

API Apps for web, mobile and logic apps

“microservices”

About Me...

- Michaël Van Wesemael
- Trainer @ U2U in
 - Web Development (ASP.NET, MVC)
 - Windows Phone, Windows Store apps
 - Azure
- Contact me: michael@u2u.be
- Twitter PiekenPuil
- <http://blogs.u2u.be/michael>
- “Putting the punk back into punctuality”

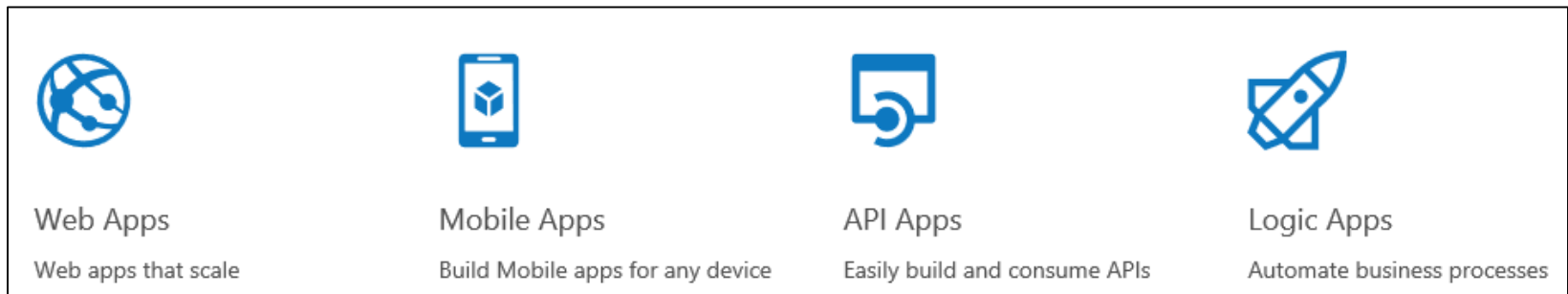


Agenda

- App Services & microservices
- What is an API app ?
- Creating an API app
- Exposing metadata with Swagger
- Deployment
- Creating an API Client
- Logic Apps
- Adding API Apps to logic apps
- Authentication

App Services

- Single Service combining
 - Azure Web Apps
 - Azure Mobile Apps
 - Azure API apps
 - Azure Logic apps

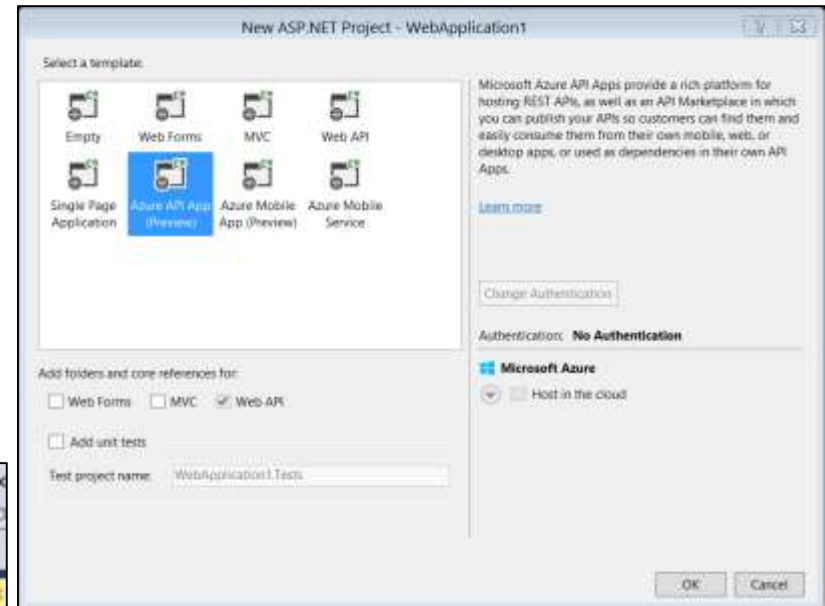
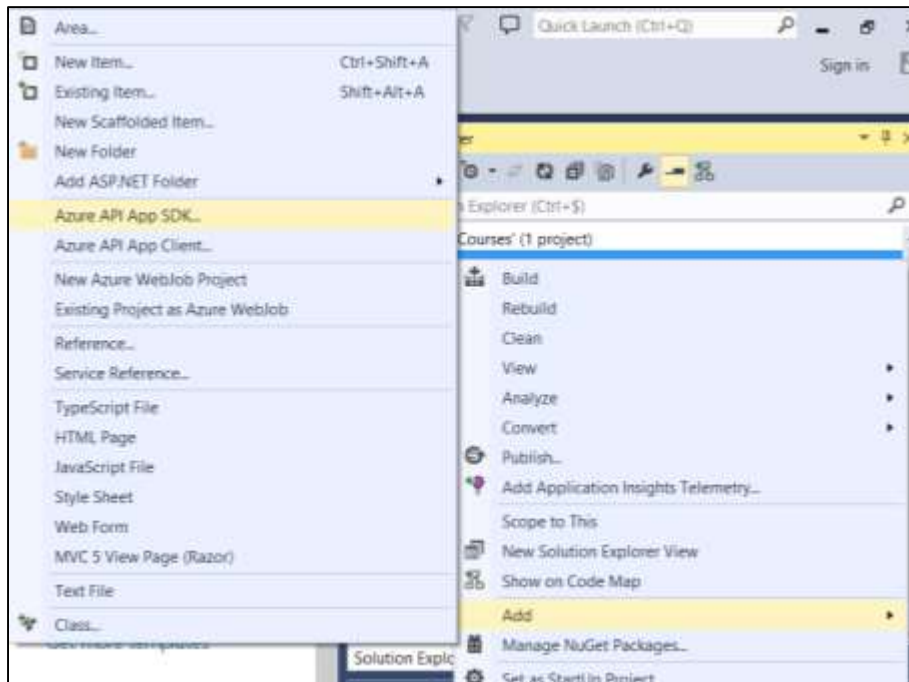


- Grouped in Resource Groups
- “MicroServices” : the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms (*M.Fowler*)

- The API Apps Service provides a rich platform for building, consuming and distributing APIs
- Choice of technology
 - ASP.NET, Java, PHP, Node.js, Python
- Simple Access Control
- Easy consumption through SDK generation
- Integrates with SaaS applications (O365, Dropbox, Twitter, Facebook,...) and Logic Apps
- Apps can be added or downloaded from marketplaces
- Auto-installation of dependencies

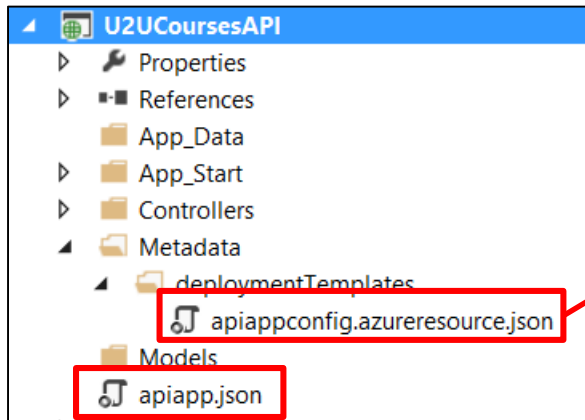
Creating an API Apps

- Create a brand new API app
- Or convert an existing one



Creating an API App

- 2 files are needed for deploying to Azure:



DeploymentInfo

Definition how the app will appear in Azure

API metadata

- Metadata is contained in apiapp.json

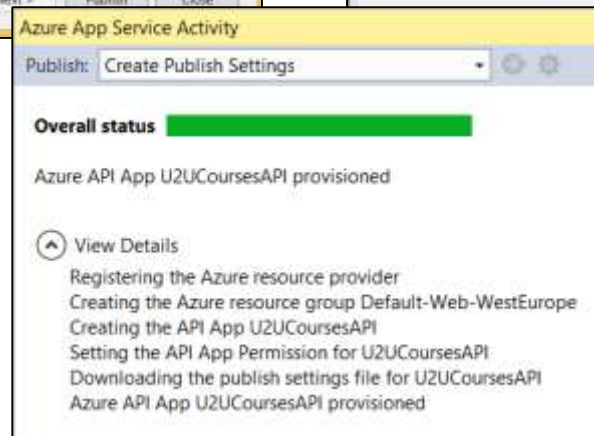
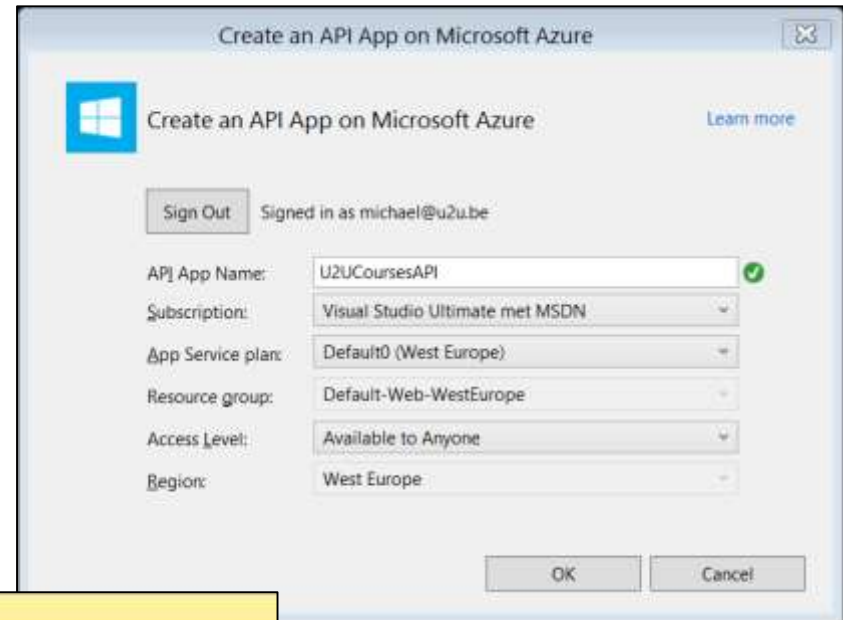
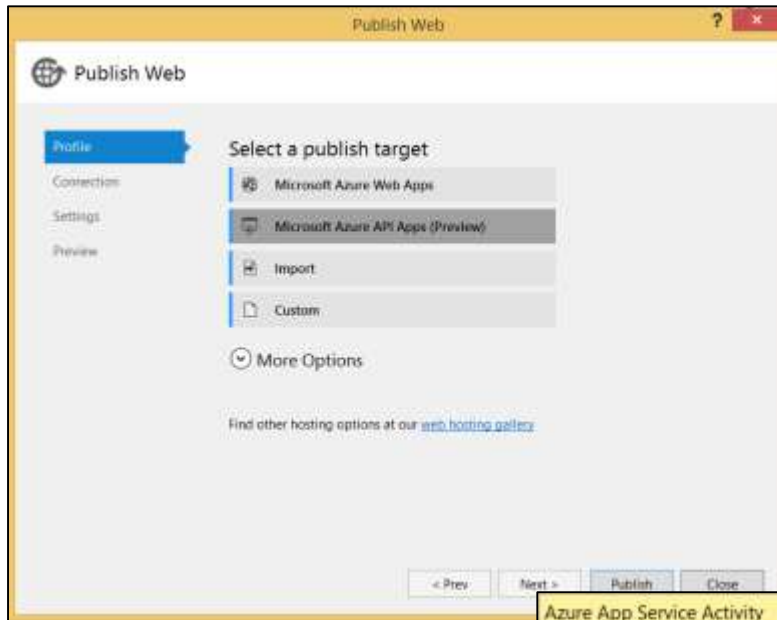
Name	Description
Id	Must be unique per namespace
Namespace	Default is microsoft.com
Gateway	Gateway-version. Gateways are used for a.o. authentication
Categories	Category for marketplace
Copyright	Copyright notice
...	...

```
{
  "$schema": "http://.../apiapp.json#",
  "id": "U2UCoursesAPI",
  "namespace": "microsoft.com",
  "gateway": "2015-01-14",
  "version": "1.0.0",
  "title": "U2UCoursesAPI",
  "summary": "",
  "author": "",
  "endpoints": {
    "apiDefinition": "/swagger/docs/v1",
    "status": null
  }
}
```


Demo

Deploying your API app

- Just call "Publish" on your project



Deploying your API App



- Check your app in the portal

The screenshot displays the Azure portal interface for the U2UCoursesAPI application. The top navigation bar includes options for Settings, Start, Stop, Restart, Update, and Delete. The main content area is divided into several sections:

- Essentials:** Provides key information about the app, including the resource group (Default-Web-WestEurope), URL (https://microsoft-apiapp3e1bd503c4b44...), status (Running), location (West Europe), package (Microsoft.ApiApp-0.9.4), and subscription details.
- Requests and errors:** A line chart showing the number of requests over time, with a peak of 16 requests. Below the chart, it shows 0 HTTP server errors and 0 HTTP 5xx errors.
- Usage:** Displays File System Storage usage at 1.71% and CPU Percentage usage. A warning icon indicates an "Error loading part" with a "Click to reload" option.
- Log Stream:** A table on the right side of the screen showing log entries with columns for OPERATION, HTTP METHOD, and LOGIC APP TRIGGER. The entries are:

OPERATION	HTTP METHOD	LOGIC APP TRIGGER
/api/Courses	GET	
/api/Courses	POST	
/api/Courses/{id}	GET	

