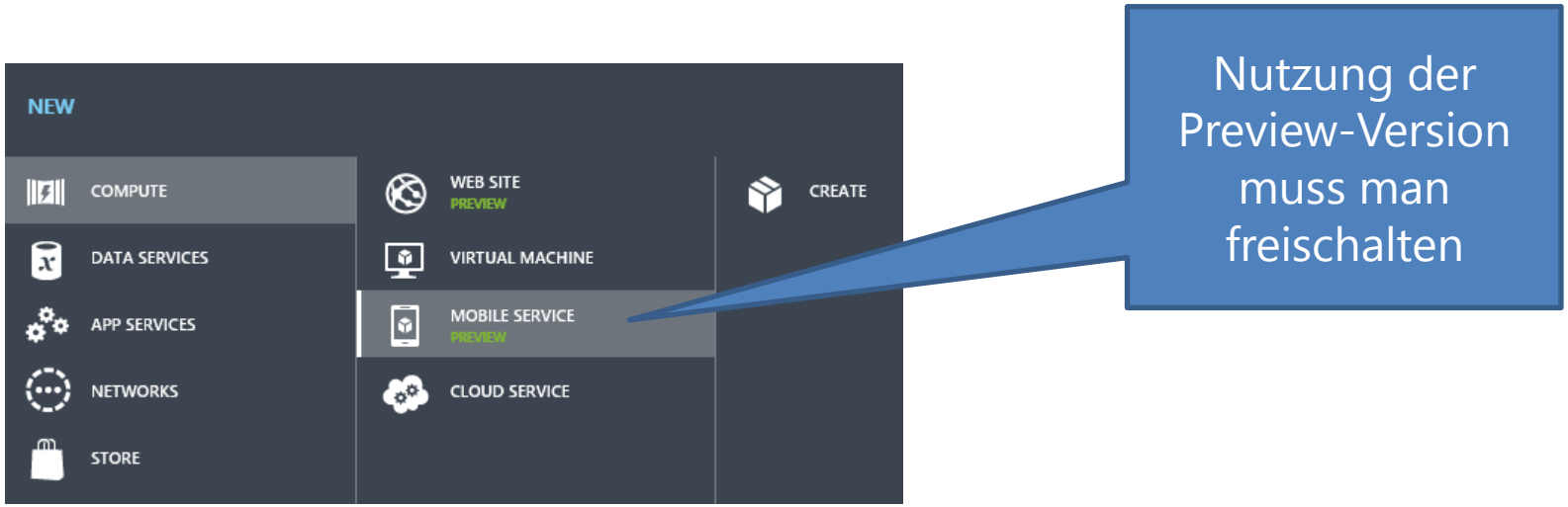
DEMO

- Überblick Windows Azure Portal
- Anlegen eines Azure Mobile Services

Nutznung der Azure Mobile Services

Prerequisites

- Windows Azure Account, z. B. den 90-Tage “Free Trial” oder ein passendes MSDN Abo



Sicht auf die Daten

Client-Seite

- Jeder Zugriff geht auf genau eine Tabellen
- Keine Joins möglich
- Vorteil: Man kann nicht so leicht was falsch machen

Server-Seite

- Default: Durchreichen der Operation an die Datenbank
- Flexibel erweiterbar durch Script-Code
- Ebenfalls sehr einfaches Konzept
- Joins prinzipiell über Views möglich, aber nicht zu empfehlen

Das REST API der Azure Mobile Services

REST API auf Tabellen einer SQL Server Datenbank

Unterstützte HTTP-Verben

- GET – Einen oder mehrere Einträge abfragen
- POST – Neuen Eintrag anlegen
- PATCH – Einen Eintrag aktualisieren
- DELETE – Eintrag löschen

URI-Format

```
https://<service_name>.azure-mobile.net/tables/<table_name>[/<item_id>]
```

- Bei GET ggf. noch weitere Parameter wie *where*, *skip*, *take*, etc.
- Das Datenformat ist JSON

Zugriffsrechte pro Tabelle und Operation

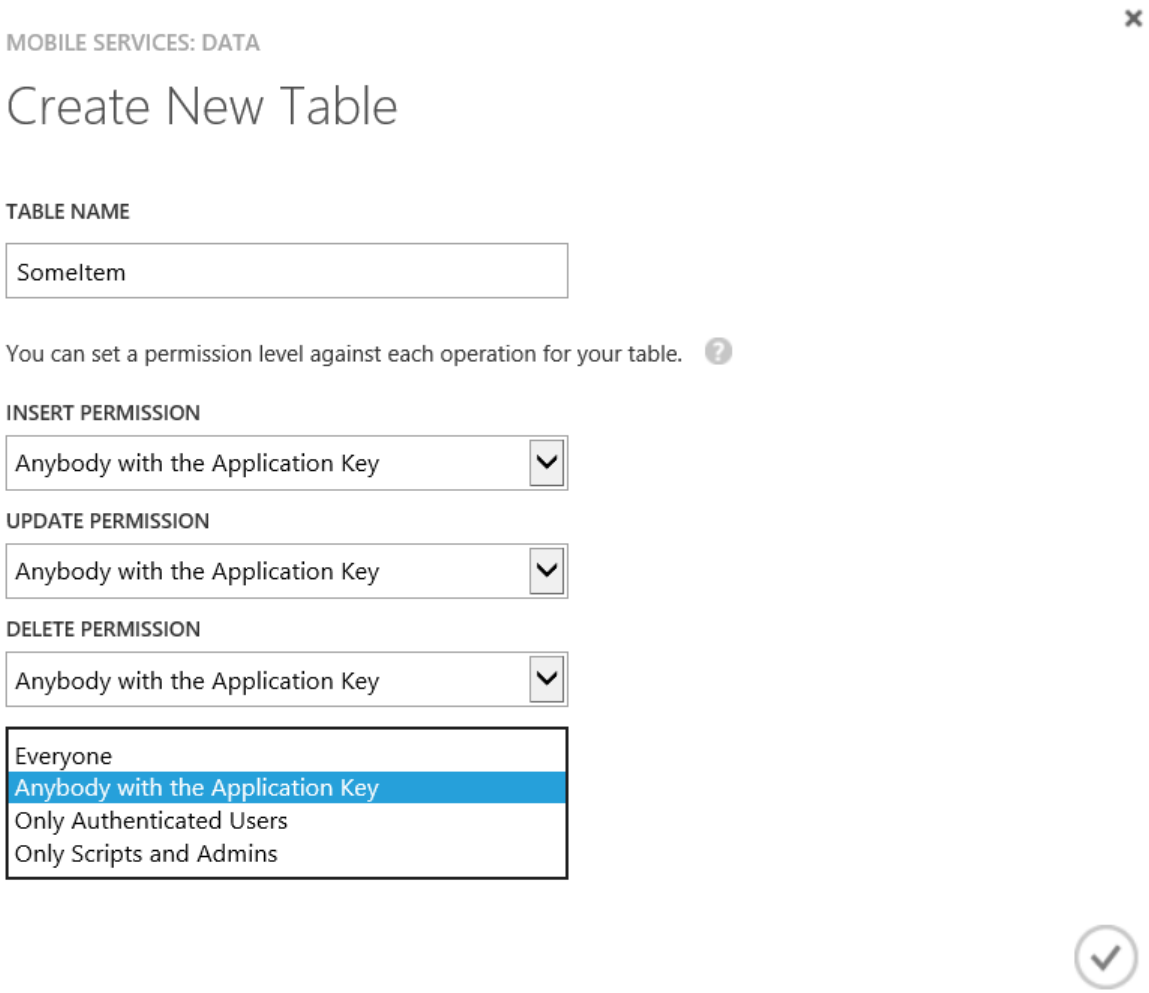
Die vier Operationen je Tabelle

- INSERT (http POST)
- UPDATE (http PATCH)
- DELETE (http DELETE)
- READ (http GET)

