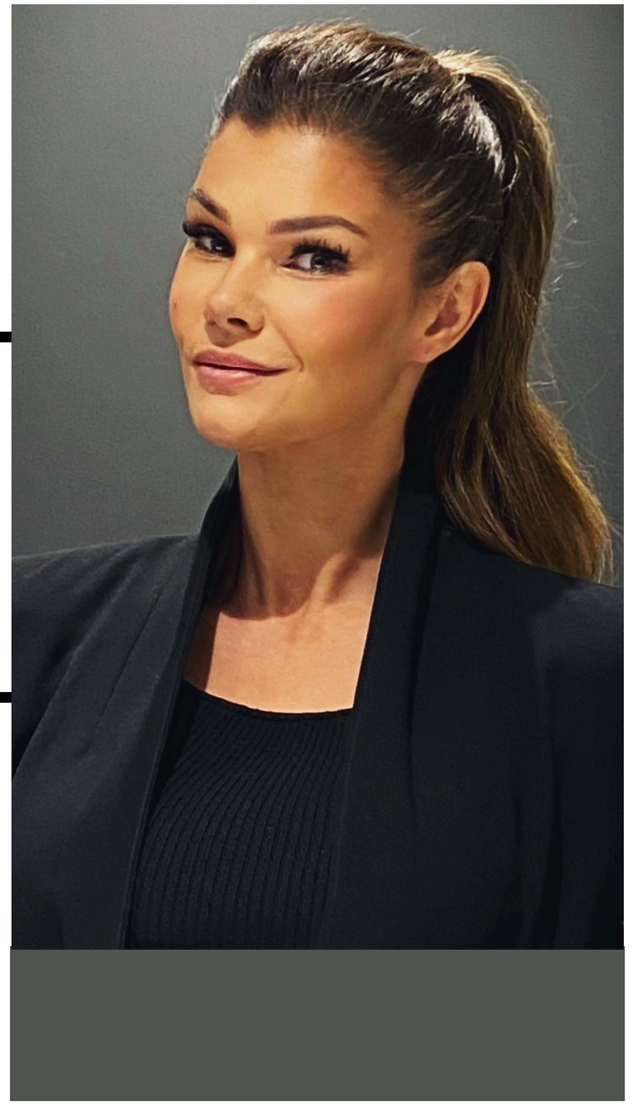
Kollokvie

AZ- 204 Azure Developer Associate

Create Serverless Applications



Monica Beate Tvedt

Teknologidirektør

Forte_ Digital

TIDLIGERE

- Agency Director - Head of Microsoft Development, Mixed Reality & Microservices at Sopra Steria
- Head of UMS Innovation Center at Unified Messaging Systems
- Global Head of SaaS Development at Unified Messaging Systems
- Senior Software Engineer Consultant, Webstep @ Sparebanken Vest
- Software Engineer, CellVision
- Gründer

PROSJEKT 2020

- Kunde: **ASKO**
  Rolle: Arkitekt og Front-end lead
- Kunde: **Kværner**
  Rolle: Arkitekt og Mobilspesialist
- Kunde: **COVID-19 Digital Feberpoliklinikk**
  Rolle**:** Løsningsarkitekt

FOREDRAG 2020

Oslo Business Forum 2020, Relevans 2020, Global AI on Tour 2020, Women in Tech 2020, Lørn.Tech.

DIVERSE NTERESSER

Alpint, tennis, programmering, tegne, lese bøker

1.0	Automating Business Processes
2.0	Azure Functions
3.0	Webhooks
4.0	SignalR
5.0	Azure API Management
-	-
6.0	Self Study

1.0 Automating Business Processes.

Business Processes in a nutshell

Modern businesses run on multiple applications and services. How well your business runs can often be impacted by how efficiently you can distribute the right data to the right task. Automating this flow of data can streamline your business even further. Choosing the right technology for these critical data integrations and process automation is also an important consideration.

