

# IoT Cloud

Mathias Dufoort

# Inhoudsopgave

## Contents

1	Cloud computing.....	8
1.1	Enkele eigenschappen.....	8
1.2	On premise (eigen servers) .....	8
1.3	Hybrid Cloud (On Premise & Cloud).....	9
1.4	IaaS: Infrastructure as a Service .....	9
1.4.1	Voorbeelden .....	9
1.5	PaaS: Platform as a Service .....	9
1.5.1	Talen .....	9
1.5.2	Voorbeelden .....	10
1.6	SaaS .....	10
1.6.1	Voorbeelden .....	10
1.7	Belangrijkste vendors.....	10
2	Microsoft Azure .....	11
2.1	Azure Portal.....	11
2.2	Azure subscription.....	11
2.3	Resource Group.....	11
2.4	Azure Virtual Machines.....	12
2.4.1	Waarom? .....	12
2.4.2	Maintenance .....	12
2.5	Azure Web App .....	12
2.5.1	Free .....	13
2.5.2	Shared.....	13
2.5.3	Basic, Premium .....	13
2.6	Scaling .....	14
2.6.1	Scale Up .....	14
2.6.2	Scale Out.....	14
2.7	Azure SQL.....	14
2.7.1	SQL Server Cloud Based .....	14
2.7.2	Security .....	15
2.7.3	Azure DTU .....	15
2.7.4	Throttling .....	15
2.7.5	SQL via C# .....	16
2.7.6	Resources.....	16
2.7.7	Azure Database for MySQL.....	16
2.8	Azure & Internet of Things .....	16

2.8.1	Waarom is Azure belangrijk voor IoT? .....	16
2.9	Azure CLI.....	17
2.9.1	Azure Portal .....	17
2.9.2	Oplossing: Azure CLI 2.0.....	17
2.10	Bash/Powershell.....	17
2.11	Andere Azure onderdelen.....	18
2.12	Samenvatting .....	18
3	Back-end services met Azure Functions .....	18
3.1	The Internet Of .....	18
3.1.1	The Internet of Information.....	18
3.1.2	The Internet of Services.....	19
3.1.3	The Internet of Things.....	19
3.1.4	Volgende stap: The Internet of Value .....	20
3.2	Herhaling: Hoe werkt HTTP? .....	21
3.2.1	Wat is HTTP?.....	21
3.2.2	HTTP Verbs.....	21
3.2.3	HTTP status code .....	22
3.2.4	HTTPS.....	22
3.2.5	HTTP Request.....	22
3.2.6	HTTP Response .....	22
3.2.7	HTTP/2 .....	23
3.3	Webservices .....	23
3.3.1	Open data .....	23
3.3.2	Cognitive services .....	23
3.3.3	Internet of Things .....	24
3.3.4	JSON .....	24
3.3.5	URL.....	24
3.3.6	Fouten.....	25
3.3.7	Samevatting .....	25
3.3.8	Hoe maken we webservices .....	26
3.4	Azure Functions.....	26
3.4.1	Waarom? .....	26
3.5	Azure Functions Security.....	26
3.5.1	3 soorten security .....	26
3.5.2	Voordelen van key security.....	27
3.5.3	Nadelen van Key Security .....	27
3.6	Andere Azure Functions.....	27
3.7	Whats next .....	28

3.8	Azure met Raspberry Pi.....	28
3.8.1	4 scenarios' .....	28
3.8.2	GET.....	28
3.8.3	POST.....	29
3.9	Azure via .NET .....	29
3.10	Samenvatting .....	29
4	Azure Storage .....	30
4.1	Soorten storage.....	30
4.2	Azure Storage account .....	30
4.3	Azure Files .....	30
4.4	Azure Disk Storage .....	31
4.5	Azure Blob Storage.....	31
4.5.1	Pricing .....	32
4.5.2	Hot Access.....	32
4.5.3	Cool Acces.....	32
4.5.4	Static website.....	32
4.6	Azure Storage Queues.....	33
4.6.1	Voorbeeld .....	33
4.6.2	Load leveling .....	33
4.6.3	Load balancing .....	34
4.6.4	Temporal devoupling.....	34
4.7	Azure table storage .....	34
4.7.1	Entities (rijen) .....	35
4.7.2	Table Storage Data Access .....	36
4.7.3	Queries.....	36
4.7.4	Mogelijke kolom types:.....	36
4.7.5	Waar gebruiken? .....	36
4.8	Azure Storage Tools .....	36
4.9	Programmeren van Azure Storage.....	36
4.9.1	Wanneer? .....	36
4.10	Enekele scenarios' (meer info: zie labo's) .....	37
4.10.1	Scenario 1: Post binary file naar Azure Functions en opslage in Azure Blob.....	37
4.10.2	Scenario 2: Stuur e-mail wanneer een blob verschijnt op Azure Blob .....	38
4.10.3	Scenario 3: We vullen een Azure Storage Queue voor verwerking door Azure Functions .....	39
4.11	Good practices .....	41
4.12	Configuratiefiles lokaal of in de cloud? .....	41
4.13	Samenvatting .....	41
5	Azure Security.....	42

5.1	Governance .....	42
5.1.1	Cost management.....	42
5.1.2	Security .....	42
5.1.3	Resource consistency.....	42
5.1.4	Identity Baseline .....	42
5.1.5	Deployment Acceleration .....	42
5.1.6	Azure Blue Prints.....	42
5.2	Azure KeyVault .....	42
5.2.1	Probleem .....	42
5.2.2	Azure KeyVault.....	43
5.2.3	Uitlezen van Azure KeyVault.....	43
5.3	Azure Active Directory (AzureAD).....	44
5.3.1	Probleem .....	44
5.3.2	Azure AD .....	44
5.4	Azure Managed Identity.....	45
5.4.1	System Assigned Managed Identity.....	45
5.5	User Assigned Managed Identity .....	45
5.5.1	Hoe implementeren.....	45
5.6	Samenvatting .....	45
6	Cosmos DB .....	45
6.1	Relationele databases .....	45
6.1.1	Waarom ontstaan?.....	46
6.2	NoSQL databases.....	46
6.2.1	Waarom?.....	46
6.2.2	Scaling .....	46
6.2.3	Availability and always-on .....	47
6.2.4	Global deployment .....	48
6.2.5	Gedistribueerde systemen.....	48
6.3	Soorten NoSQL Databases .....	48
6.3.1	Key/Value.....	48
6.3.2	Document databases .....	49
6.3.3	Column store .....	50
6.3.4	Graph Database .....	50
6.4	Normalisatie .....	51
6.4.1	Voorbeeld SQL .....	51
6.4.2	Voorbeeld NoSQL.....	51
6.5	CosmosDB .....	52
6.5.1	CosmosDB API.....	53