Mögliche Berechtigungen je Operation

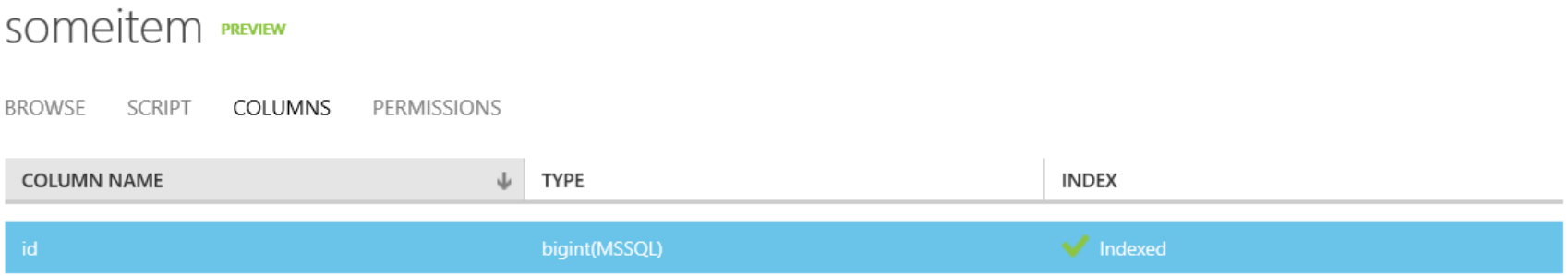
- Everyone
- Anybody with the Application Key
- Only Authenticated Users
- Only Scripts and Admins

Anlegen einer Tabelle im Portal



Default einer neuen Tabelle

Automatische ID-Spalte mit Index drauf



DEMO

- Anlegen einer Tabelle
- REST API mit Hilfe von POSTMAN ausprobieren
- Dynamisches Schema
- Zugriffskontrolle

Zugriffsschlüssel

Application Key

- Nur für Entwicklung und Tests verwenden
- Nicht in fertige App einbauen
  (besser Benutzerauthentifizierung verwenden)

Master Key

- Administration, Backup, Installation
- Server zu Server Kommunikation
- Auf gar keinen Fall rausgeben
- Kann Scripts abschalten

Serverseitiges Scripting

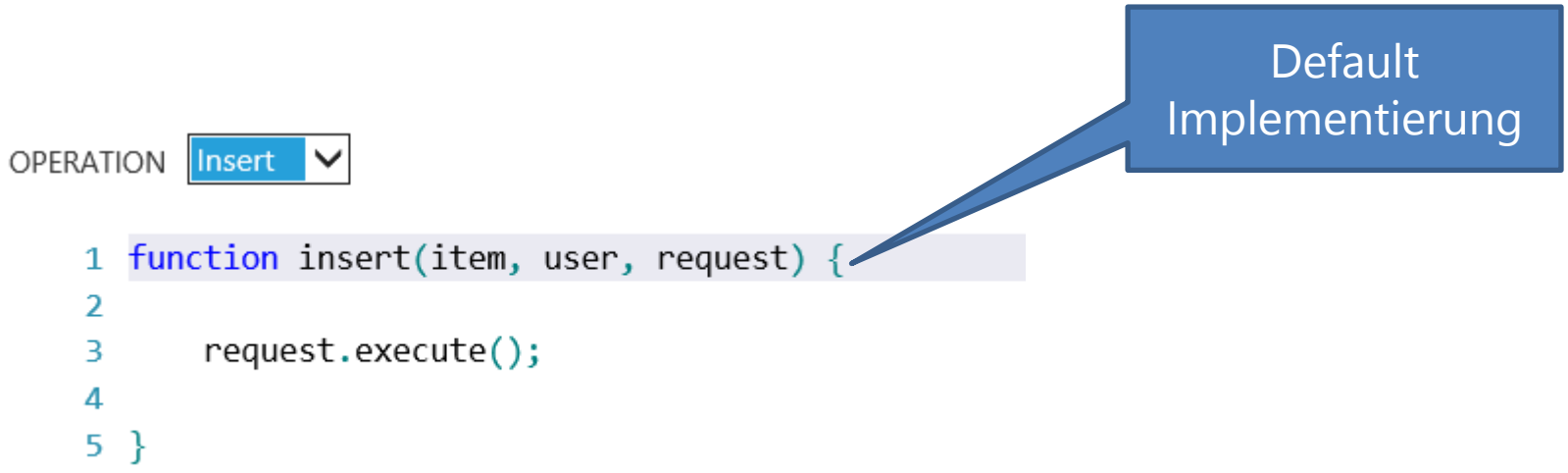
Ein Script pro Operation (insert, update, delete, read) und pro Tabelle

- Basierend auf node.js
- Zurzeit JavaScript, weil universell verwendbar
  (C# ist aber ganz oben auf der Wunschliste)

node.js basiert auf Googles JavaScript Technologie und läuft jetzt auf Microsoft Azure Servern

Serverseitiges Scripting

Standard-Script am Beispiel der Insert-Operation



item: das einzufügende Item
user: der angemeldete Benutzer
request: das Requestobject dieser Operation

Serverseitiges Scripting

Veränderung des einzufügenden Items

OPERATION Insert

```
1 function insert(item, user, request) {
2
3     item.created = new Date();
4     request.execute();
5
6 }
```

Vor dem Insert Spalte füllen.

Serverseitiges Scripting

Ändern des Ergebnisses nach execute

OPERATION Read

```
1  function read(query, user, request) {
2
3      request.execute({
4          success: function(results) {
5              var now = new Date();
6              results.forEach(function(item) {
7                  item.retrievedAt = now;
8                  delete item.id;
9              });
10             request.respond(); // Writes the response.
11         }
12     });
13
14 }
```

Nach dem Lesen Element hinzufügen bzw. löschen

JavaScript ist hier besser geeignet als C#

Serverseitiges Scripting

Logging

OPERATION Insert

```
1  function insert(item, user, request) {
2
3      item.created = new Date();
4
5      request.execute({
6          success: function(){
7              console.log("Successfully inserted:", item);
8              request.respond();
9          }
10     });
11
12 }
```

Log via console Objekt

Serverseitiges Scripting

Ergebnisse vollständig selbst berechnen

OPERATION Read

```
1 function read(query, user, request) {
2
3     // Do some calculation...
4
5     request.respond(200, "Hallo Welt");
6 }
```

Eigener HTTP Statuscode und eigenes Ergebnis

Stets vorhandene JavaScript Objekte

Table
Operationen auf Tabellen

tables
Operationen auf Tabellen

User
Aktueller User

mssql
Datenbankzugriff über Transact-SQL

console
Logging

statusCodes
HTTP Status Codes

Weitere Module über require("...")

apns
Apple Push Notification Service

azure
Windows Azure SDK

mpns
Windows Phone 8 Push Notification Service

request
Externer REST Service

sendgrid
E-Mail Service

wns
Windows Store Notification Service

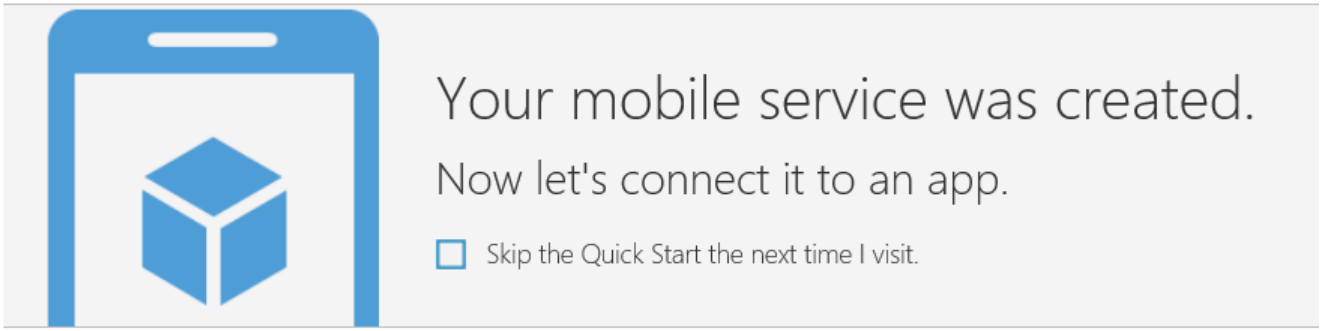
Anprogrammieren der Mobile Services

- Portal generiert Beispiel-Apps
  - Windows / WinRT
  - Windows Phone 8
  - iOS, Androis, HTML
  - Beliebige via REST API