Business processes modeled in software are often called **workflows**. Azure includes four different technologies that you can use to build and implement workflows that integrate multiple systems:

- Logic Apps
- Microsoft Power Automate
- WebJobs
- Azure Functions

Similarities

These four technologies have some similarities. For example:

- They can all accept **inputs**. An input is a piece of data or a file that is supplied to the workflow.
- They can all run **actions**. An action is a simple operation that the workflow executes and may often modify data or cause another action to be performed.
- They can all include **conditions**. A condition is a test, often run against an input, that may decide which action to execute next.
- They can all produce **outputs**. An output is a piece of data or a file that is created by the workflow.

In addition, workflows created with these technologies can either start based on a schedule or they can be triggered by some external event.

Design-First Technologies

Logic Apps
Logic Apps is more appropriate for use by IT professionals, developers, or DevOps practitioners. By using the design-first approach in Logic Apps, you can draw out complex workflows that model complex business processes. Alternatively, if you prefer to work in code, you can create or edit a workflow in JSON notation.
Has over ***200 Connectors*** included - possibility to create your own connectors.

Microsoft Power Automate
Microsoft Power Automate is more appropriate for use by non-technical staff. Under the hood, Microsoft Power Automate is built on Logic Apps. There are four different types of flow that you can create:

- **Automated**: A flow that is started by a trigger from some event. For example, the event could be the arrival of a new tweet or a new file being uploaded.
- **Button**: Use a button flow to run a repetitive task with a single click from your mobile device.
- **Scheduled**: A flow that executes on a regular basis such as once a week, on a specific date, or after 10 hours.
- **Business process**: A flow that models a business process such as the stock ordering process or the complaints procedure. The flow process can have: notification to required people; with their approval recorded; calendar dates for steps; and recorded time of flow steps.

Design-First Technologies Compared

	Microsoft Power Automate	Logic Apps
Intended users	Office workers and business analysts	Developers and IT pros
Intended scenarios	Self-service workflow creation	Advanced integration projects
Design tools	GUI only. Browser and mobile app	Browser and Visual Studio designer. Code editing is possible
Application Lifecycle Management	Power Automate includes testing and production environments	Logic Apps source code can be included in Azure DevOps and source code management systems

Code-First Technologies

WebJobs and the WebJobs SDK

Part of the ***Azure App Service*** that you can use to run a program or script automatically. Choose this if you want the code to be a part of an existing App Service application and to be managed as part of that application, for example in the same Azure DevOps environment. WebJobs are also the only technology that permits developers to control ***retry policies***. There are two kinds of WebJob:

- **Continuous.** These WebJobs run in a continuous loop. For example, you could use a continuous WebJob to check a shared folder for a new photo.
- **Triggered.** These WebJobs run when you manually start them or on a schedule. For example, when a user uploads a new profile picture, you may need to generate a smaller thumbnail photograph.