6.5.2	cosmosDB API Account .....	54
6.5.3	Kiezen van een API.....	54
6.5.4	Container toevoegen .....	55
6.5.5	Manueel data toevoegen.....	55
6.5.6	Firewall .....	56
6.5.7	Aanspreken vanuit Azure Functions met .NET.....	56
6.5.8	CosmosDB extra's .....	58
6.6	Extra's.....	58
7	Azure Service bus.....	59
7.1	Azure Service Bus .....	59
7.1.1	Queues.....	59
7.1.2	Topics.....	59
7.1.3	Namespaces.....	59
7.1.4	Implementeren? .....	60
7.1.5	Verbinden? .....	60
7.2	Azure Event Hubs .....	60
7.3	Azure Data Explorer .....	61
7.4	Azure Event Grid .....	61
7.4.1	Event source .....	62
7.5	Azure Logic Apps .....	62
7.5.1	Activeren?.....	62
7.6	Azure Messaging Services .....	62
7.6.1	Event vs Message.....	62
7.6.2	Azure event Grid .....	63
7.6.3	Azure Event Hubs.....	63
7.6.4	Azure Service Bus.....	63
7.6.5	Logic Apps .....	63
7.7	Samenvatting .....	63
8	IoT Hub .....	63
8.1	Werking .....	64
8.1.1	Wat als device niet krachtig genoeg is?.....	64
8.2	Hoe data verwerken die binnenkomt op IoT Hub?.....	65
8.3	IoT Hub Device Twin.....	65
8.3.1	Reporting properties.....	66
8.3.2	Direct methods .....	66
8.4	IoT Hub Trigger voor Azure Functions.....	67
8.5	Ondersteuning .....	67
8.5.1	SDK's .....	67

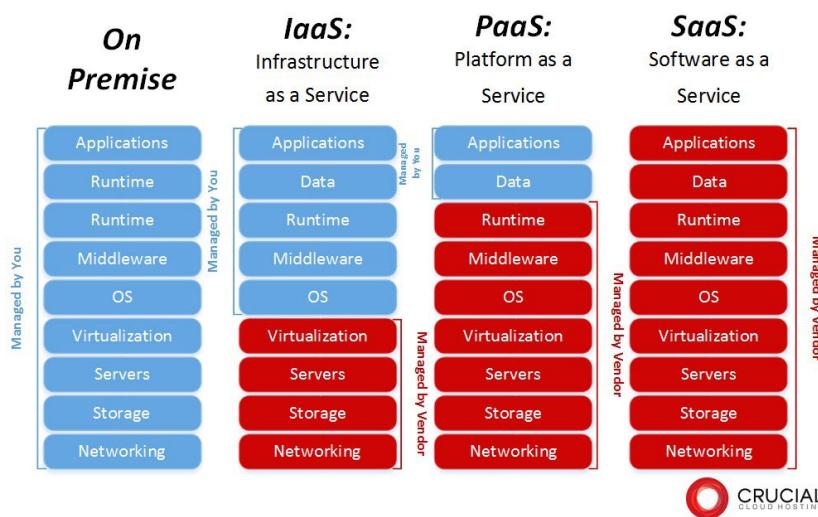
8.5.2	Devices.....	67
8.6	IoT Edge.....	68
8.6.1	IoT hub issues: .....	68
8.6.2	Doelstelling .....	68
8.6.3	Voorbeelden .....	68
8.6.4	Opbouw .....	68
8.6.5	Azure IoT Edge Runtime.....	68
8.6.6	Modules.....	69
8.7	IoT in the Cloud vs IoT on the Edge.....	69
8.7.1	IoT in the Cloud.....	69
8.7.2	IoT on the Edge .....	69
8.8	Samenvatting .....	70
9	MQTT .....	70
9.1	Broker.....	70
9.1.1	Unmanaged services.....	70
9.1.2	Managed services .....	70
9.2	Doel .....	71
9.3	Voordeel.....	71
9.4	Eigenschappen .....	71
9.5	sectoren .....	71
9.6	Topics .....	72
9.6.1	Wildcards .....	72
9.7	Quality of Service (QoS) .....	73
9.7.1	Level 0: fire and forget.....	73
9.7.2	Level 1: Delivered at least once.....	73
9.7.3	Level 2: Delivered exactly once.....	73
9.8	Communicatie.....	73
9.8.1	Payload .....	73
9.8.2	Eigenschappen.....	74
9.8.3	Security .....	74
9.9	MQTT Tools .....	74
9.10	MQTT.NET & Python .....	74
9.10.1	In C#.....	74
9.10.2	In Python .....	75
9.11	MQTT & Azure Functions .....	76
9.11.1	Opstelling .....	76
9.12	Samenvatting .....	78

# 1 Cloud computing

= the practice of using a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server or a personal computer.

## 1.1 Enkele eigenschappen

- Geen eigen hardware (we kopen niets aan)
- Ongelimiteerde computing power
- Ongelimiteerde storage capaciteit
- Scaling up and down op aanvraag of automatisch
- Scaling in en out op aanvraag of automatisch
- Geografische spreiding • Pay what you use



Figuur 1: Verschillende manieren om aan Cloud computing te doen

## 1.2 On premise (eigen servers)

- IT Afdeling is verantwoordelijk voor ALLES
- Back-up & recovery
- Aankoop hardware
- Scaling is duur
- Meeste vrijheid om te werken
- Soms investeren in hardware die je maar beperkt aantal dagen nodig hebt
  - Vb: webshop wordt meer belast tijdens de kerstperiode
- Soms verplicht door wetgeving
- Medische data
- Financiële data
- Gebrek aan vertrouwen bij public cloud provider (cfr NSA)

### **1.3 Hybrid Cloud (On Premise & Cloud)**

Makes use of existing in-house infrastructure and Netplan cloud services to provide the best of both worlds.