Prerequisites

- Visual Studio 2012
- Windows Phone SDK 8.0
- Live SDK for Windows and Windows Phone (optional)
- Ggf. weitere Tools für iOS oder Android
- Windows Azure Account oder 90 Tage “Free Trial”

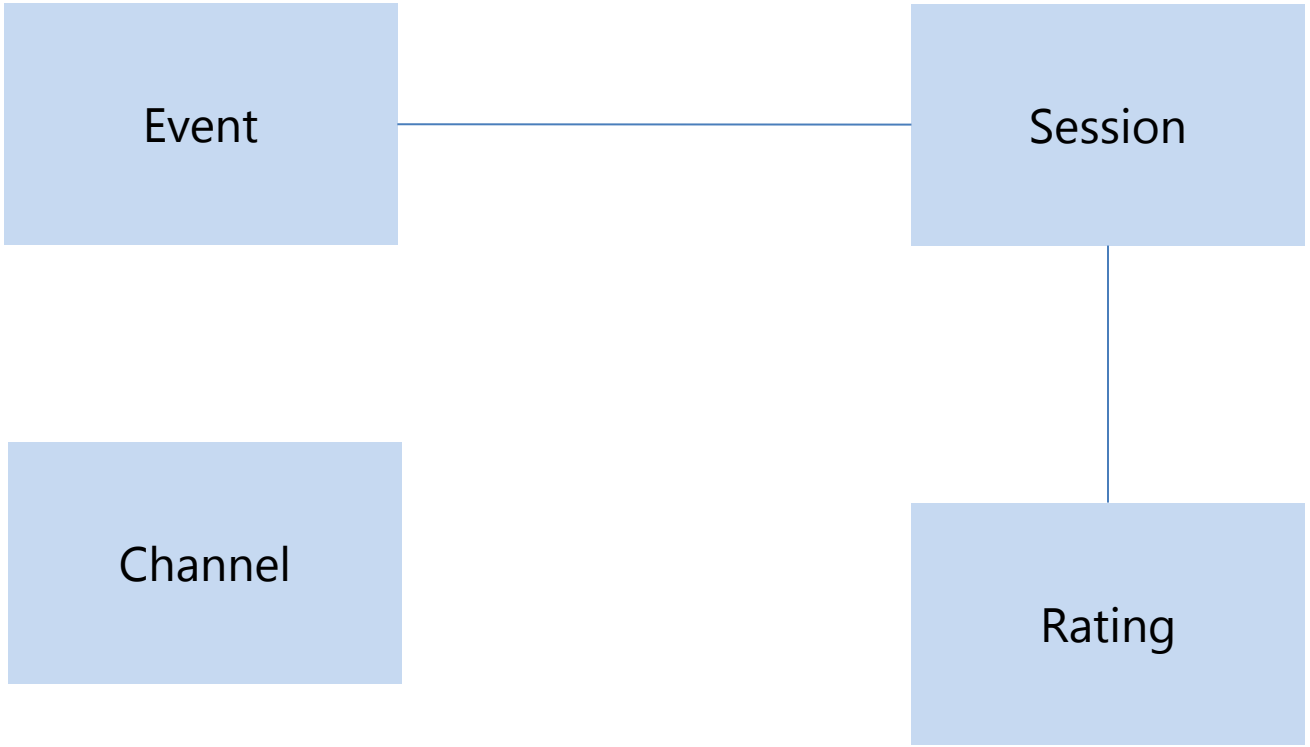
Quick Start im Azure Portal



Beispielprojekt „Event Buddy“

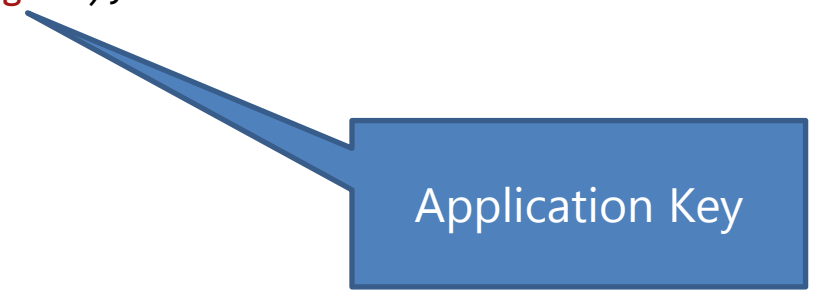
Anlegen von Events, Sessions und deren Bewertung

Entities



Initialisierung des Mobile Service Clients

```
sealed class App : Application
{
    // Create Mobil Service Client singleton in App class.
    public static MobileServiceClient MobileService =
        new MobileServiceClient(
            "https://eventbuddy-stla.azure-mobile.net/",
            "YGClaqlKzjZrf3iUIQLyLzCwBzUJyg57");
    …
}
```



Grundprinzip: *„Keep it simple“*

Beispielhafte Datamodel Klasse

```
public class MyData
{
    [JsonProperty(PropertyName = "id")]
    public int Id { get; set; }

    [JsonProperty(PropertyName = "name")]
    public string Name { get; set; }

    [JsonProperty(PropertyName = "start")]
    public DateTime Start { get; set; }

    [JsonProperty(PropertyName = "end")]
    public DateTime End { get; set; }

    [JsonProperty(PropertyName = "description")]
    public string Description { get; set; }
}
```

PropertyName weil JSON camelCasing verwendet

REST Operationen in Code

Einfügen neues Element (http POST)

```
await App.MobileService.GetTable<MyData>().InsertAsync(item);
```

Aktualisieren eines Elements (http PATCH)

```
await App.MobileService.GetTable<MyData>().UpdateAsync(item);
```

Löschen eines Elements (http DELETE)

```
await App.MobileService.GetTable<MyData>().DeleteAsync(item);
```

Lesen von Elementen (http GET)

in Kombination mit Where, Skip, Take, etc.

```
await App.MobileService.GetTable<MyData>().ReadAsync(…);
```

DEMO

- Anlegen eines neuen Events in Event Buddy App

Speichern eines neuen Events

```
private async Task SaveEvent(Event item)
{
    // Save a new event.
    await App.MobileService.GetTable<Event>().InsertAsync(item);
    Events.Add(item);
}
```

Nach dem Aufruf hat item eine gültige Id erhalten.

Laden der Events

```
private async Task LoadEvents()
{
    // Query for all existing events.
    Events = await App.MobileService.GetTable<Event>().ToEnumerableAsync();
}

public dynamic Events
{
    get { return _events; }
    set
    {
        _events.Clear();
        foreach (var item in value)
            _events.Add(item);
    }
}
readonly ObservableCollection<Event> _events = new ObservableCollection<Event>();
```

Benutzer-Authentifizierung

Azure Mobile Services hat vier fertige Identity-Provider

- Microsoft Account (Live ID)
- Facebook Login
- Twitter Login
- Google Login
- (weitere in Vorbereitung…)