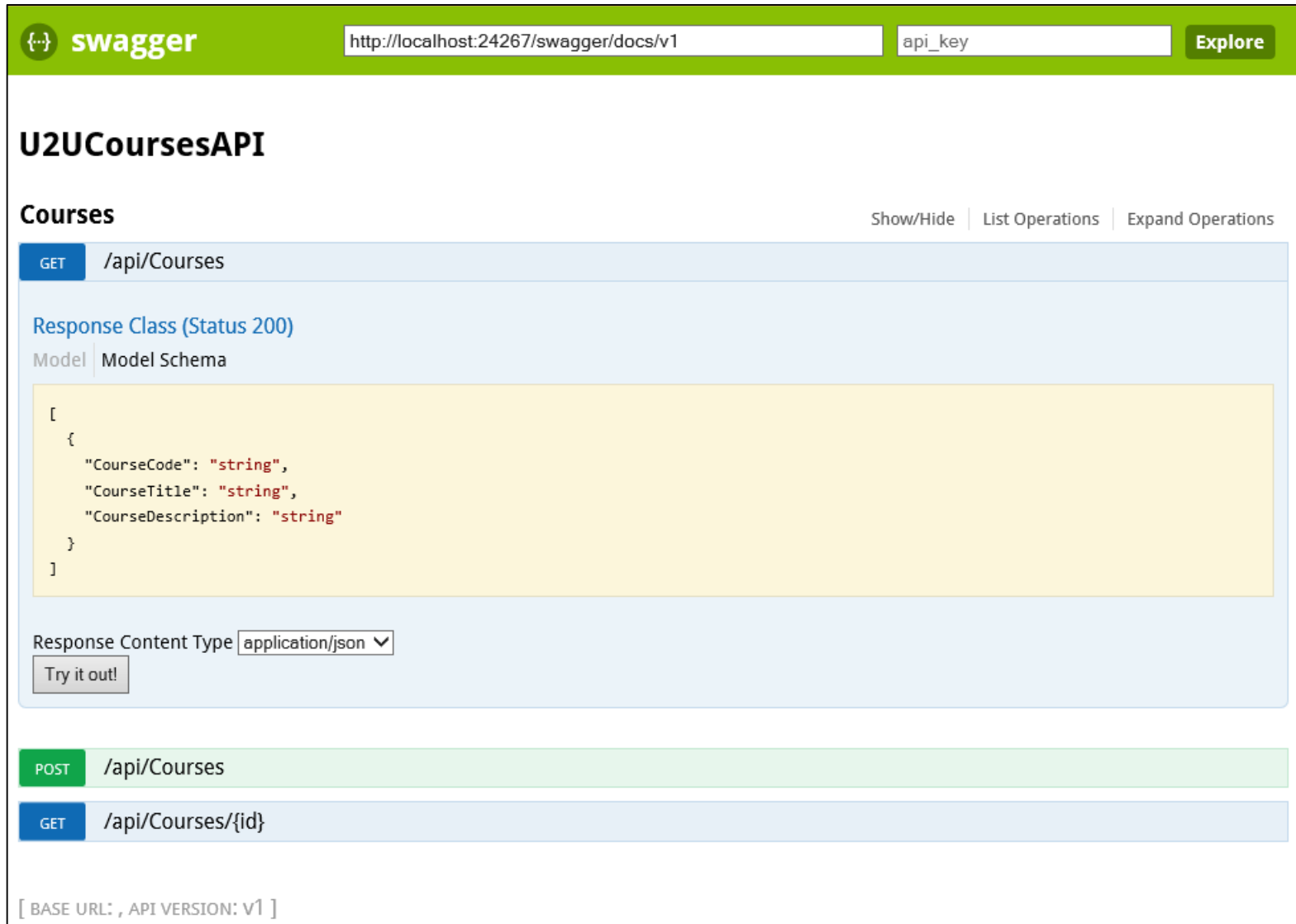
Demo

Swag apps with Swagger

- API Apps can use Swagger for testing and describing your api
 - <http://swagger.io/>
- Swagger-configuration done by App_Start/SwaggerConfig
- Many settings are possible through this configuration
 - E.g. : allow Swagger Testing UI by uncommenting EnableSwaggerUI
 - Add testing-url to your app (/swagger)



Swag apps with Swagger



The screenshot shows the Swagger UI interface for the U2UCoursesAPI. At the top, there is a green header with the Swagger logo, a text input field containing the URL `http://localhost:24267/swagger/docs/v1`, another text input field for `api_key`, and an `Explore` button. Below the header, the title **U2UCoursesAPI** is displayed. Underneath, the **Courses** section is active, with options for `Show/Hide`, `List Operations`, and `Expand Operations`. The first operation is a `GET` request to `/api/Courses`. The response class for this operation is `Status 200`. The response body is shown in a yellow box with the following JSON structure:

```
[
  {
    "CourseCode": "string",
    "CourseTitle": "string",
    "CourseDescription": "string"
  }
]
```

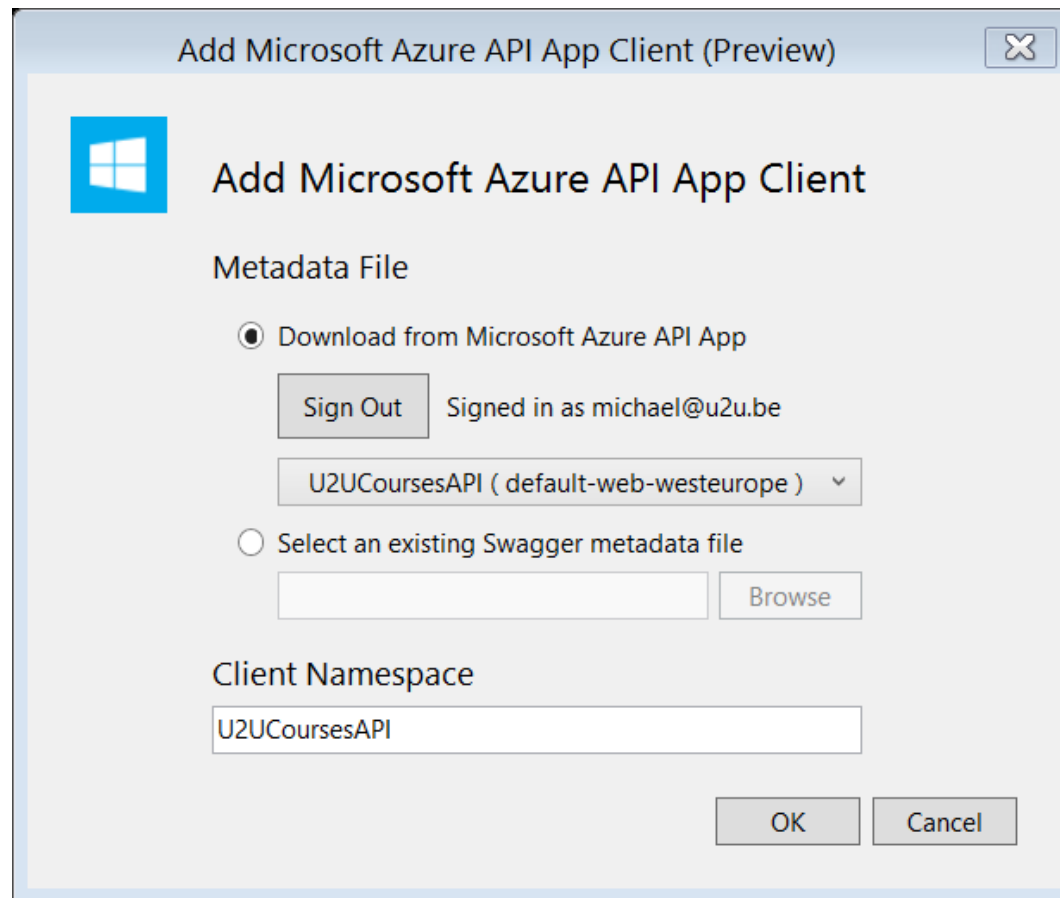
Below the response body, there is a `Response Content Type` dropdown menu set to `application/json` and a `Try it out!` button. Below this, there are two more operations listed: a `POST` request to `/api/Courses` and a `GET` request to `/api/Courses/{id}`. At the bottom left, there is a footer that reads `[BASE URL: , API VERSION: v1]`.