- Eigen datacenter koppelen aan cloud omgeving
- Bepaalde diensten draaien in cloud omgeving andere in eigen datacenter
- Zware load laten uitvoeren op de cloud
- Bepaalde data mag NIET in cloud opgeslagen worden (vb: medische gegevens)
- Kan ook gebruikt worden in back-up scenario's

### **1.4 IaaS: Infrastructure as a Service**

- Geen eigen hardware kopen
- We huren virtual machines in Cloud omgeving
- Systeem beheerder moet zelf server configureren en beveiligen en updaten
- Veel flexibiliteit naar software installatie toe
- Zeer veel vrijheid maar ook verantwoordelijkheid
- Je moet zelf scaling doen (soms auto scaling mogelijk)
- Veel gebruik voor migratie bestaande On Premise naar cloud

#### **1.4.1 Voorbeelden**

- Amazon Web Services (AWS)
- Microsoft Azure
- IBM Bluemix
- Google Cloud platform

### **1.5 PaaS: Platform as a Service**

- Geen systeembeheerder nodig
- Ontwikkelaar maakt applicatie en "plaatst" deze op Cloud platform
- We moeten ons geen zorgen maken in servers, hosting, back-ups, scaling,... het platform zal dit voor ons beheren
- Zeer veel flexibiliteit

Wij zullen vooral PaaS gebruiken in deze module.

#### **1.5.1 Talen**

- ASP.NET Core
- NodeJS
- Python
- Java
- PHP

### 1.5.2 Voorbeelden

- Amazon Web Services (AWS)
- Microsoft Azure
- IBM Bluemix
- Google Cloud platform
- Heroku

## 1.6 SaaS

- Software draait meestal niet lokaal (uitzondering Adobe & Office)
- We betalen per maand/gebruiker
- Flexibele abonnementen, snel op te zetten
- We moeten geen rekening houden met back-ups en uptime

### 1.6.1 Voorbeelden

- salesforce
- Office 365
- Dropbox, OneDrive, Google Drive, iCloud
- Gmail
- Adobe Creative Cloud

## 1.7 Belangrijkste vendors

- Amazon Web Services
- Microsoft Azure
- Google Cloud Platform

Figure 1. Magic Quadrant for Cloud Infrastructure and Platform Services



Figuur 2: Magic Quadrant

## 2 Microsoft Azure

We kiezen voor Microsoft Azure omwille van:

- Zowel .NET als Open Source
- Meeste opties en mogelijkheden
- Zeer lage instapdrempel voor studenten

### 2.1 Azure Portal

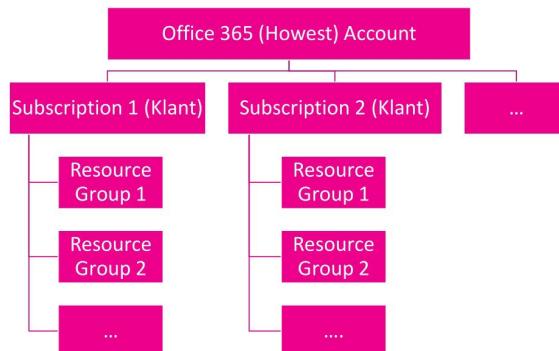
Inloggen via <https://portal.azure.com>

- Beheren van uw Cloud services en applicaties
- Inloggen met Howest Account
- Kies een voldoende sterk passwoord en gebruik 2FA

### 2.2 Azure subscription

- Abonnement
- Dit zal bepalen hoe ze factureren
- Verschillende opties:
  - Credits op voorhand
  - Pay by use
- In de praktijk: per klant een subscription
- Voor de lessen: we hebben de “gratis subscription”
- Meerdere subscriptions mogelijk

### 2.3 Resource Group



Figuur 3: Resource groups

- Logische container
- Per project
- Per soort services – Storage
  - Webservers
  - Fileservers
  - SQL, Blob, ...
- Je kiest dit zelf
- Makkelijk om te testen, je kan een volledige container verwijderen zonder dat je alles apart moet verwijderen
- Toegangsrechten per container zijn mogelijk

- via Office 365 account
- Je kies datacenter locatie van je resource group, niet alle resources moeten op dezelfde locatie staan als je resource group.

## 2.4 Azure Virtual Machines

- Eigen server op Azure omgeving
- Meestal on-premise (server op bedrijf) die we virtueel verplaatsen naar Azure
- Heel wat mogelijkheden – File servers
  - Database servers
  - Webservers
  - Application servers – ...
- Toegang tot resources op server die niet mogelijk zijn via een web applicatie
  - Bv. communicatie met oude software pakketten

Bv. Genereren van Word/Excel documenten – ...

### 2.4.1 Waarom?

Wanneer je volledige controle wenst over je machine

- Volledige controle == volledige verantwoordelijkheid!
  - Niet te onderschatten
  - Security, patching, scaling
- Windows
- Linux

### 2.4.2 Maintenance

- Planned maintenance
  - Updates van het Azure platform (stabiliteit, security, performance)
  - Soms moet de VM herstarten
- Unplanned maintenance
  - Storing in de onderliggende architectuur (netwerk-, disk-, rack-problemen)
  - Azure zal automatisch VM verplaatsen naar werkende infrastructuur

Hoe kan ik mijn VM ‘up and running’ houden?

- Availability set
  - Garantie van 99.95% uptime
  - Minstens 2 virtual machines nodig
  - Duurder

## 2.5 Azure Web App

- Azure Platform Service

- Laat ons toe om webapps online te plaatsen
- We moeten GEEN eigen server aanmaken
- We moeten GEEN webserver configureren
- Click & go
- Ondersteuning voor:
  - ASP.NET (Core)
  - PHP
  - Python Flask
  - Java
  - Node.JS – . . .
- Makkelijkste manier om uw applicatie online te krijgen, easy deployment
- Eenvoudige te koppelen aan een GitHub Repository
- Autoscale (not free)
- Monitoring
- Staging mode

### 2.5.1 Free

= gratis plan voor Azure Web App

- Gedeelde server met andere web apps
- Je weet niet welke server, is transparant voor gebruiker
- 1GB storage
- Beperking op trafiek per dag: 165MB

### 2.5.2 Shared

- Gedeelde server
  - 1GB storage
  - Mogelijkheid tot DNS, vb: www.mijnnaam.be
- 

### 2.5.3 Basic, Premium

- App service plan ⇒ eigen server, dus niks delen met derden (je kan niet inloggen op die server)
- SSL
- Custom domains
- CPU keuze
- Memory keuze
- Scaling tot 3 toestellen

## 2.6 Scaling

### 2.6.1 Scale Up

= **vertical scaling**

- Server krachtiger maken, meer memory en CPU

**Pros:**

- Minder energie dan scale out

- Eenvoudiger te implementeren

- Minder licenties (n.v.t. op Azure Web App) **Cons:**

- Duurder

- Indien we maar 1 machine gebruiken: ⇒ hardware failure en toepassing is down

### 2.6.2 Scale Out

= **horizontal scaling**

- Meerdere machiens maar minder krachtig **Pros:**

- Goedkoper

- Betere bescherming bij hardware failure: je hebt meerdere machines

**Cons:**

- Meer licenties nodig (n.v.t. op Azure)