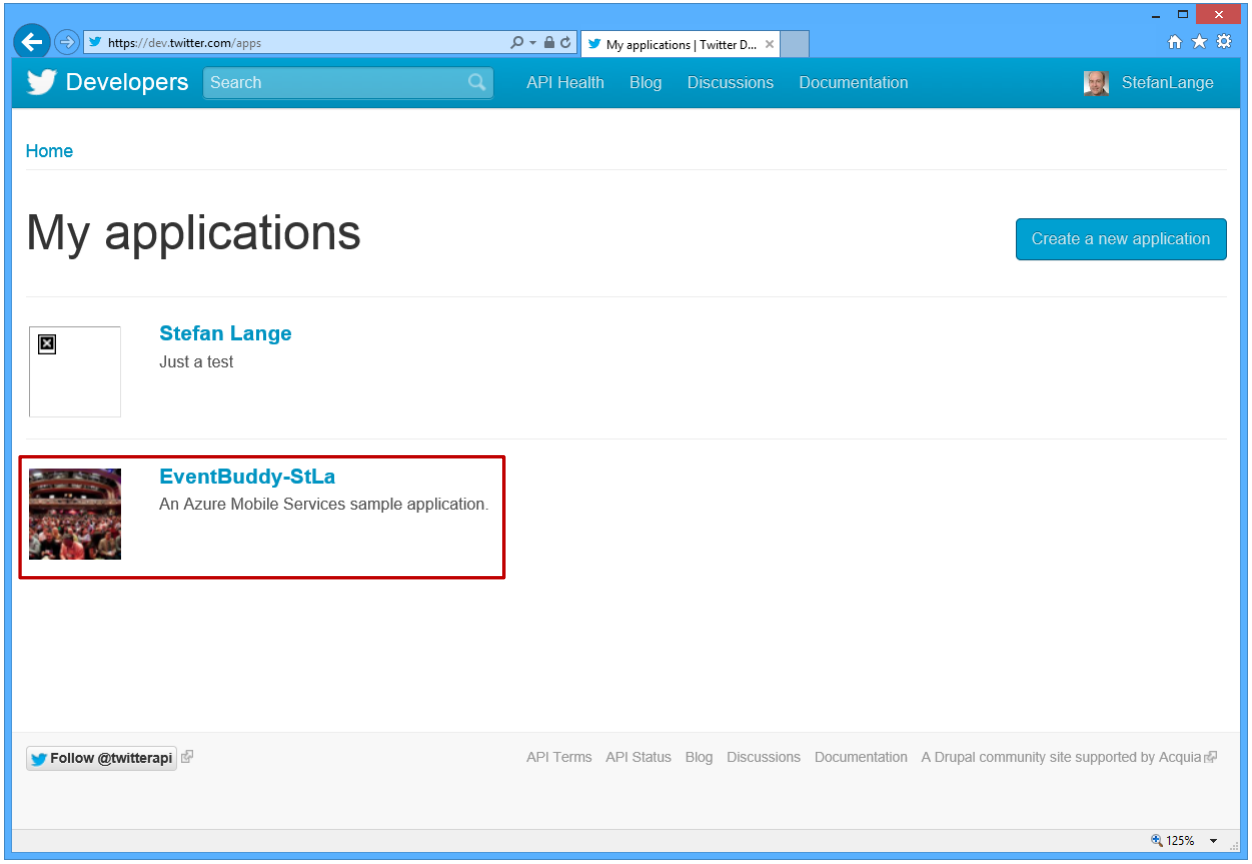
Vorteile der fertigen Provider

- Keine eigene Benutzerverwaltung
- Sicherheit für den Anwender, denn App kennt die Credentials nicht
- Man kann sich auf seine eigentliche Anwendung konzentrieren
- Eigener Identity Provider ist ein Projekt für sich und zudem noch ein ziemlich anspruchsvolles

Vorbereitung des Logins via Twitter

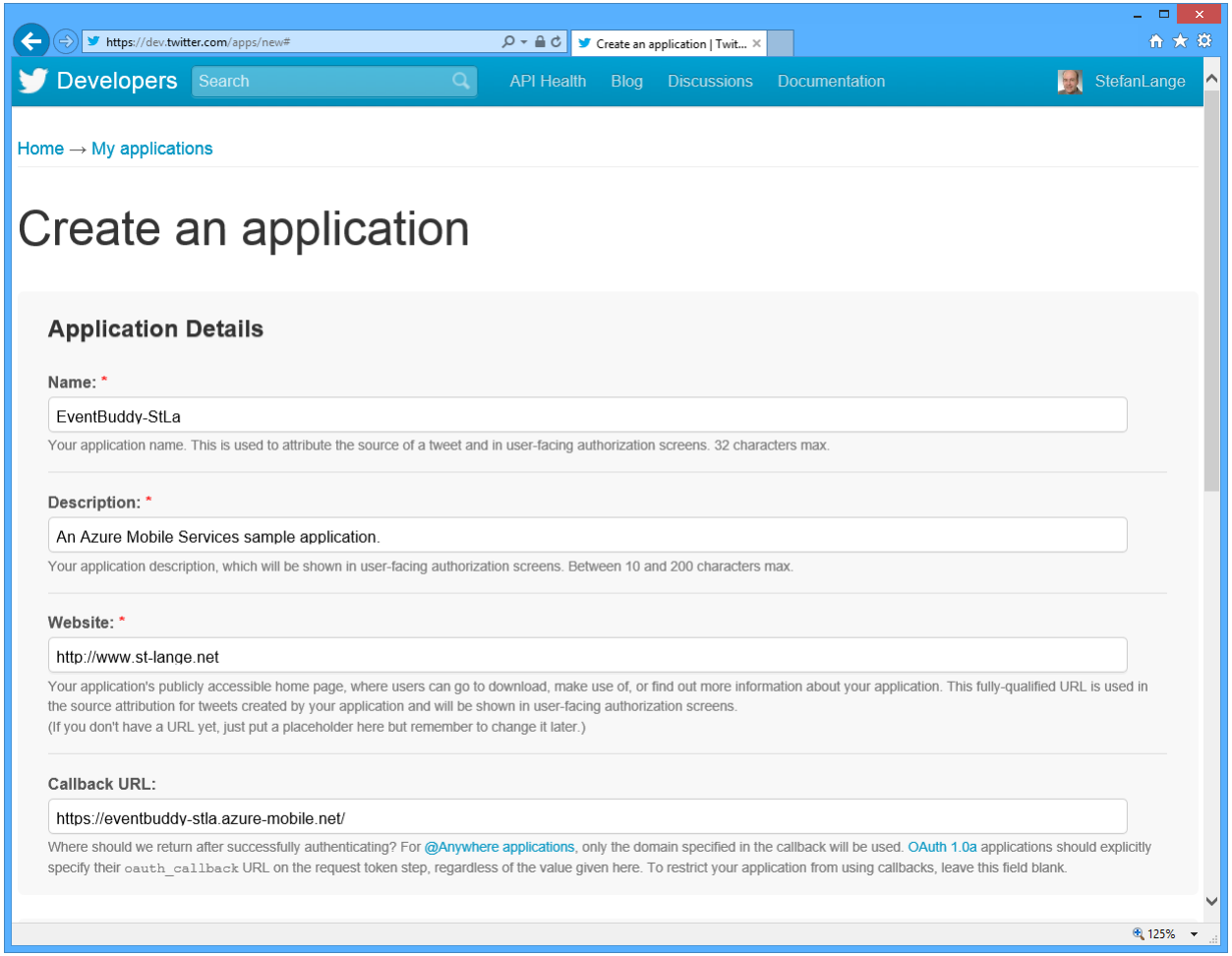
Twitter die eigene Anwendung bekannt machen