Demo

Creating an API Client

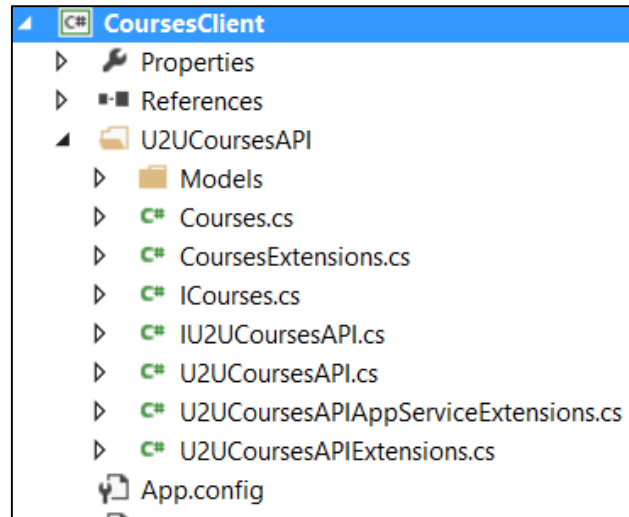


- Call "Add Azure API Client" from within your project
- Client Code is generated based on Swagger metadata



Creating an API Client

- Adds Several files to your client application based on Swagger-exposed metadata



- After that, coding is easy...

```
var client = new U2UCoursesAPI();

var courses = client.Courses.GetCourses();
foreach (var item in courses)
{
    Console.WriteLine(item.CourseCode);
}
```

Creating an API Client

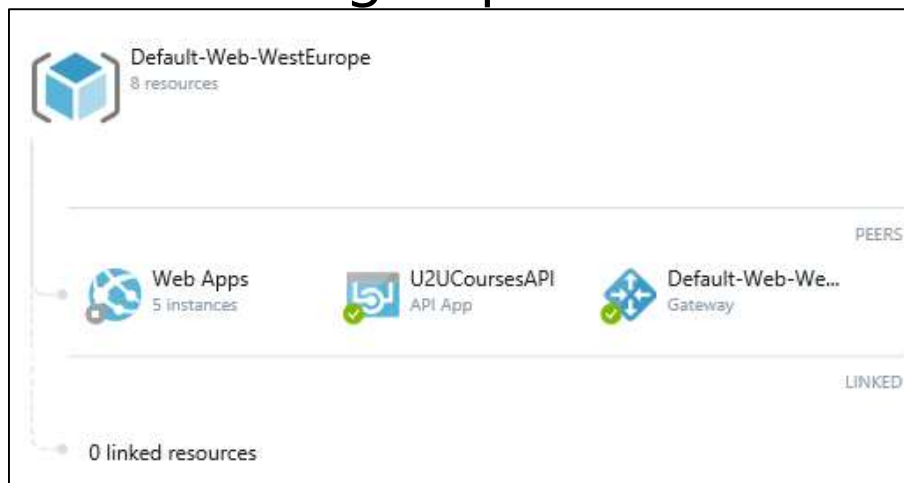


- Swagger CodeGen allows you to create Clients SDK in different languages
- API Apps can be clients of other API Apps (fits in the Microservice idea)

Demo

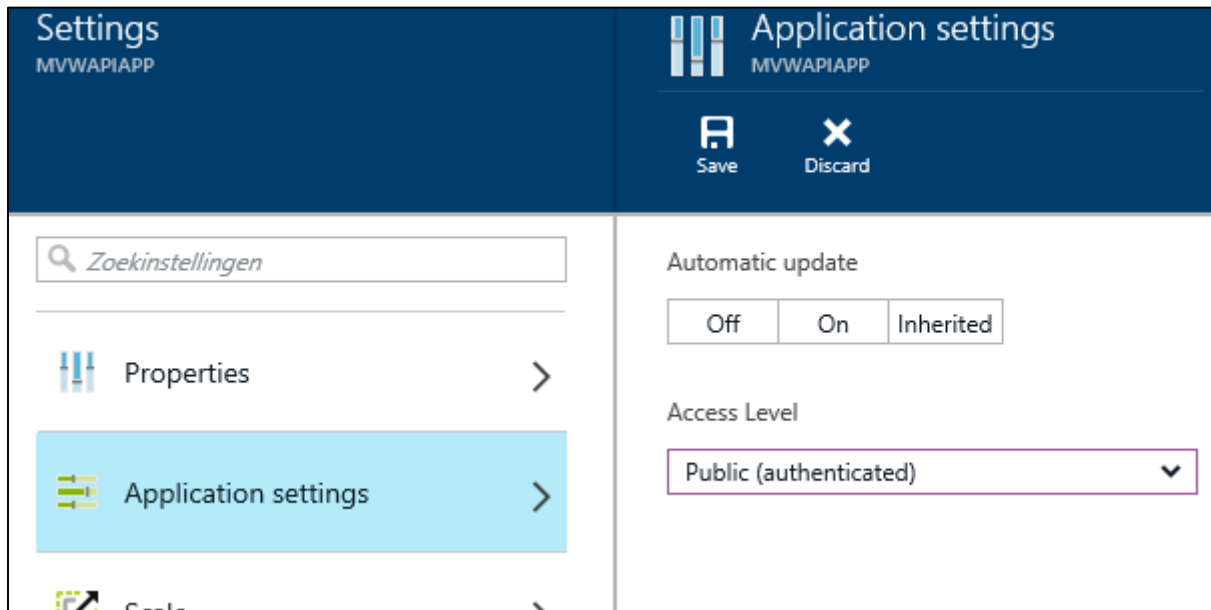
Add Authentication

- API Apps have accessibility settings
 - **Public (anonymous)** - Anyone can call the API app
 - **Public (authenticated)** - Only authenticated users are allowed to call the API app from outside the resource group
 - **Internal** - Only other API apps in the same resource group are allowed to call the API app
- All requests go through a Gateway defined on resourcegroup! Gateway chooses identityprovider. Same for all API's in resourcegroup