- Meer plaats in datacenter

- Meer energieverbruik

- Complexer netwerk

- Soms toepassing aanpassen

## 2.7 Azure SQL

- SQL Server Database op Cloud platform
- We moeten zelf geen hardware/software aankopen
- Niet verantwoordelijk voor backups
- Eenvoudige schalen bij zware loads
- 3 opties
  - Serverless Managed Database (onze voorkeur)
  - SQL Managed Instance
  - SQL Virtual Machine

### 2.7.1 SQL Server Cloud Based

- Compatible met SQL server 2012
- Te beheren via Enterprise manager
- Werkt zoals een gewone SQL server
- Connecteren is mogelijk via:

- .NET
- Python
- PHP
- NodeJS – ...
- High Availability
  - Replicatie over 3 servers (default)
  - Automatische Back-ups
- Database draait op een server
- Verschillende pricing mogelijkheden

## 2.7.2 Security

Azure Firewall: IP adres van je netwerk toelaten

## 2.7.3 Azure DTU

- DTU = Database Transaction Units
- Soort ‘munteenheid’
- Gemengde eenheid van:
  - CPU
  - Memory
  - I/O
- Deze resources krijg je ter beschikking
- vCores = virtuele CPUs die de service mag gebruiken
- Hoe meer DTU, hoe meer power, hoe duurder
- <https://docs.microsoft.com/en-us/azure/sql-database/sql-database-what-is-a-dtu>

## 2.7.4 Throttling

= Onderbreken van database communicatie omdat je teveel resources (DTU's) gebruikt (zelf voor retry zorgen).

Throttling Types		
Throttling type	Soft Throttling limit exceeded	Hard Throttling limit exceeded
Temporary disk space problem occurred	0x01	0x02
Temporary log space problem occurred	0x04	0x08
High-volume transaction/write/update activity exists	0x10	0x20
High-volume database input/output (I/O) activity exists	0x40	0x80
High-volume CPU activity exists	0x100	0x200
Database quota exceeded	0x400	0x800
Too many concurrent requests occurred	0x4000	0x8000

Figuur 4: Throttling types

Throttling Modes			
Throttling mode	Description	Types of statements disallowed	Types of statements allowed
0x00	<b>AllowAll</b> - No throttling, all queries permitted.	No statements disallowed	All statements allowed
0x01	<b>RejectUpsert</b> - Updates and Inserts will fail.	INSERT, UPDATE, CREATE TABLE   INDEX	DELETE, DROP TABLE   INDEX, TRUNCATE
0x02	<b>RejectAllWrites</b> - All writes (including deletes) will fail.	INSERT, UPDATE, DELETE, CREATE, DROP	SELECT
0x03	<b>RejectAll</b> - All reads and writes will fail.	All statements disallowed	No statements allowed

Figuur 5: Throttling modes

### 2.7.5 SQL via C#

```

RetryPolicy myretrypolicy = new RetryPolicy<SqlAzureTransientErrorDetectionStrategy>(3,
TimeSpan.FromSeconds(30));

using (ReliableSqlConnection cnn = new ReliableSqlConnection(connString, myretrypolicy, myretrypolicy))
{
    try
    {
        cnn.Open();

        using (var cmd = cnn.CreateCommand())
        {
            cmd.CommandText = "SELECT * FROM HumanResources.Employee";

            using (var rdr = cnn.ExecuteCommand<IDataReader>(cmd))
            {
                //
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message.ToString());
    }
}

```

Figuur 6: Voorbeeld SQL command in C#

### 2.7.6 Resources

- <https://msdn.microsoft.com/en-us/library/azure/dn338079.aspx>
- <http://geekswithblogs.net/ScottKlein/archive/2012/01/27/understanding-sql-azure-throttling-and-i.aspx>

### 2.7.7 Azure Database for MySQL

- Toegang via MySQL Workbench
- Werkt zoals een gewone MySQL DB
- Je moet geen eigen servers opzetten, is volledig managed
- Automatische Back-ups

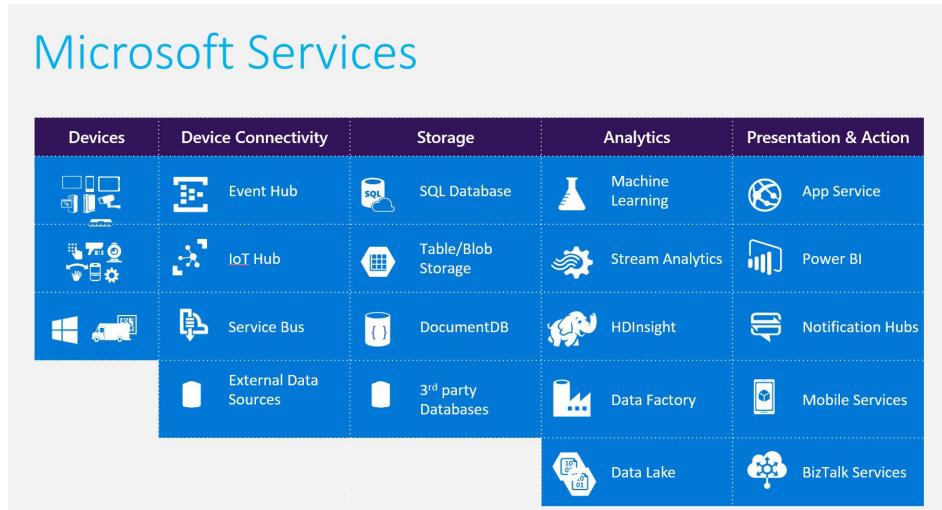
## 2.8 Azure & Internet of Things

### 2.8.1 Waarom is Azure belangrijk voor IoT?

- Azure Event Hubs
  - Ontvangen van berichten afkomstig van toestellen
- **Azure IoT Hub**
  - Ontvangen van berichten

- Versturen van berichten naar toestellen
- Azure Streaming Analytics
  - Verwerken van events afkomstig van Event Hubs en IoT Hub

**Bovenstaande zeer belangrijk voor ons!**



Figuur 7: Microsoft Services

## 2.9 Azure CLI

### 2.9.1 Azure Portal

- OK voor dagelijks gebruik
- Niet makkelijk te automatiseren
- Wat als ik 100 sites nodig heb?
- Wat als ik 50 servers nodig heb?