- https://dev.twitter.com/apps

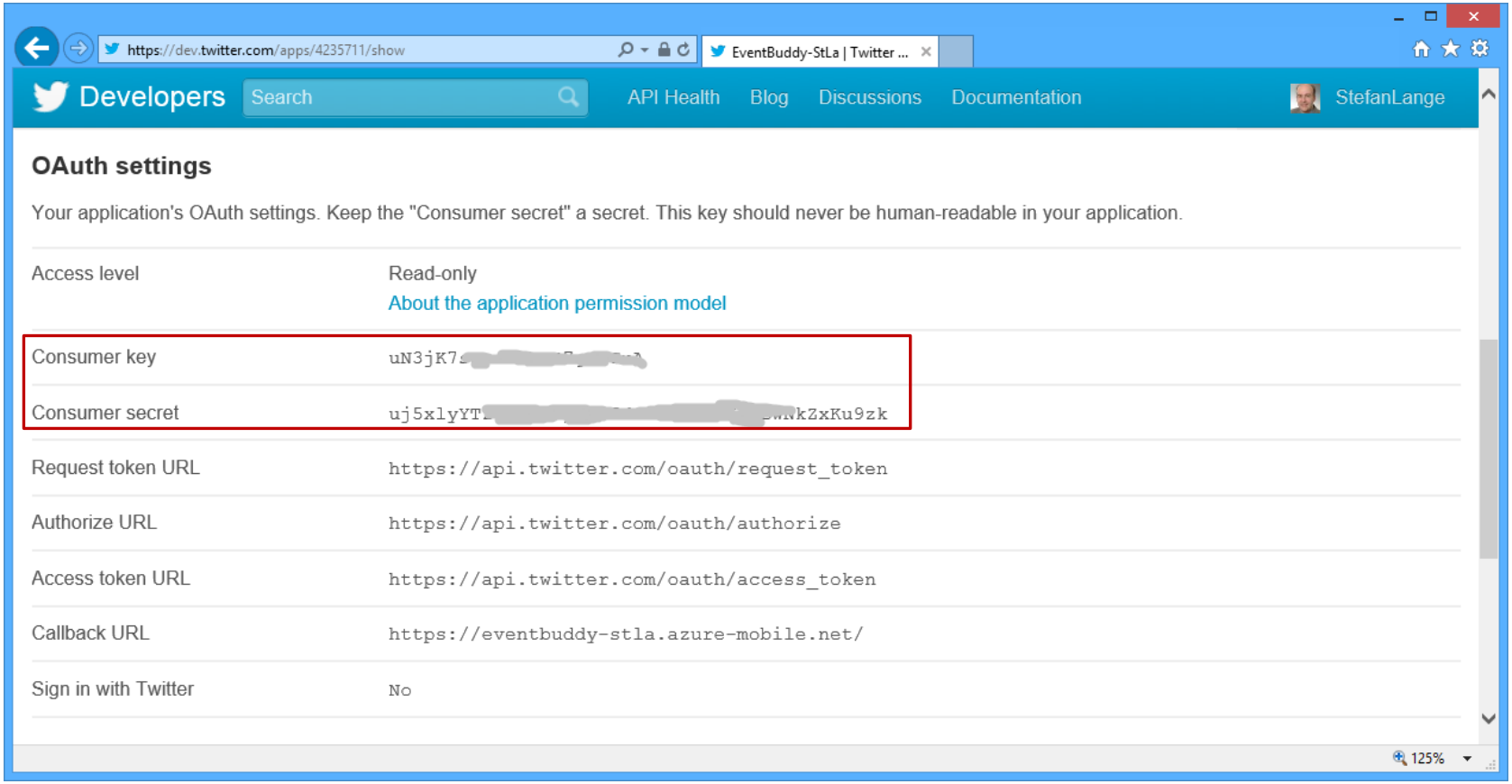


Anlegen einer neuen Application

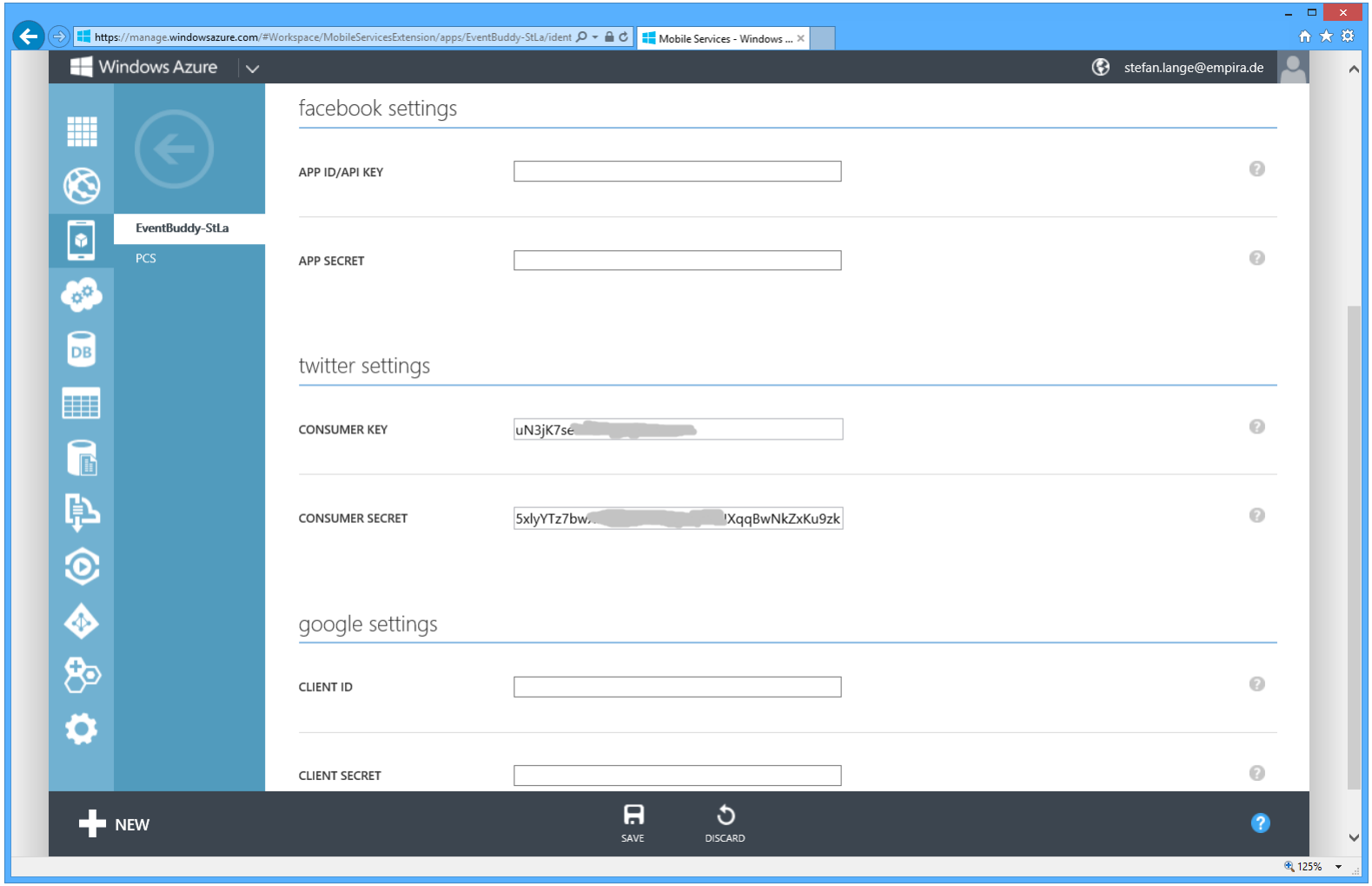
Der Mobile Service wird bei Twitter registriert (nicht die eigentliche App)



Twitter generiert die OAuth Settings



Eintragen in den Mobile Service



Login

```
public async Task LoginTwitter()
{
    var user = App.MobileService.CurrentUser;
    while (user == null)
    {
        string message;
        try
        {
            user = await App.MobileService.LoginAsync(
                MobileServiceAuthenticationProvider.Twitter);
            message = string.Format("You are now logged in as '{0}'",
                user.UserId);
        }
        catch (InvalidOperationException)
        {
            message = "You must log in. Login Required";
        }
        var dialog = new MessageDialog(message);
        dialog.Commands.Add(new UICommand("OK"));
        await dialog.ShowAsync();
    }
}
```