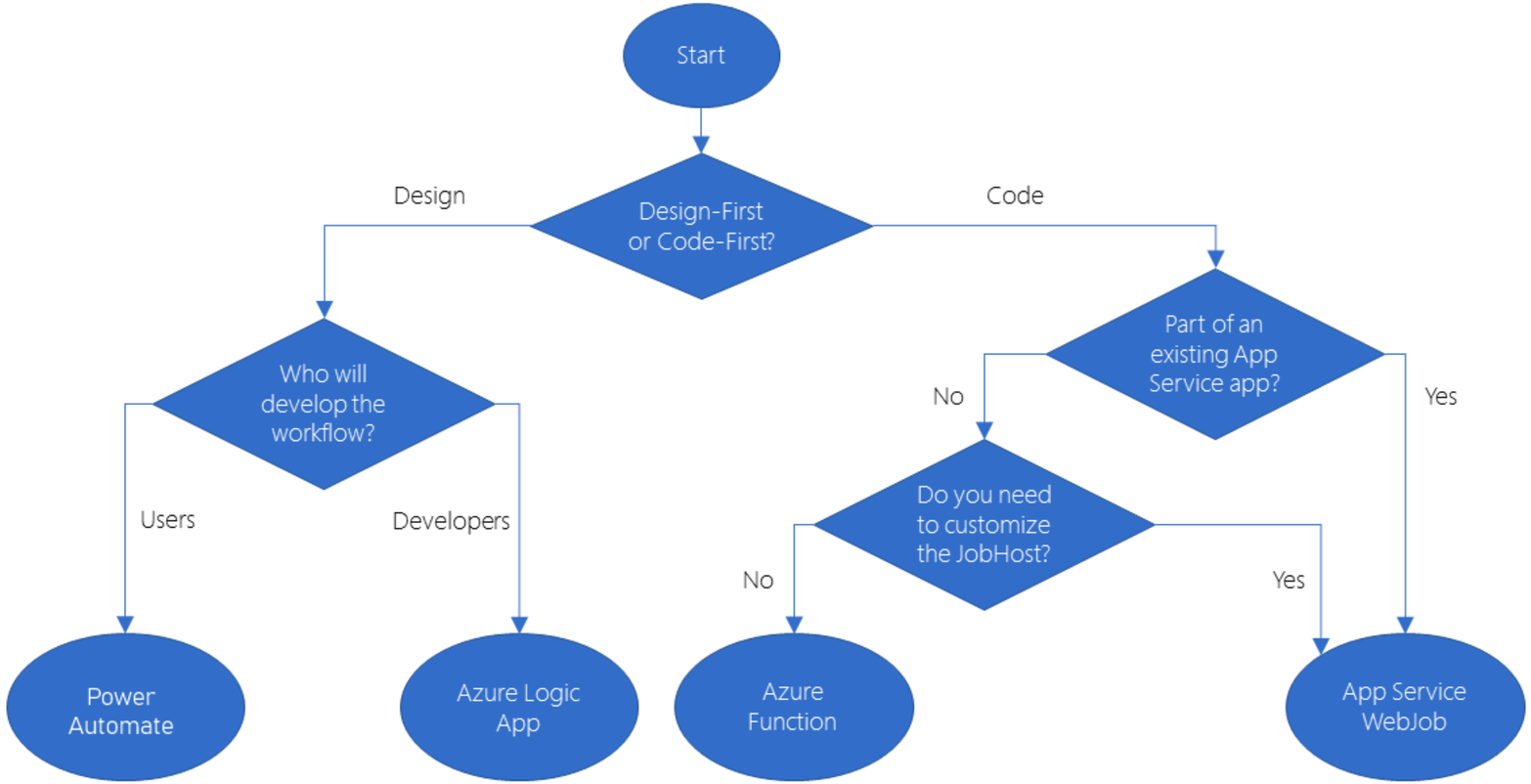
Azure Functions

An Azure Function is a simple way for you to run small pieces of code in the cloud, without having to worry about the infrastructure required to host that code. In addition, with the consumption plan option, ***you only pay for the time when the code runs***. Azure ***automatically scales*** your function in response to the demand from users. Azure Functions can integrate with many different services both within Azure and from third parties. These services can trigger your function, or send data input to your function, or receive data output from your function.

Code-First Technologies Compared

	Azure WebJobs	**Azure Functions**
Supported languages	C# if you are using the WebJobs SDK	C#, Java, JavaScript, PowerShell, etc.
Automatic scaling	No	Yes
Development and testing in a browser	No	Yes
Pay-per-use pricing	No	Yes
Integration with Logic Apps	No	Yes
Package managers	NuGet if you are using the WebJobs SDK	Nuget and NPM
Can be part of an App Service application	Yes	No
Provides close control of `JobHost`	Yes	No

How to choose a service - simplified



2.0
Azure Functions.

Azure Functions in a nutshell

Azure Functions lets you develop serverless applications on Azure. A solution for easily running small pieces of code, or “functions”, without worrying about a whole application.