Add Authentication

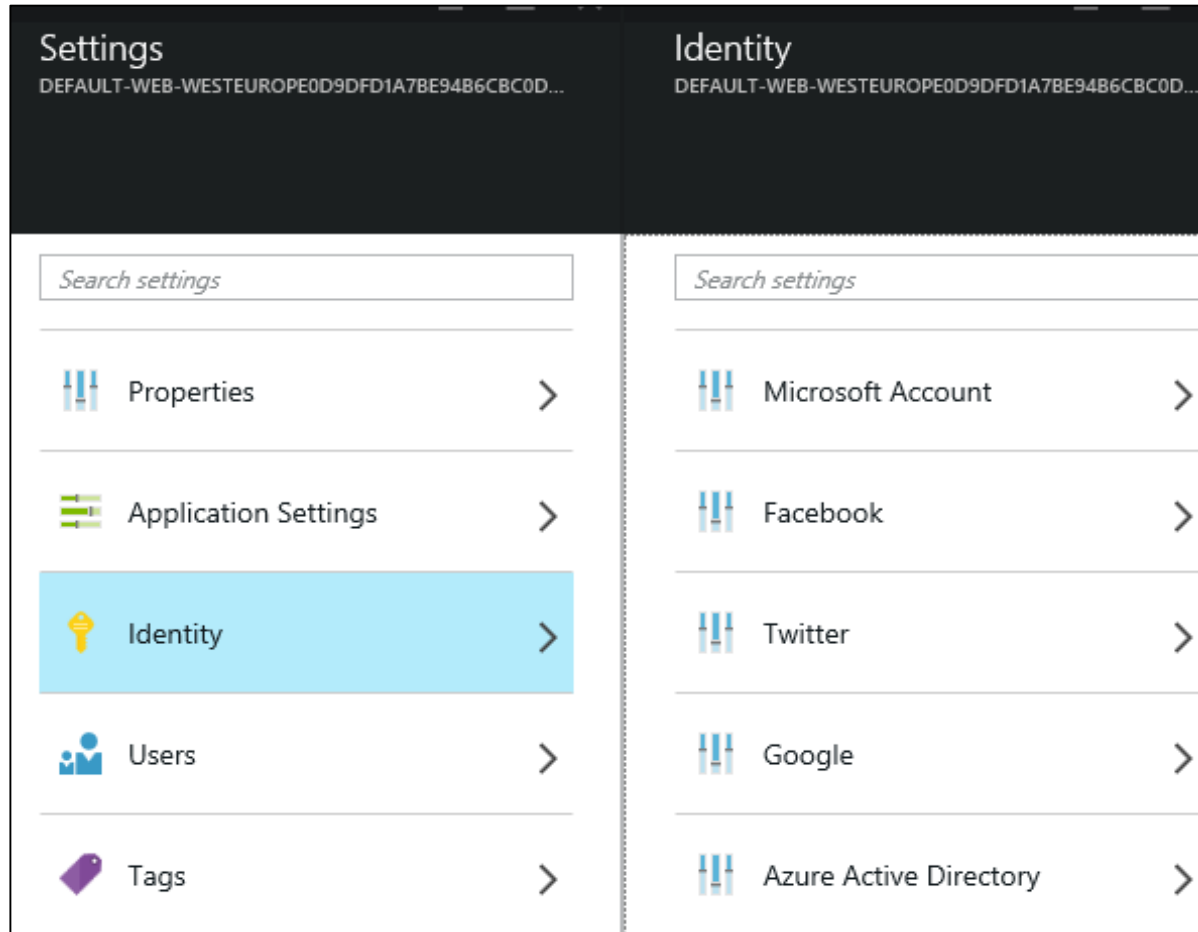
- Add authentication on API App properties



Add Authentication



- Specify IdentityProvider on Gateway-properties



Add Authentication

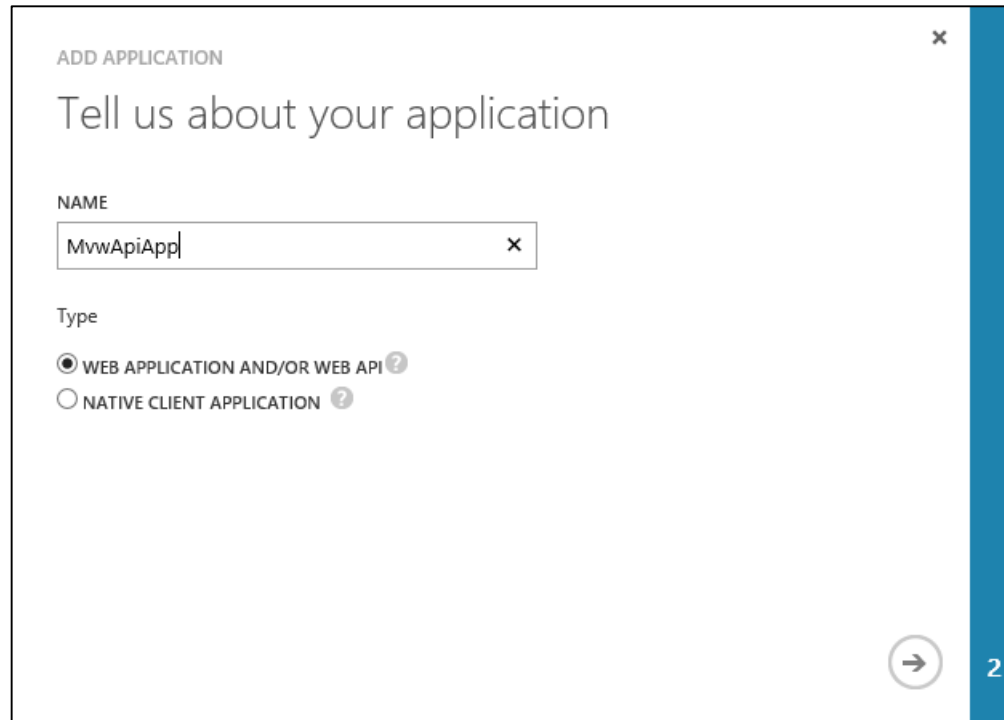


- Authentication depends on the chosen identity provider
 - You need to register an app in facebook, twitter, AAD,...
 - This gives you an App ID and App secret
 - Needed on gateway

Demo

Add Authentication - AAD

- Add an application to AAD (“Application that my organization is developing”)



ADD APPLICATION ×

Tell us about your application

NAME

MvwApiApp ×

Type

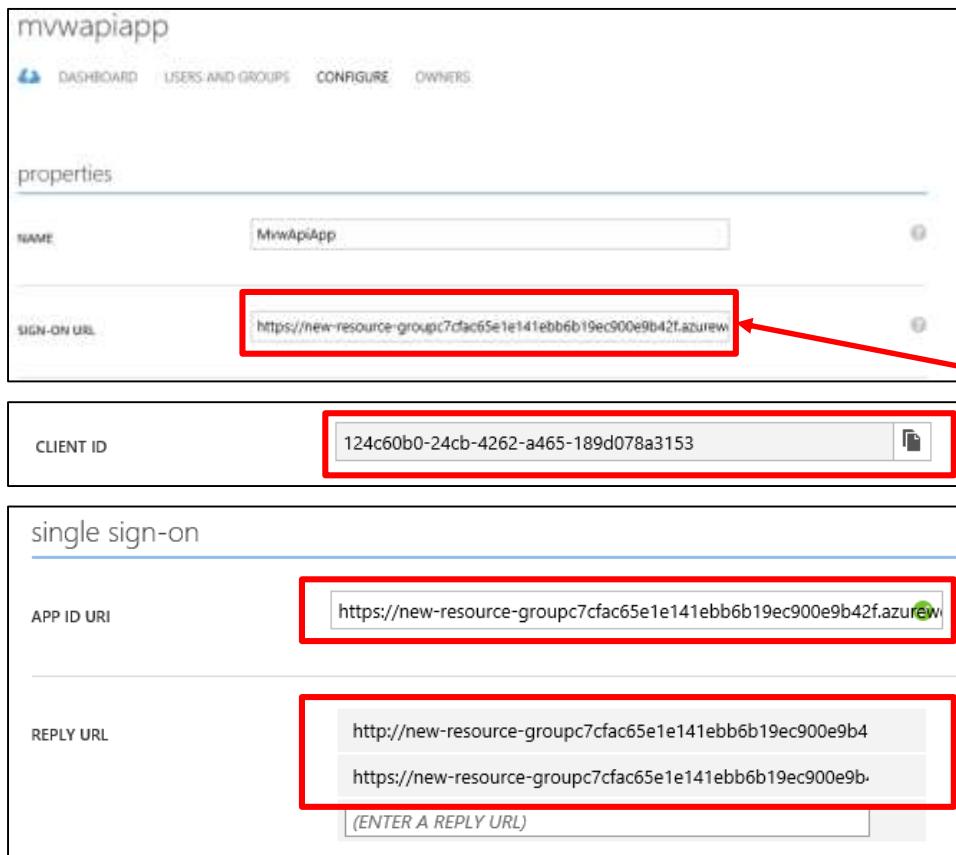
WEB APPLICATION AND/OR WEB API ?