DEMO

- Mit Twitter-Konto anmelden

Twitter Bild-URL speichern

OPERATION Insert

```
1  function insert(item, user, request) {
2      item.userId = user.userId;
3
4      if (item.speaker) {
5          var url = "https://api.twitter.com/1/users/show.json?screen_name=" + item.speaker;
6          var req = require("request");
7
8          req.get(url,
9          function(error, result, body) {
10             item.img = "";
11             if (error || result.statusCode != 200) request.execute();
12
13             var json = JSON.parse(body);
14             if(json.profile_image_url){
15                 var biggerImg =  json.profile_image_url.replace("normal","bigger");
16                 item.img = biggerImg;
17             }
18
19             request.execute();
20         });
21     }
22     else {
23         item.img = "Assets/NoProfile.png";
24         request.execute();
25     }
26 }
```

Push Notification

- Via Script werden Notifications an registrierte Geräte geschickt

(Demo mit Windows Phone)

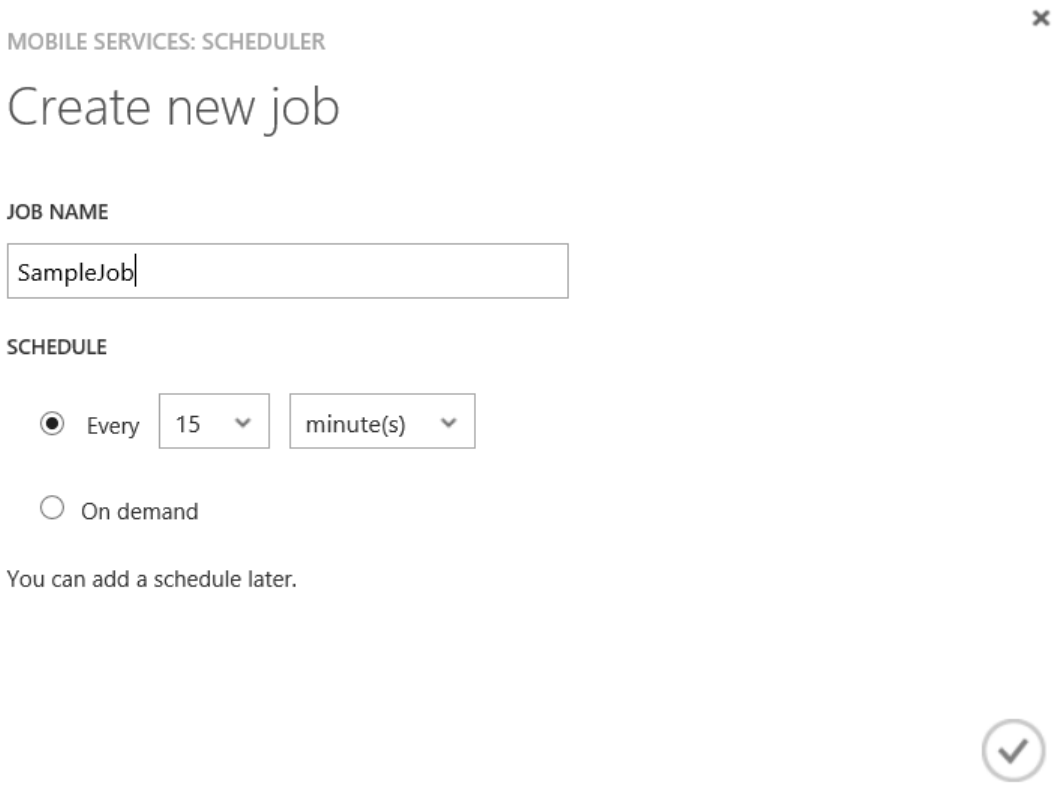
Scheduler Jobs

Periodischen Ausführen eines Scripts

- Bereinigen der Datenbank
- Archivierung alter Datensätze
- Verarbeiten von Datensätzen, z.B. Bilder skalieren
- Verschicken von Nachrichten

Anlegen eines neuen Jobs

Maximal 1 Job bei einem kostenlosem Mobile Service möglich



Beispiel Job

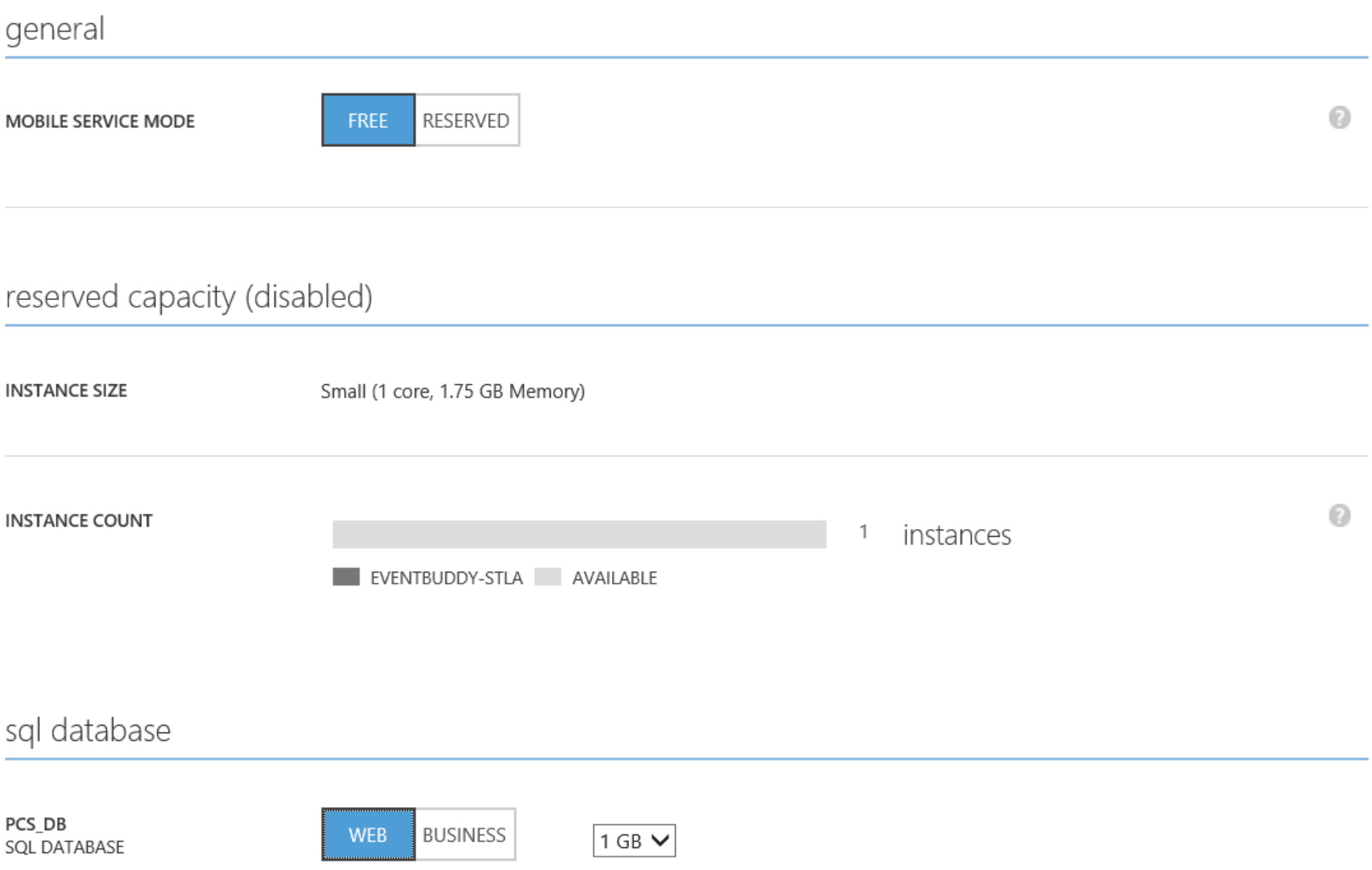
Löschen doppelter Channel-Einträge

```
function SampleJob() {
    cleanup_channels();
}

function cleanup_channels() {
    var sql = "SELECT MAX(Id) as Id, Uri FROM Channel " +
        "GROUP BY Uri HAVING COUNT(*) > 1";
    var channelTable = tables.getTable('Channel');

    mssql.query(sql, {
        success: function(results) {
            if (results.length > 0) {
                for (var i = 0; i < results.length; i++) {
                    channelTable.del(results[i].Id);
                    console.log('Deleted duplicate channel:' +
                    results[i].Uri);
                }
            } else {
                console.log('No duplicate rows found.');
            }
        }
    });
}