### 2.9.2 Oplossing: Azure CLI 2.0

- Commandline Azure resources aanmaken
- Makkelijk met scripts
- Ideaal voor DevOps

## 2.10 Bash/Powershell

- Bash commandline in portal
- Alles wat je via UI kan doen kan je ook via commandline in de portal

## 2.11 Andere Azure onderdelen

The screenshot shows a grid of Azure services. Each service is represented by an icon and a name. Some services have a star icon next to them. A search bar at the top left says 'Search Everything'. At the top right, there's a rating scale from 0 to 9 labeled 'How likely are you to recommend portal.azure.com?'. Below the grid, there's a section titled 'Recent' with a list of services: Cost Management + Billing, Reservations, Marketplace, Quickstart Center, Resource groups, Service Health, Shared dashboards, Subscriptions, Help + support, Management groups, Marketplace, Recent, Reservations, Tags, What's new, Container services (deprecated), Function App, Service Fabric clusters, Mesh applications, Dials (classic), Shared image galleries, Container instances, Availability sets, Disks, Image definitions, Image versions, On-premises Data Gateways, Route tables, Network security groups, Network interfaces, Public IP addresses, Front Door, Load balancers, Application Gateways, ExpressRoute circuits, Network security groups (classic), Connections, DDoS protection plane, Firewalls, Front Door WAF policies, Recovery Services vaults, Storage Sync Services, ImportExport jobs, Data Share invitations, Storage accounts, Storage Sync Managers, Azure NetApp Files, Data Shares, StorSimple Device Managers, Data Lake Storage Gen1, Data Box Edge / Data Box Gateway, App Management services, CDN profiles, Search services, Notification Hubs, App Service environments, API Connections, App Service Certificates, Media services, SignalR, Container instances, Kubernetes services, Container registries, Batch accounts, App Services, and App Service Domains.

Figuur 8: Overzicht Azure onderdelen (ter illustratie: het zijn er veel)

## 2.12 Samenvatting

- Wat is Cloud computing?
  - Wie zijn de grote spelers?
  - Welke soorten zijn er en wat zijn hun eigenschappen?
  - Wat is een Azure subscription?
  - Wat zijn resource groups?
- Hoe kan je scalen?
- Voor en nadelen van Azure VM's?
  - Wanneer Azure VM gebruiken?
  - Wat is SQL Azure en wat zijn DTU's?
  - Wat is throttling (bij SQL Azure)?
  - Wat is een Azure Web App?

# 3 Back-end services met Azure Functions

## 3.1 The Internet Of ...

### 3.1.1 The Internet of Information

- = webpagina's met content
  - Opzoeken van informatie
  - Raadplegen van informatie
  - E-commerce
  - Social networks

- Entertainment
- Technologieën
  - HTML
  - CSS
  - JavaScript
  - PHP/ASP/JAVA
- Voorbeelden – Google
  - Facebook
  - Amazon
  - Stackoverflow
  - Nieuwssites
  - Reddit

### 3.1.2 The Internet of Services

- Connectiviteit
  - Driving factor for mobile applications – Cross platform
  - Data op makkelijke manier uitwisselen
  - Functionaliteit makkelijk aanroepen
  - Centraal staan Cloud platforms
  - REST-Based
- Voorbeelden – Netflix
  - Azure
  - Amazon Web Services (AWS)
  - IBM Bluemix

### 3.1.3 The Internet of Things

= Connecting hardware

- Devices koppelen aan het internet
  - Versturen van informatie naar cloud
  - Toestellen praten onderling met elkaar (M2M = machine to machine)
- Communicatie van/naar toestel
  - We kunnen berichten sturen naar toestel
  - We ontvangen berichten van toestel in de cloud
- Voorbeelden
  - Smart thermostat
  - Smart fridge
  - Auto's die verbonden zijn met de cloud (Tesla)

- Andere smart home devices (Alexa, Google Home, . . . )

Internet of Things & Internet of services overlappen

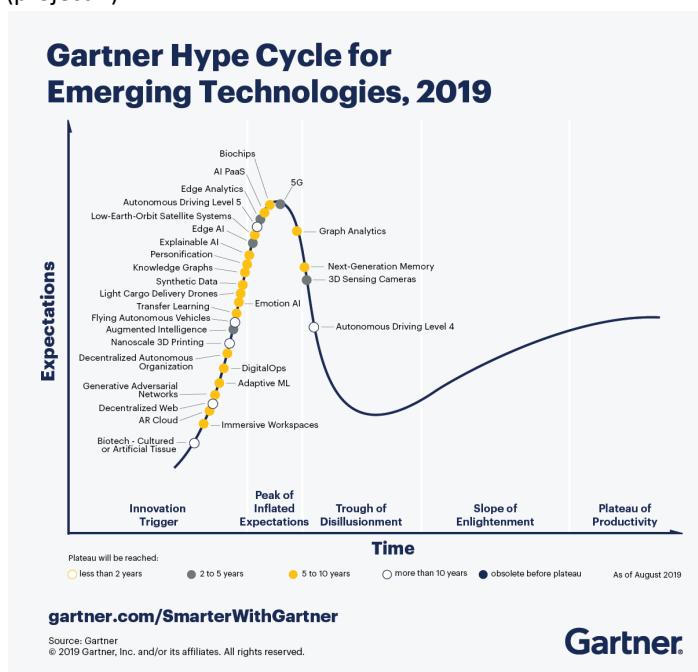
### 3.1.4 Volgende stap: The Internet of Value

"Value"verhandelen via het internet

- Bitcoin
- Andere cryptocurrencies

Onderliggende technologie veel belangrijker: **Blockchain**

- Distributed ledger (grootboek)
- Registreren van transacties in die ledger
- Iedere transactie bevat een verwijzing naar de vorige transacties (chain, linked list)
  - Blockchain
- Public
- Gratis
- Open
- Private blockchain mogelijk
  - Veel evolutie in de wereld van de fintech
  - Etherium
  - Smart contracts
- IoT & blockchain zeer veel potentieel
- Bachelor-proef (project 4)



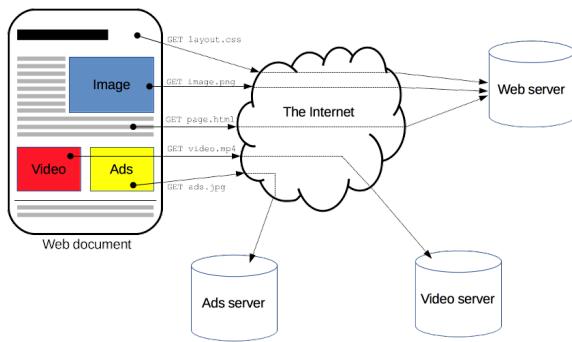
Figuur 9

Onze focus: Internet of Things & Internet of Services

## 3.2 Herhaling: Hoe werkt HTTP?

OSI model		
Layer	Name	Example protocols
7	Application Layer	HTTP, FTP, DNS, SNMP, Telnet
6	Presentation Layer	SSL, TLS
5	Session Layer	NetBIOS, PPTP
4	Transport Layer	TCP, UDP
3	Network Layer	IP, ARP, ICMP, IPSec
2	Data Link Layer	PPP, ATM, Ethernet
1	Physical Layer	Ethernet, USB, Bluetooth, IEEE802.11

Figuur 10: HTTP staat op OSI laag 7: de application layer



Figuur 11: Hoe werkt een website?