NATIVE CLIENT APPLICATION ?

→ 2

Add Authentication

- Copy the URL's from your app's blade into AAD.
- Copy the ApplicationID to your blade



mvwapiapp

DASHBOARD USERS AND GROUPS CONFIGURE OWNERS

properties

NAME MvwApiApp

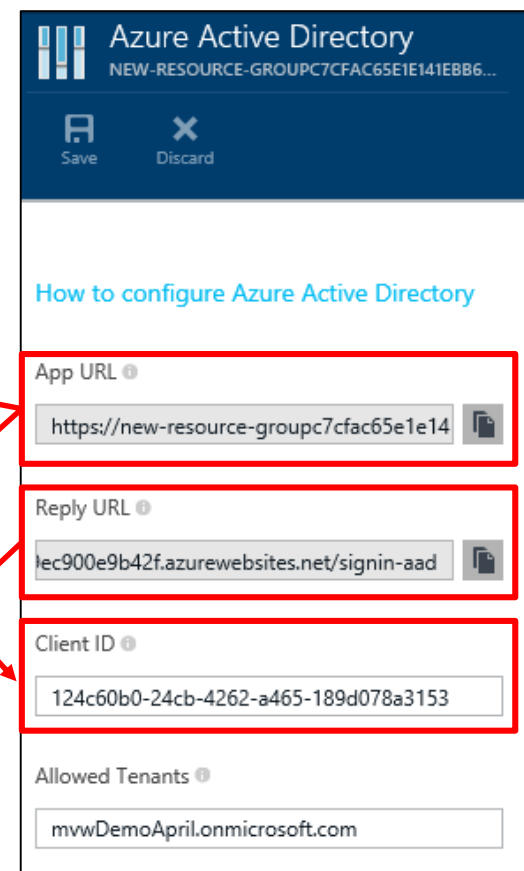
SIGN-ON URL https://new-resource-group7cfac65e1e141ebb6b19ec900e9b42f.azurew...

CLIENT ID 124c60b0-24cb-4262-a465-189d078a3153

single sign-on

APP ID URI https://new-resource-group7cfac65e1e141ebb6b19ec900e9b42f.azurew...

REPLY URL http://new-resource-group7cfac65e1e141ebb6b19ec900e9b4...
https://new-resource-group7cfac65e1e141ebb6b19ec900e9b-...
(ENTER A REPLY URL)



Azure Active Directory
NEW-RESOURCE-GROUP7CFAC65E1E141EBB6...

Save Discard

How to configure Azure Active Directory

App URL https://new-resource-group7cfac65e1e14...

Reply URL ec900e9b42f.azurewebsites.net/signin-aad

Client ID 124c60b0-24cb-4262-a465-189d078a3153

Allowed Tenants mvwDemoApril.onmicrosoft.com

Add authentication



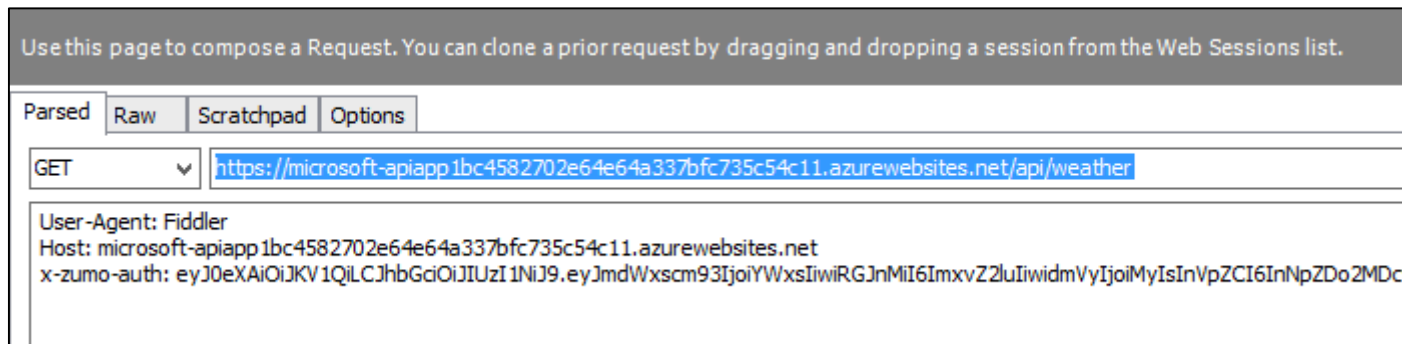
- Access your api app → HTTP 403 Forbidden
- Go to the login-url of the gateway
 - <http://gatewayurl/login/providername>
 - Providername :
 - aad
 - google
 - facebook
 - twitter
 - microsoftaccount
- Login -> Get the x-zumo-auth header from the response

Name	Value
ARRAffinity	e8cc0382dba13d11c36611
wams_nonce	cLvGoD1WJmuel_k1d4Ocd
x-zumo-auth	eyJ0eXAiOiJKV1QiLCJhbG
new-resource-groupc7fac...	
Application Cache	

Add authentication



- Use the x-zumo-auth header when calling your api



- API Client will have easy way of doing this. (not yet in preview)