```

Azure Console

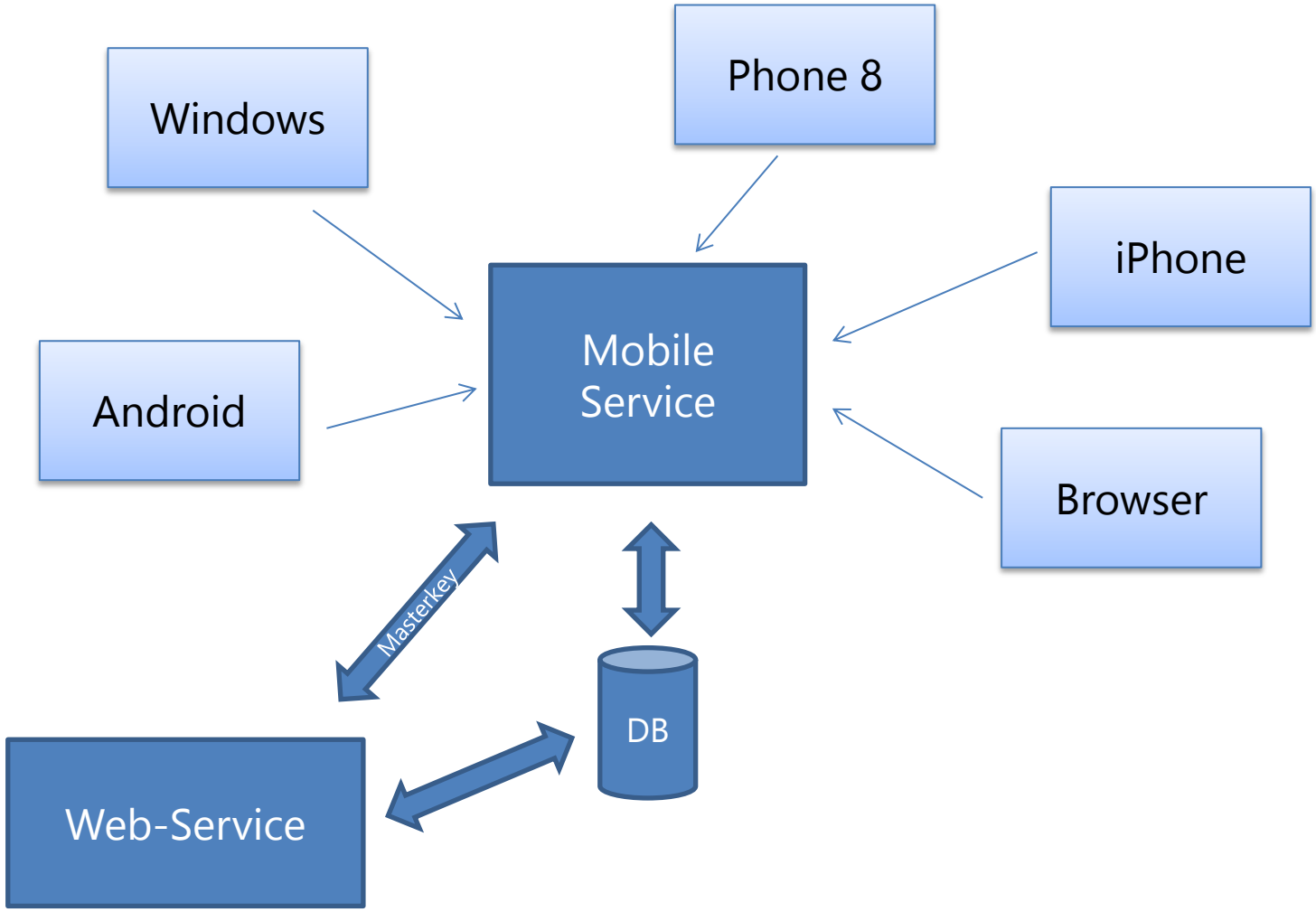
- Anlegen und Konfigurieren von Mobile Services per PowerShell Commandline
- Beispielsweise Backup der Scripts aller Tabellen
- Verwalten von Dev/Staging/Prod Services

Skalierbarkeit des Services



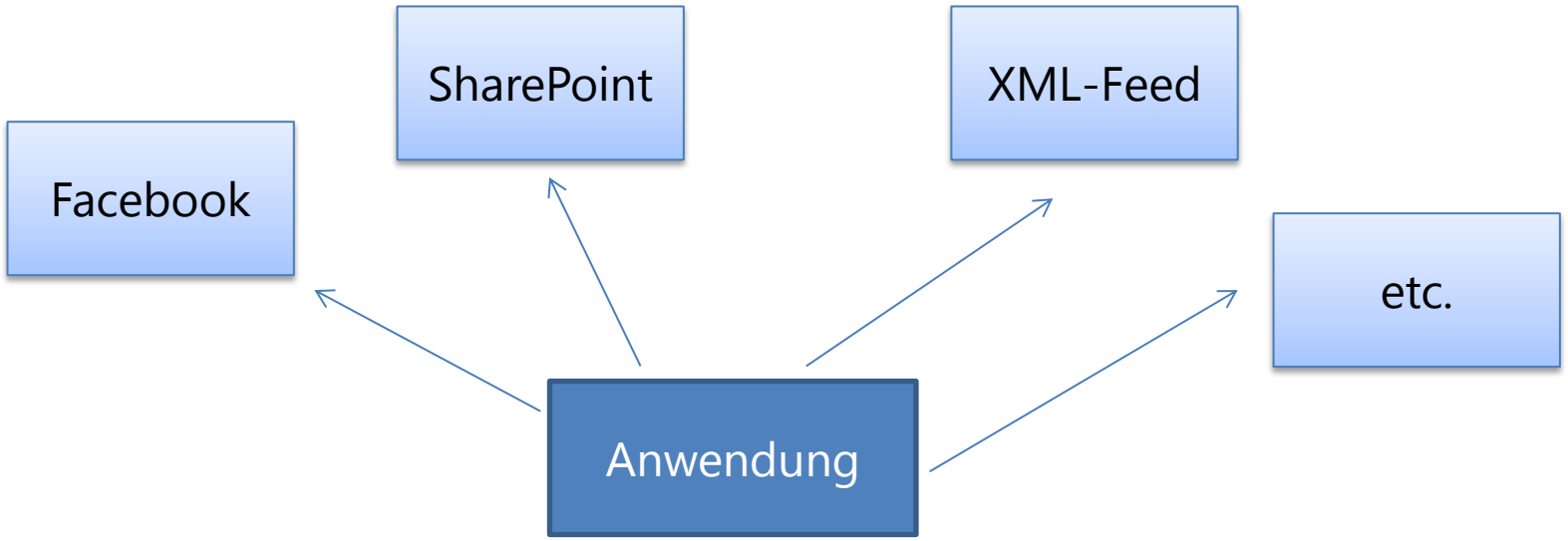
Beispiel Szenario 1

Daten synchronisieren zwischen Geräten



Beispiel Szenario 2 (ohne WAMS)