### 3.2.1 Wat is HTTP?

- Hyper-Text Transfer Protocol
- Onderliggende protocol waarop Internet werkt
- Opvragen van tekst, bestanden vanaf servers
- Request *meestal* afkomstig van een webbrowser maar ook smartphone, IoT device
- HTTP zal bepalen hoe een request en response er moeten uitzien
- HTTP bevat een aantal commando's: **HTTP Verbs**
- HTTP is **stateless**: het zal geen rekening houden met voorgaande requests ("Fire and Forget")
- HTTP is **niet sessionless**: we kunnen cookies (client-side) gebruiken om data bij te houden
- HTTP is relatief eenvoudig: volledig text-based

### 3.2.2 HTTP Verbs

= HTTP commands/methods

- **GET** - ophalen data (safe: wijzigt niets op server) (SELECT)
- **POST** - Toevoegen data: vb: formulier (INSERT)
- **PUT** - Idempotent => meerdere requests hebben hetzelfde zelfde effect (UPDATE)
- **DELETE** - Idempotent => meerdere requests hebben hetzelfde zelfde effect (DELETE)

### 3.2.3 HTTP status code

HTTP Response bevat naast data (html of andere) ook een status code:

- 1xx: informatief • 2xx: success
- 3xx: redirection
- 4xx: client error
- 5xx: server error

Niet altijd evident om te weten wat je moet kiezen !

### 3.2.4 HTTPS

- Beveiligen van transport
- Geen beveiliging van de data
- Defacto standaard, zonder HTTPS mag je eigenlijk **niet** in productie plaatsen
- Browsers melden dit reeds, vb:Chrome



Figuur 12: HTTPS info in Google Chrome. Hier: geen HTTPS verbinding, enkel HTTP

### 3.2.5 HTTP Request

```
GET http://www.honest.be/ HTTP/1.1
Host: www.honest.be
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.91 Safari/537.36
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8,nl;q=0.6
```

Figuur 13: Voorbeeld HTTP GET-Request

- Belangrijkste HTTP Header info: User-Agent zal dit opstellen
- GET = HTTP verb
- Host = de server
- User-Agent = browser info
- Accept = welke type data kan de client ontvangen

### 3.2.6 HTTP Response

```
HTTP/1.1 200 OK
Date: Tue, 19 Sep 2017 13:06:28 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 61041
```

Figuur 14: HTTP Response met status code 200

- Belangrijkste HTTP Header info: Server zal dit opstellen
- 200 OK = status code
- Content-Type = wat is het type van de info die ik ontvang, vb text/html

- Inhoud resource: vb: HTML code

### 3.2.7 HTTP/2

- Sinds 2015 actief
- High-Level compatibility met HTTP/1.1 - methods, status codes, URI's en Header fields zijn hetzelfde
- Request multiplexing over een enkele TCP verbinding
  - Meerdere request in parallel mogelijk over 1 tcp connection
  - Asynchroon downloaden van meerdere bestanden mogelijk
- Header compression (optimalisatie van het netwerk)
- Binary Protocol (HTTP/1.1 is tekst protocol)
- **HTTP/2 Server Push:** staat een HTTP/2-server toe om resources te sturen naar een HTTP/2compatibele client voor de client ze vraagt (=performancetechniek)

## 3.3 Webservices

= functies die we kunnen aanroepen via het internet

- Opvragen van data via het internet
- Devices aanspreekbaar maken via het internet
- Protocol is default (HTTP)
- Technologie-onafhankelijk (Java, C#, PHP, Python)
- Platform-onafhankelijk (Windows, Linux, Android, iOS)
- 2 Protocollen:
  - SOAP (verouderd, meestal gebruikt tussen 2000-2010, gebruiken wij niet)
  - REST (onze keuze)
- Datatransport (formaat van de data) via
  - XML
  - JSON

### 3.3.1 Open data

= data die vrij beschikbaar is

- Opvraagbaar via webservices
- Meer en meer steden stellen data beschikbaar:
  - Gent: <https://data.stad.gent/>
  - Federale overheid: <http://data.gov.be/en>

### 3.3.2 Cognitive services

- = AI API op Azure
  - <https://azure.microsoft.com/en-us/services/cognitive-services/>
  - Spraakherkenning
  - Beeldherkenning

- Taal – . . .

### 3.3.3 Internet of Things

- Philips Hue API (<http://developers.meethue.com/>)
- Parrot AR Drone (<https://github.com/andrew/ar-drone-rest>) Alle voorgaande voorbeelden:

  1. Maken gebruik van HTTP
  2. GET/POST request naar device
  3. In de body van het request: JSON data
  4. Cross-platform

Deze manier van werken zal je bijna altijd tegenkomen!

### 3.3.4 JSON

- Het meest gebruikte formaat voor data is XML en JSON (wij kiezen voor JSON)
- De data die je terug krijgt zal altijd mappen met een object aan de serverkant

#### **JSON = JavaScript Object Notation**

- Eenvoudig, zeer licht formaat
- We schrijven alles tussen accolades
- Basis is altijd key:value pair
  - De key = lowercase, altijd tussen quotes
  - Value: indien string ⇒ quotes
  - Key en value gescheiden door een dubbelpunt :
  - Key:value koppels gescheiden door een komma ,
  - Validatie en testen kan op <https://jsonlint.com/>

#### **Complexe JSON**

- Een key kan als value terug een JSON string bevatten
- Daarbinne kan je terug een key plaatsen met een nieuwe JSON string
- Je kan zoveel nesten als je wil.

#### **Complexe JSON Arrays**

- Wanneer er meerdere JSON objecten zijn spreken we van een array
- Deze plaatsen we tussen []
- Een value kan terug een array van andere JSON objecten zijn

### 3.3.5 URL

- Alles is uniek identificeerbaar via een URL (Uniform Resource Locator)
- Gebruik **geen** spaties in de URL
- Gebruik **geen** hoofdletters
- Gebruik meervoud voor resource namen (gebruik users ipv user)

- Gebruik GEEN HTTP Verb om operatie aan te duiden (vb: geen get of post in url)!!