- Can use your language of choice, such as ***C#, F#, Node.js, Java, or PHP***.
- Pay only for the time your code runs
- Azure scales as needed.

What can we use Azure Functions for?

Azure Functions is great when working with **IoT**, **simple API**s and for **processing data**. Consider Functions for tasks like image or order processing, file maintenance, or for any tasks that you want to run on a schedule.

Functions supports ***triggers***, which are ways to start execution of your code, and ***bindings***, which are ways to simplify coding for input and output data.

Azure Functions provide the following templates

- **HTTPTrigger**, Trigger via HTTP requests
- **TimeTrigger**, Scheduled tasks
- **Webhooks,** GitHub repository events
- **CosmoDBTrigger**, Process Cosmos DB docs when they are added or updated in collections in a NoSQL database.
- **BlobTrigger**, Process Storage blobs when they are added to containers.
- **QueueTrigger**, Respond to messages as they arrive in Storage queue.
- **EventHubTrigger**, Respond to events delivered to the Event Hub. Useful in workflow processing and IoT scenarios.
- **ServiceBusQueueTrigger**, Connect to other services each time a message is added to the service bus.
- **ServiceBusTopicTrigger**, Connect to other services by subscribing to topics.

2.1 Case Discussion

We have an .NET core web app which creates and schedules classes.
When a class is over we want to send an email to all students asking them to rate their teacher.

How could we achieve this using Azure Functions and which trigger would you use?

Azure Functions Core Concepts

- Defining input and output bindings:
  - Function.json
  - Attributes
- A project (Function App) can contain multiple functions.
- Each function can have different triggers.
- One function must have exactly ***one trigger***.
- One function can have ***multiple input and output bindings***.

2.2 Task

Send me a DB describing what the following Azure

Function does:

FUNCTION.JSON

```
{ "bindings": [{
     "name": "order",
     "type": "queueTrigger",
     "direction": "in",
     "queueName": "myqueue-items",
     "connection": "MY_STORAGE_KEY"
},{
     "name": "$return",
     "type": "table",
     "direction": "out",
     "tableName": "outTable",

    "connection": " MY_TABLE_STORAGE_KEY”}]}
```

FUNCTION.CS

```
public static Person Run(JObject order, TraceWriter log) {
     return new Person() {
          PartitionKey = "Orders",
          RowKey = Guid.NewGuid().ToString(),
          Name = order["Name"].ToString(),
          MobileNumber = order["MobileNumber"].ToString() };
 }
```

2.3 Task - Attributes

Write the same trigger and bindings by attributes instead of using a function.json file.

2.4 Solution

```
{
    [FunctionName("QueueTriggerTableOutput")]
    [return: Table("outTable", Connection = “MY_TABLE_STORAGE_KEY")]
    public static Person Run(
        [QueueTrigger("myqueue-items", Connection = “MY_STORAGE_KEY”)]JObject order,
        ILogger log)
    {
        return new Person() {
                PartitionKey = "Orders",
                RowKey = Guid.NewGuid().ToString(),
                Name = order["Name"].ToString(),
                MobileNumber = order["MobileNumber"].ToString() };
    }
}
```

Time Trigger - Running Scheduled tasks

A timer trigger lets you execute a function at a set interval
using CRON Expressions.

CRON:
{second} {minute} {hour} {day} {month} {day-of-week}

"schedule": "0 */5 * * * *",
"name": "myTimer",
"type": "timerTrigger",
"direction": "in"

The following example shows a function that is executed once every five minutes (eg if the function starts at 18:57:00, the next performance will be at 19:00:00):

```
[FunctionName("TimerTriggerCSharp")]
public static void Run(
    [TimerTrigger("0 */5 * * * *")]TimerInfo myTimer,
    ILogger log)
{
     if(myTimer.IsPastDue)
     {
          log.LogInformation("Timer is running late!");
     }

     log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");
}
```

2.4 Task - Durable Functions

Send me a DM describing what a durable function is and two Durable Functions patterns.