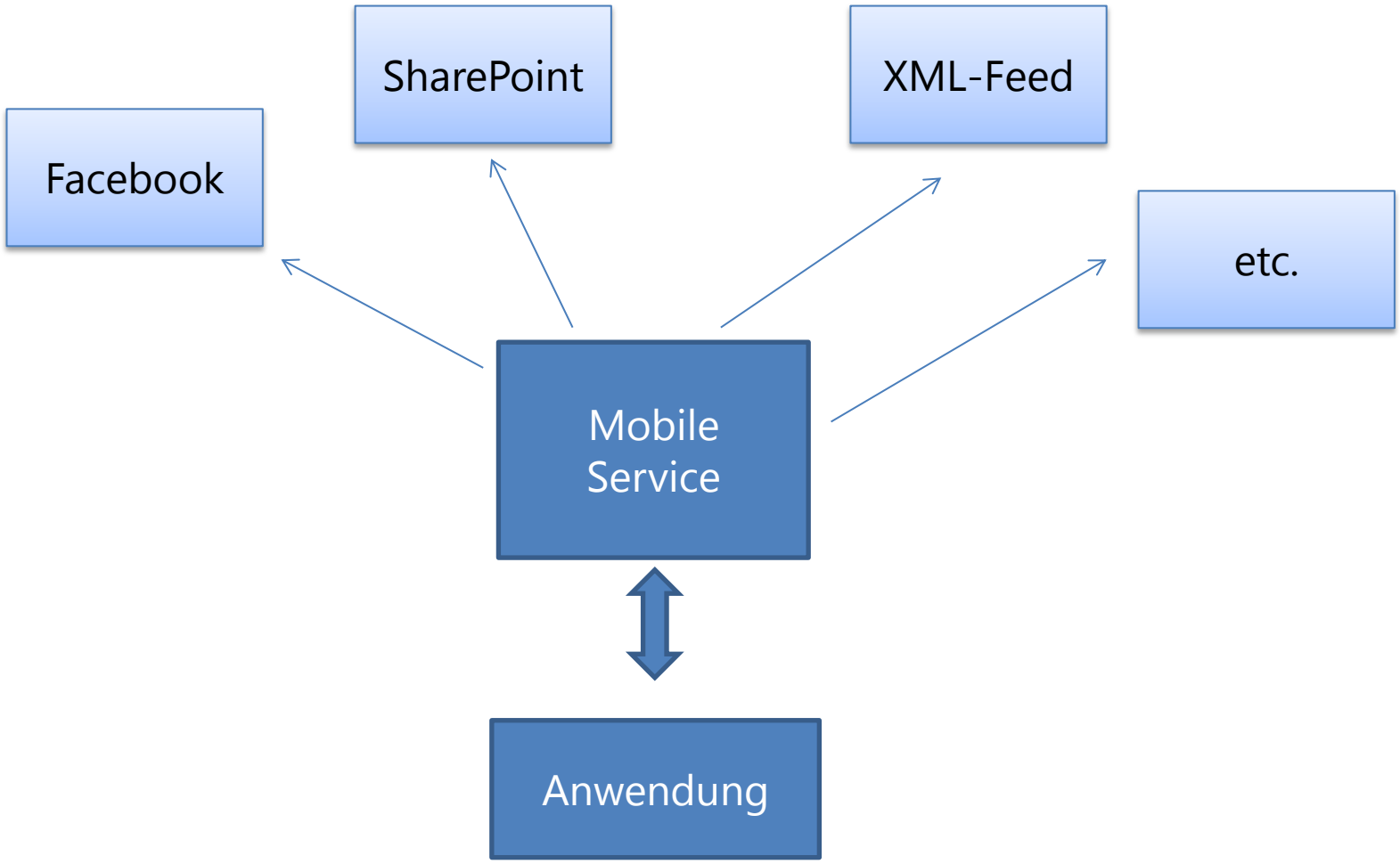
Eine Anwendung mit vielen Datenquellen



Wenn API sich irgendwo ändert, crasht die Anwendung

Beispiel Szenario 2 (mit WAMS)

Datenintegration über „Single API“ mit Mobile Services

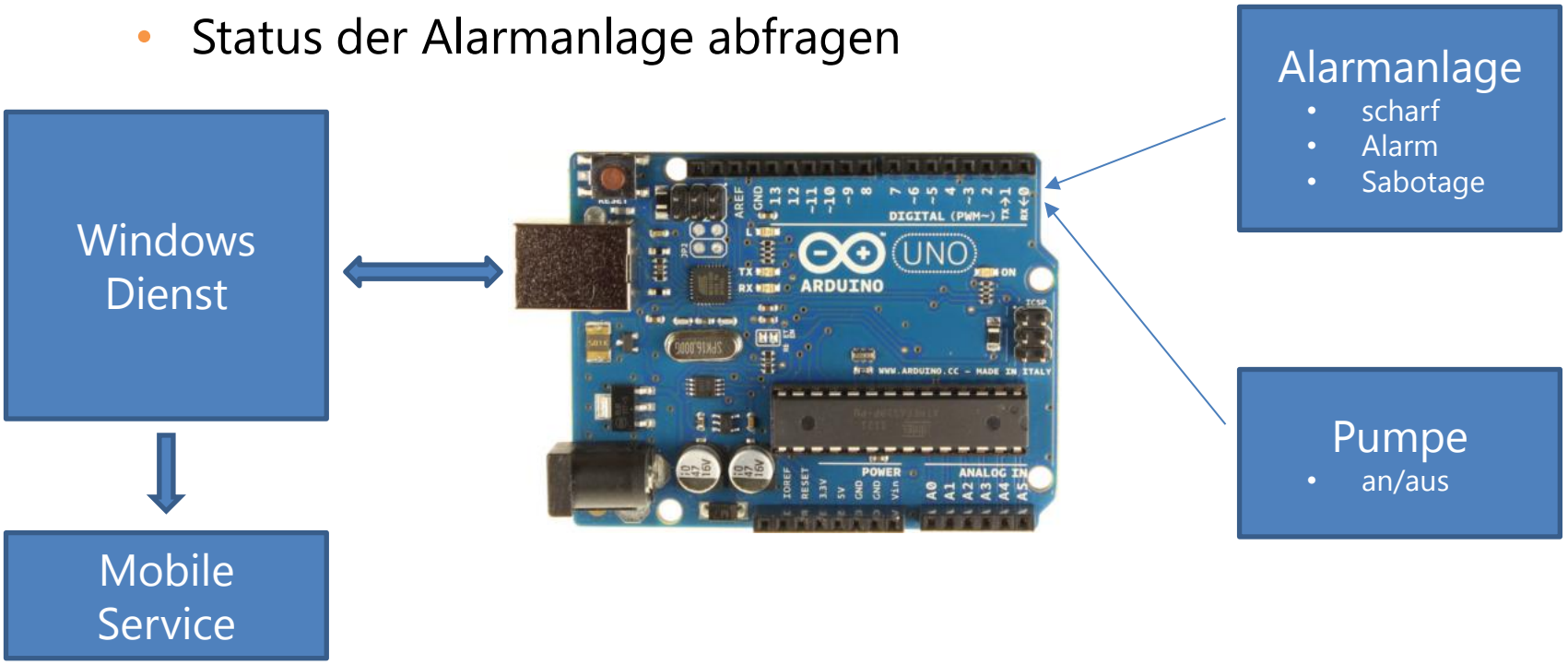


Azure Mobile Services

Wieso eigentlich ***Mobile*** Services?

Was ich zurzeit gerade damit mache…

- Überwachung der Kondensatpumpe (Heizung) und der Alarmanlage
- Sensorenabfrage mit Arduino-Bord über USB an einen Windows-Service in einem PC
  - Info aufs Handy wenn Pumpe ausfällt
  - Status der Alarmanlage abfragen



Fazit

- Alles was man für sein App Backend braucht
- Einstieg einfach und kostenlos
- Fertige Bausteine „out of the box“
- Keine „externen“ Hürden mehr bei der App-Entwicklung
- Focus auf die Apps, nicht auf die Infrastruktur
- Auch über Mobile Szenarien hinaus praktisch

Super coole Sache – unbedingt ausprobieren

Mehr Info zu Mobile Services

- Josh Twist
  http://www.thejoyofcode.com/
- Carlos Figueira
  http://blogs.msdn.com/b/carlosfigueira/
- Nick Harris
  http://www.nickharris.net/

Das war’s

Vielen Dank für Eure Aufmerksamkeit

Fragen?

Stefan.Lange@empira.de