Demo

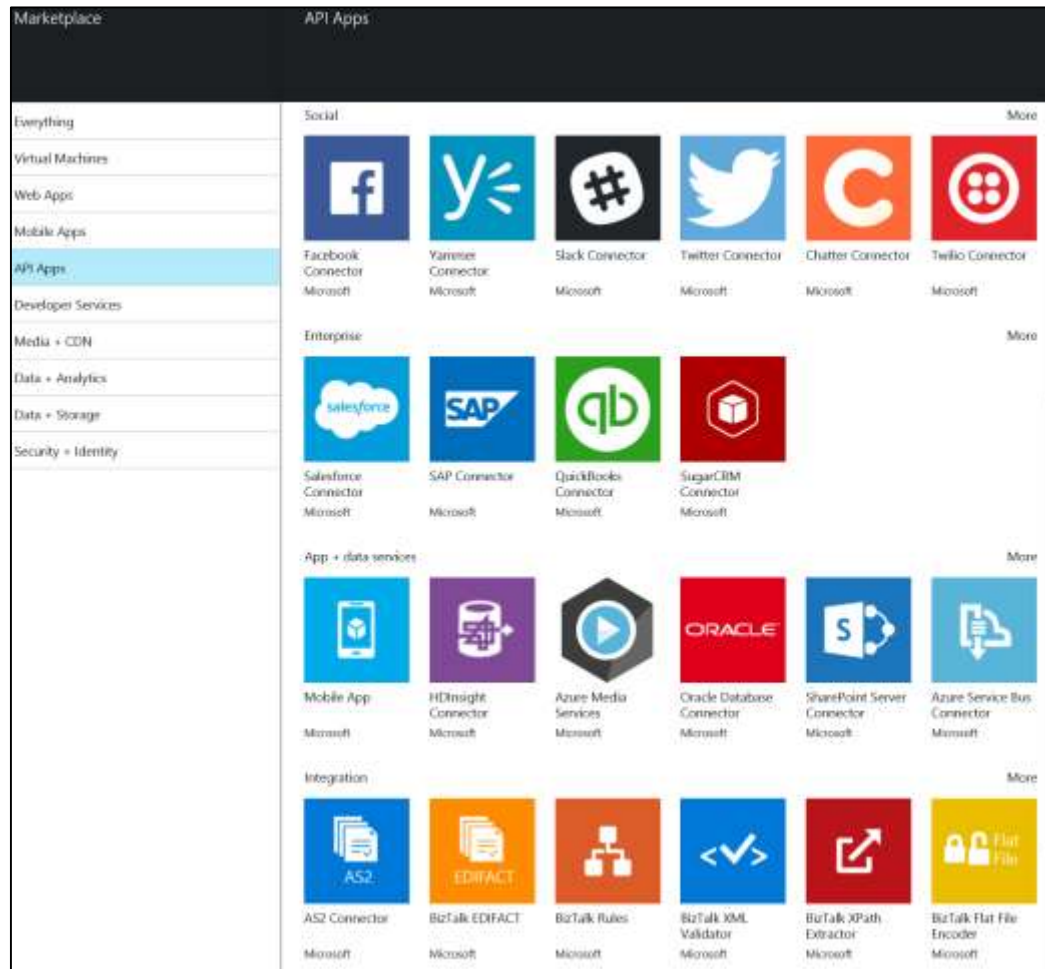
Logic Apps



- Platform as a service (PaaS)
- Allows to automate business processes by visually designing workflows in the browser
- Made by combining API apps and connectors
- Define when your process needs to run with triggers
- Use Biztalk functionality for more complex workflows

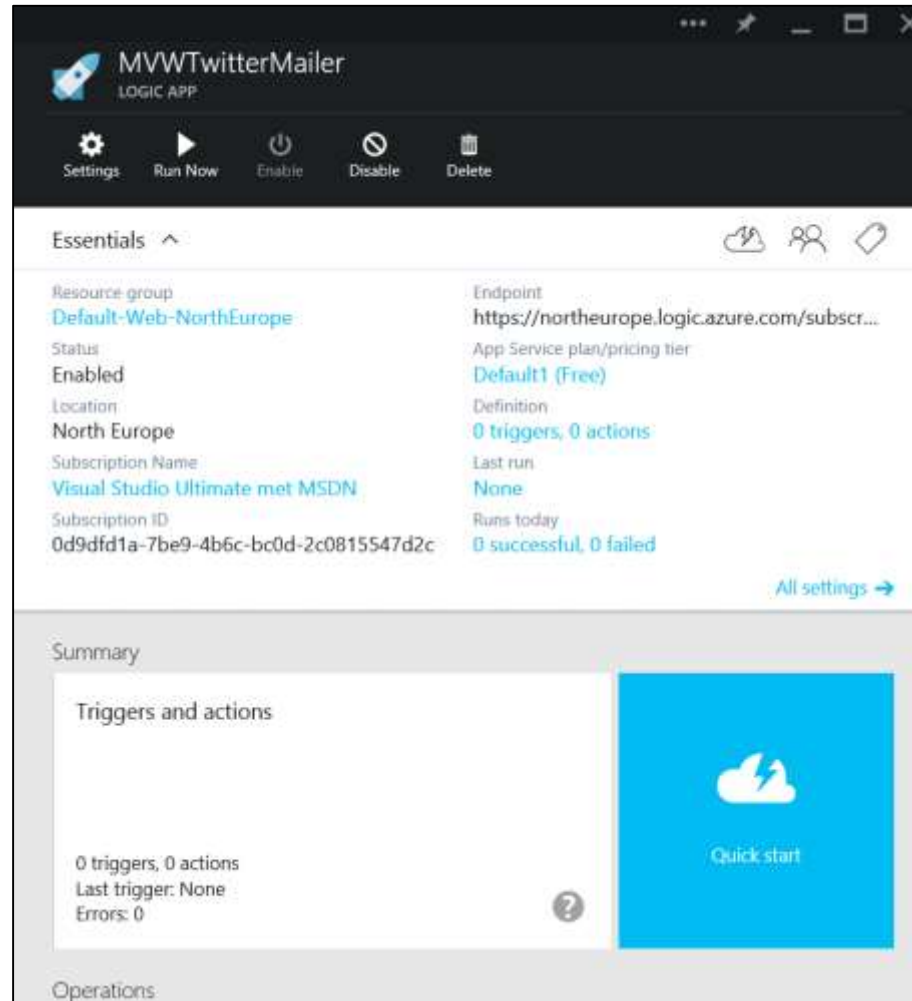
Creating Logic Apps

- Creating Connectors
 - Connectors are API apps, available in the marketplace



Creating Logic Apps

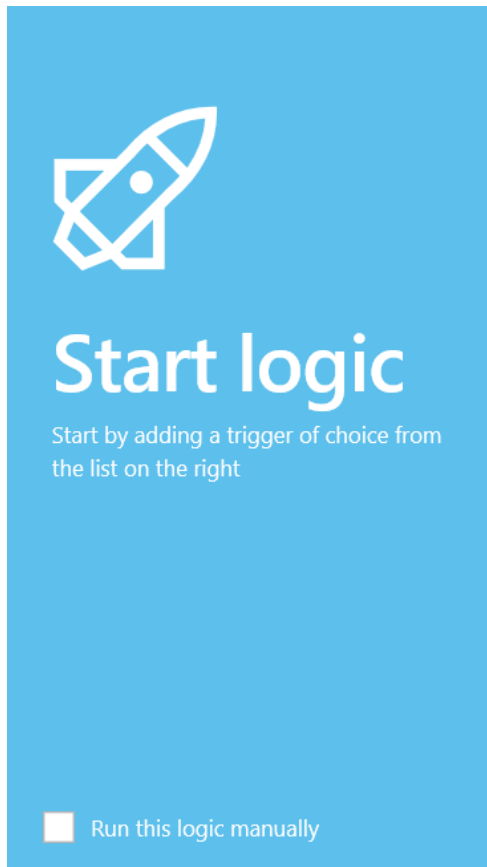
- Create Your Logic App in the browser




Creating logic apps



- Add triggers and actions
- E.g. : start the flow when a tweet about U2UTraining is detected, or start every hour, or ...