3.0 Webhooks.

Webhooks in a nutshell

Webhooks offer a lightweight mechanism for your app to be notified by another service when something of interest happens, ie. when a wiki-page on GitHub is updated. Webhooks can be used as a trigger for your Azure Function.

Webhooks are user-defined HTTP callbacks. They're triggered by some event, such as pushing code to a repo or updating a wiki page. When the event occurs, the source site makes an HTTP request to the URL configured for the webhook (your Azure Function URL). With Azure Functions, we can define logic in a function that can be run when a webhook message is received.

4.0 SignalR.

Enable automatic updates in a web application using Azure Functions and SignalR Service.

Use Case: Web application that reports stock information by fetching changes from the server based on a timer. In this case, an Azure Function with a httpTrigger, where the client uses a timer to send a http-request every five seconds.

```
//JavaScript
setInterval(this.update, 5000);
```

Goal: Update the app's notification mechanism from polling to push-based architecture with SignalR Service, Azure Cosmos DB and Azure Functions.

But why do this?
Analysis of current solution

Drawbacks:

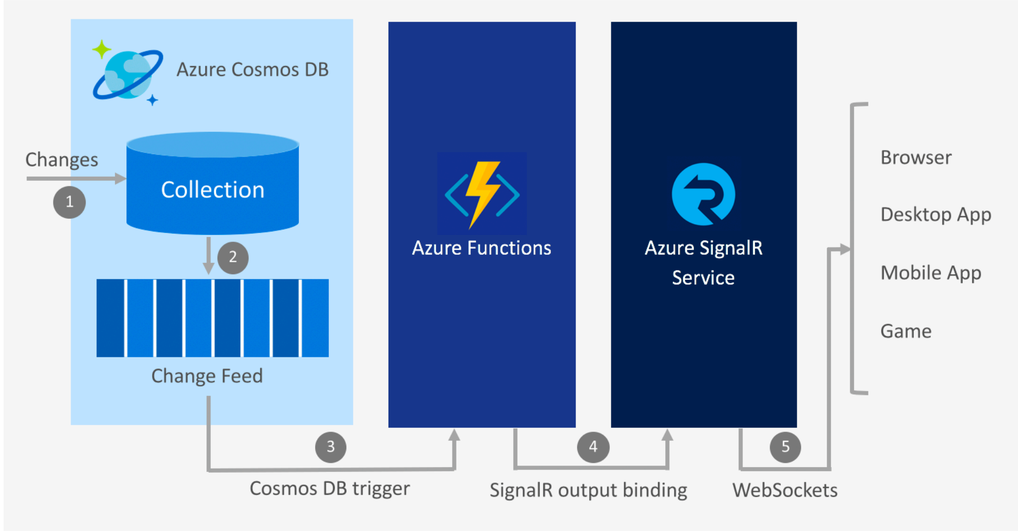
- The client application contacts the server whether or not changes exist to the underlying data.
- Once data is returned from the server the entire list of stocks is updated on the web page, again, regardless of any changes in the data.
- Delays also often exist between when new data becomes available and when it's detected by the app.
- HTTP request headers includes hundreds of bytes of data along with the session's cookie. All this overhead, especially when under heavy load, creates wasted resources and unnecessarily taxes the server.

Gains -
Analysis of our new solution

Azure Cosmos DB exposes a "change feed". It outputs a sorted list of changed documents in the order in which they were modified. Using the Azure Cosmos DB trigger automatically makes our data retrieval more efficient.

Azure Functions features a binding that runs code anytime data is updated in an Azure Cosmos DB change feed.
When paired with a persistent connection to the client (SignalR), the function can contact individual clients on-demand, which is the foundation for a real-time application architecture.

SignalR features persistent connections between the client and server, which allows the server to push data to the client at will. This reduces network traffic and load on the server. Clients can connect to the service using a SignalR client SDK that is available in .NET, JavaScript, and Java.