#### Voorbeelden

- `http://localhost:8080/UserManagement/rest/UserService/getUser/1`
  - NIET OK: operatienaam in URL en hoofdletters
- `http://localhost:8080/usermanagement/rest/userservice/users/1`
  - OK: resourcenaam = meervoud, gevolgd door ID van de user die we willen opvragen

#### 3.3.6 Fouten

Wat als er iets foutloopt in een service?

- Wat moet ik returnen?
- Stuur **NOOIT** Exceptions of interne foutmeldingen van het systeem terug naar de client in productie systemen
- Keer een status code Internal Server Error 500 terug met eventueel wat informatie die jij zelf opstelt
- Denk na welke info je bij de foutmelding wil terugsturen
  - Geen connectie informatie
  - Geen database namen
  - Geen SQL statements – Wees voorzichtig. . .

#### 3.3.7 Samenvatting

- HTTP GET
  - Alleen lezen (SELECT in database)
- HTTP PUT & DELETE
  - DELETE = DELETE in SQL
  - PUT = UPDATE in database
  - **Idempotent methods** ⇒ voer je ze 1x uit of 100x, het resultaat blijft hetzelfde
- HTTP POST
  - INSERT in SQL

#### Webservices zijn stateless

- Geen state van de client bijhouden aan serverkant
- Geen sessies gebruiken
- Bij iedere request naar de server moet ja alle info meesturen zodat server het request kan afhandelen
- Ieder request is onafhankelijk van elkaar
- Dit verhoogt de schaalbaarheid
- Eenvoudiger applicatie design
- **Vb:** vanuit een mobile app vragen we sensor data op via webservice. De gebruiker van de app kan een filter instellen en wil enkel de temperatuur sensor data zien. Dit wil

zeggen bij iedere request naar de server MOET je meegeven welke data de gebruiker wil zien. Je mag aan de server kant NIET bijhouden welke gebruiker welke sensors data wenst te zien

### 3.3.8 Hoe maken we webservices

Aanwezig op ieder platform:

- PHP
  - Laraval, Lumen, Wave
- Python
  - Eve, Flask-Restfull
- .NET
  - ASP.NET Core met C#
  - Azure Functions

\* Met C#, NodeJS of Python

## 3.4 Azure Functions

= Serverless functions

### 3.4.1 Waarom?

- Eenvoudig concept, zeer eenvoudig aan te maken
- Ondersteuning voor verschillende talen
  - Wij gebruiken C# maar Python kan ook: <https://docs.microsoft.com/en-us/azure/functions/functions-reference-python>
- Snel en eenvoudig schaalbaar
- Geen zorgen over servers en serverbeheer (serverless)

## 3.5 Azure Functions Security

- Iedereen kan functies aanspreken
- Geen controle over wie wat doet
- Een hacker kan de factuur doen oplopen door constant de service aan te roepen
- Andere normale gebruikers zullen tragere API aanroepen krijgen
- Concreet: we moeten dit proberen te vermijden

### 3.5.1 3 soorten security

1. Anonymous
  - Iedereen kan de functie aanroepen
2. Function
  - Er zal een function key meegegeven worden via de URL
  - Deze key kan alleen gebruikt worden bij de gekozen functie
3. Admin
  - De admin key zal mee moeten verstuurd worden via de URL

- Deze key kan gebruikt worden voor alle functies binnen de applicatie

In code: via HttpTrigger

```
[FunctionName("MySecureAPIAdmin")]
public static HttpResponseMessage MySecureAPIAdmin([HttpTrigger(AuthorizationLevel.Admin,
[FunctionName("MySecureAPIFunction")]
public static HttpResponseMessage MySecureAPIFunction([HttpTrigger(AuthorizationLevel.Function,
[FunctionName("MySecureAPIFunctionAnonymous")]
public static HttpResponseMessage MySecureAPIFunctionAnonymous([HttpTrigger(AuthorizationLevel.Anonymous,
```

Figuur 15

<https://myazurefunctions.demos.azurewebsites.net/api/secureapifunctionkey?code=BbNn5StOGluAsJqoxgyHHcQh05gIXL7F1P1lYn/fcmpWPFk0GgvJmA==>

Figuur 16: De key zit in de URL

### 3.5.2 Voordelen van key security

- Makkelijk op te zetten
- Beheer keys valt mee bij laag aantal gebruikers
- Ideaal voor scenario waar je controle hebt over de client
  - Vb: je interne Android of iOS hebt (app niet publiek in store)
  - Vb: interne web applications

### 3.5.3 Nadelen van Key Security

- Key moet in de applicatie zitten die we verspreiden
  - Vb: publieke mobile apps die iedereen kan downloaden
- Bij zeer veel gebruikers zal key management lastig worden **Oplossing:**
- JWT tokens
- Inloggen via:
  - Facebook
  - Twitter
  - Google Account
  - Microsoft Account

## 3.6 Andere Azure Functions

- HTTPTrigger
  - Webservice
- Timer trigger
  - via cron expressie de functie op een tijdstip uitvoeren
  - 0 \* /5 \* \* \* (=once every five minutes)
  - <https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-timer>
- IoT Hub Trigger
- ...

### 3.7 Whats next

Welke nieuwe technologie moeten we volgen?

- GraphQL (<https://graphql.org/>)
  - Query taal voor API
  - We sturen queries door naar de API die resultaten zal returnen
  - In volle ontwikkeling
- gRPC (<https://grpc.io/>)
  - Open Source Remote Procedure Framework
  - We roepen methodes aan op remote servers
  - Platform-onafhankelijk en open source
  - Enkel HTTP/2
- Mooie onderzoeksvragen voor Project 4 in 3MCT

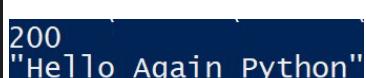
### 3.8 Azure met Raspberry Pi

#### 3.8.1 4 scenarios'

1. HTTP GET ophalen van data uit Webservice op Azure
  2. HTTP POST naar Webservice op Azure
  3. HTTP PUT updaten van data
  4. HTTP DELETE verwijderen van data
- Wij maken gebruik van Python op de Raspberry Pi of op de PC
  - Andere mogelijkheden zijn JavaScript, C#, C++, . . .
  - Standaard geen support voor HTTP Requests
    - Wij maken gebruik van Python Requests library
    - Zeer eenvoudig in gebruik
    - Goede documentatie: <http://docs.python-requests.org/en/v0.10.6/>
    - Gebruik User Guide uit de documentatie

#### 3.8.2 GET

```
import requests
url = 'https://myazuredemos.azurewebsites.net/api/HelloWorld/Python'
ret = requests.get(url)
print(ret.status_code)
print(ret.text)
```



The screenshot shows a terminal window with two parts. On the left, a Python script is displayed in a code editor. It imports the requests module, defines a URL for an Azure function, and uses requests.get() to make a GET request. It then prints the status code and the text of the response. On the right, the terminal output shows the response from the Azure function, which includes a status code of 200 and the text "Hello Again Python".