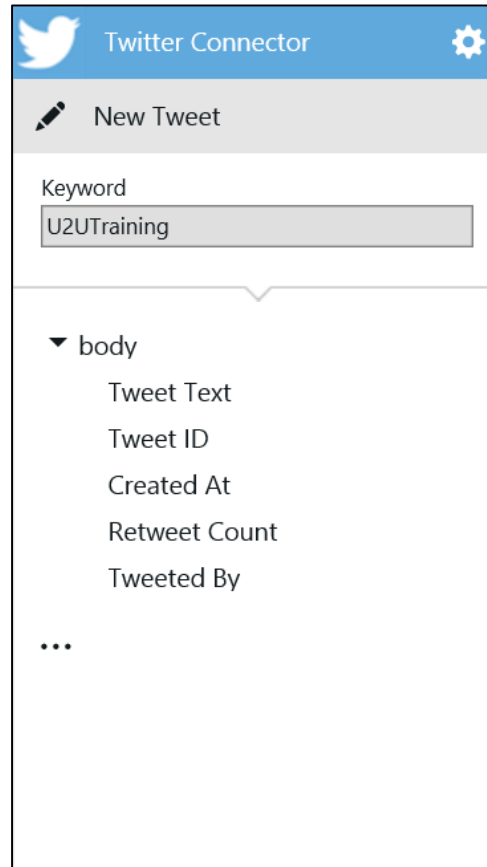







Start logic

Start by adding a trigger of choice from the list on the right

Run this logic manually



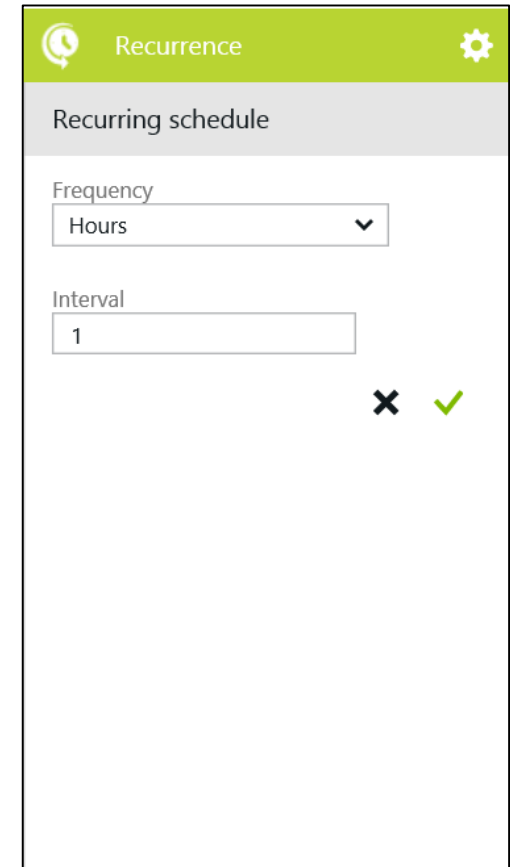
 Twitter Connector 



 New Tweet

Keyword

▼ body

- Tweet Text
- Tweet ID
- Created At
- Retweet Count
- Tweeted By
- ...





 Recurrence 

Recurring schedule

Frequency

Interval

Creating logic apps

- Connect API apps together for creating a functional whole
- E.g. : Every hour check for tweets and mail them

The image displays a Logic App workflow with three connectors:

- Recurrence** (Green header):
 - Recurring schedule
 - Frequency: Hours
 - Interval: 1
- Twitter Connector** (Blue header):
 - Search Tweets
 - Specify a query to retrieve tweets: #U2UTraining
 - body
 - Tweet Text
 - Retweet Count
 - Tweeted By
 - ...
- SMTP Connector** (Orange header):
 - Send Email
 - To: michael@u2u.be
 - Cc:
 - Subject: New Tweet about U2U Training
 - Body: @first(body('twitterconnector'), TweetText)
 - body

Demo

Using your API Apps



- Your own API Apps can be used, like the others.
- Just add them in your Logic App
- Additional Coding is needed if you want to use them as triggers in Logic apps

API App triggers



- Comparable to events
- Allows clients of API apps to respond
- Exposes the API app to Logic Apps
- Two types of triggers
 - Poll Triggers : Client polls the API app for notification of an event having been fired
 - Push Triggers : Client is notified by the API app when an event fires

Poll Triggers

- Add a GET method that will be called regularly

```
[HttpGet]
[Route("api/triggers/weatherchanged")]
public HttpResponseMessage WeatherChangedTrigger(string triggerstate,
string location)
{
    //Check if weather changed since timestamp in triggerState for location
    var hasChanged = true;
    if (hasChanged)
    {
        return Request.EventTriggered(new { info = "Sunny" },
DateTime.UtcNow.ToString(), TimeSpan.FromHours(2));
    }
    return Request.EventWaitPoll(TimeSpan.FromHours(1),
DateTime.UtcNow.ToString());
}
```

Info to pass

Triggerstate
(last time you checked)

Recommend time to
wait before next poll

Push Triggers



- More Complex
- Create a PUT method taking TriggerInput
 - Specifies returnUrl and what needs to be returned
 - Returns Registration
 - Uses a “triggerstore” you created

```
[HttpPut]
[Route("api/triggers/weatherwarning")]
public HttpResponseMessage WeatherWarningTrigger(string triggerId,
    [FromBody] TriggerInput<string, WeatherWarning> triggerInput)
{
    triggerStore.RegisterTrigger(triggerId, triggerInput);
    return this.Request.PushTriggerRegistered(triggerInput.GetCallback());
}
```

Push Triggers



- Create your TriggerStore
 - Created as singleton class
 - Collects all incoming requests
 - Contains code for monitoring events (e.g. FileSystemWatcher)
 - Invokes callback when clients need to be warned

```
internal void RegisterTrigger(string triggerId,
    TriggerInput<string, WeatherWarning> triggerInput)
{
    var weatherWatcher = new WeatherAlarm();
    weatherWatcher.WarningIssued += (s, ea) =>
    {
        Runtime runtime = Runtime.FromAppSettings();
        ClientTriggerCallback<WeatherWarning> callback =
triggerInput.GetCallback();

        callback.InvokeAsync(runtime, ea.WeatherWarning);
    };
    _store.Add(triggerId, weatherWatcher);
}
```

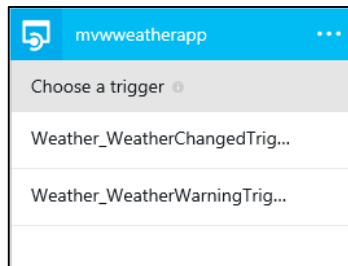

Triggers



- Your triggers show up in the API definition

OPERATION	HTTP METHOD	LOGIC APP TRIGGER
/api/triggers/weatherchanged	GET	poll
/api/triggers/weatherwarning/...	PUT	push

- And they become available for your logic apps



Demo

Summary



- API Apps allow you to build and distribute API's
- Metadata is exposed through Swagger
- API Client SDK's can be generated
- Authentication can be done with social network credentials or AAD
- Business flows can be created with Logic Apps
- Logic Apps consume API Apps
- API Apps + Logic Apps = Microservice Architecture