Example:
Azure Function before/after

//Pulling - return the entire dataset

```
module.exports = async function (context, req, stocks) {
   context.res.body = stocks;
};
```

//Real-time - return only updated stocks

```
module.exports = async function (context, documents) {
    const updates = documents.map(stock => ({
        target: 'updated',
        arguments: [stock]
    }));

    context.bindings.signalRMessages = updates;
    context.done();
}
```

Example:
Bindings before/after

//Pulling - httpTrigger in, CosmosDB source in, http response out

```
{
  "bindings": [
    {
      "type": "httpTrigger",
      "authLevel": "anonymous",
      "direction": "in",
      "name": "req",
      "methods": ["get"]
    },
    {
      "type": "http",
      "direction": "out",
      "name": "res"
    },
    {
      "type": "cosmosDB",
      "direction": "in",
      "name": "stocks",
      "ConnectionStringSetting": "AzureCosmosDBConnectionString",
      "databaseName": "stocksdb",
      "collectionName": "stocks"
    }
  ]
}
```

//Real-time - Cosmos DB trigger in, SignalR out

```
{
  "type": "cosmosDBTrigger",
  "name": "documents",
  "direction": "in",
  "leaseCollectionName": "leases",
  "connectionStringSetting": "AzureCosmosDBConnectionString",
  "databaseName": "stocksdb",
  "collectionName": "stocks",
  "createLeaseCollectionIfNotExists": "true",
  "feedPollDelay": 500
}
```

```
{
  "type": "signalR",
  "name": "signalRMessages",
  "connectionString": "AzureSignalRConnectionString",
  "hubName": "stocks",
  "direction": "out"
}
```

4.1 - Task

Let’s create ourselves a Azure Function with a timed trigger in the Azure Portal..

5.0
Azure API Management.

APIM in a nutshell

You can use Azure Functions and Azure API Management to build complete APIs with a microservices architecture. It enables you to implement Azure Functions as serverless components.
We use API Management to assemble these microservices into a single API product at a single URL with consistent behavior imposed by using API Management policies.

Azure API Management (APIM) is a fully managed cloud service that you can use to publish, secure, transform, maintain, and monitor APIs. It also handles all the tasks involved in mediating API calls, including request authentication and authorization, rate limit and quota enforcement, request and response transformation, logging and tracing, and API version management.

Which challenges are we solving?

Microservices architectures can present challenges, such as

- Client apps are coupled to microservices. If you want to ***change the location*** or ***definition*** of the microservice, you may have to reconfigure or update the client app.
- Each microservice may be presented under ***different domain names*** or IP addresses. This presentation can give an impression of inconsistency to users and can negatively affect your branding.
- It can be difficult to enforce ***consistent API rules*** and standards across all microservices. For example, one team may prefer to respond with XML and another may prefer JSON.
- You're reliant on individual teams to implement ***security*** in their microservice correctly. It's difficult to impose these requirements centrally.

How does API Management help?

Composing an API using API Management has advantages that include

- Client apps are ***coupled to the API Management*** expressing business logic, not the underlying technical implementation with individual microservices. You can change the location and definition of the services without necessarily reconfiguring or updating the client apps.
- API Management acts as an intermediary. It forwards requests to the right microservice, wherever it is located, and returns responses to users. Users ***never see the different URIs*** where microservices are hosted.
- You can use API Management policies to ***enforce consistent rules*** on all microservices in the product. For example, you can transform all XML responses into JSON, if that is your preferred format.
- Policies also enable you to ***enforce consistent security requirements***.

5.1 - Task

A. Let’s expose our Azure Function app as a consistent API by using Azure API Management.
B. Test our new endpoint and review our security settings.

6.0
Self Study

Microsoft Learn - Create Serverless Applications

https://docs.microsoft.com/en-us/learn/paths/create-serverless-applications/

Thank you.