Figuur 17: HTTP GET request en response

Werking:

- Importeren requests library
- url opstellen die we willen aanroepen
- requests.get() methode uitvoeren met url als parameter
- status\_code = HTTP Statuscode

- text = inhoud van response

### 3.8.3 POST

```

import requests
import json

#HTTP POST
print("HTTP POST DELING")
payload = {'getal1' : 4 , 'getal2' : 2}
url = 'https://myazurefunctionsdemos.azurewebsites.net/api/rekenmachine/delen/'
json = json.dumps(payload)
ret = requests.post(url,data=json)
print(ret.status_code)
print(ret.text)

```

HTTP POST DELING  
200  
{"quotient":2}

Figuur 18: HTTP POST request en response

- Importeren van requests + json
- Payload bevat Python dictionary
- Deze omzetten naar json string via json.dumps
- Via request.post kunnen we de data parameter opvullen met json in de body
- We krijgen statuscode terug
- We krijgen inhoud van body terug

```

jsonstring = ret.text
obj = json.loads(jsonstring)
print("Het resultaat is {}".format(obj["quotient"]))

```

Figuur 19: Hoe kunnen we de ontvangen JSON inladen en opvragen?

- Ontvangen JSON string opslaan in variabele
- Ontvangen JSON string inladen via loads in een dictionary
- Daarna kunnen we de waarden uit de dictionary opvragen zoals altijd

## 3.9 Azure via .NET

- HttpClient gebruiken in .NET
- Zie les device programming

## 3.10 Samenvatting

- Over welke soorten ‘Internet’ spreken we?
- Wat zijn Webservices en hun eigenschappen?
- Wanneer moet je GET/POST/PUT/DELETE gebruiken?
- Welk HTTP verbs zijn idempotent?
- Welke statuscodes moet ik terugsturen?
- Wat is JSON en zorg dat je manueel JSON kan schrijven?
- Hoe kan je Azure Functions aanroepen vanuit Python?
- Welk soorten Azure Functions zijn er?
- Wat is een cron expression?

## 4 Azure Storage

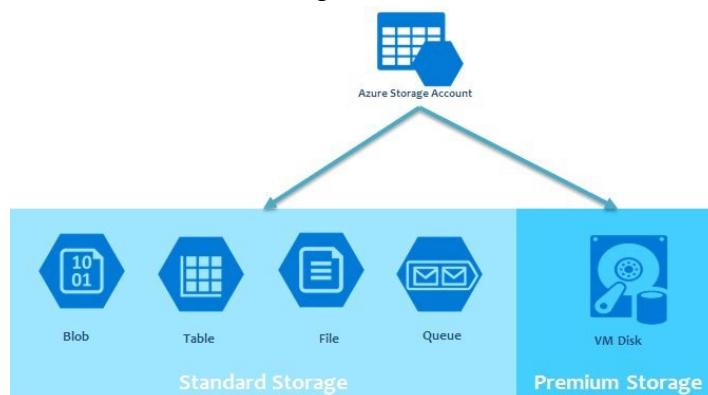
- Opslag van data in Cloud omgeving
- Zeer flexibel in gebruik en prijs
- Unlimited storage (toch geen limiet waar wij zullen tegenlopen)
- Basis voor heel wat Azure services: VM's, Functions, . . .

### 4.1 Soorten storage

- File (zien we niet in labo)
- Disk (zien we niet in labo)
- Blob
- Queue
- Table

### 4.2 Azure Storage account

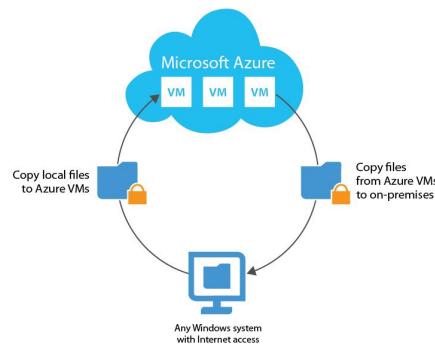
- Alles komt hierin samen
- Dit moeten we aanmaken in Azure
- Daarbinnen maken we dan de soorten storage aan



Figuur 20: Azure Storage account

### 4.3 Azure Files

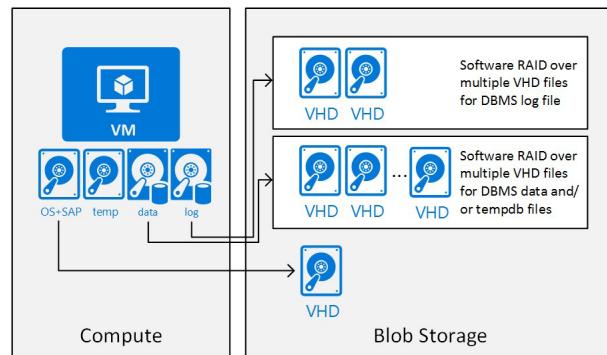
- Hard Disk maar in de Cloud
- Je kan deze mappen naar eigen pc, vb Z:\. . .
- Handig als je veilig files wil kopiëren tussen Cloud server en on-premise server
- Zal de gekende protocollen volgen zoals SMB 3.0
- Veel gebruikt in Hybrid Cloud
- Werkt niet op school (blocked)
- Kan je thuis eens proberen



Figuur 21: Azure File Storage

#### 4.4 Azure Disk Storage

- Basis voor Azure VM
- Op deze locaties komt de server te staan
- Ook mogelijkheid om datadisks te maken
- Hoge beschikbaarheid
- Lage latency
- Hoge throughput (2000MB/s)
- Mogelijkheid voor SSD disk
- Disk Encryption beschikbaar



Figuur 22: Azure Disk Storage

#### 4.5 Azure Blob Storage

- ‘blob’ == binary large object
- Opslag van bestanden: afbeeldingen, PDF, CSS, JS files van web apps kan je hier plaatsen
- Single page apps (Vue, Angular, Blazor)
- Container
  - Bevat de bestanden

- Naam opgeven
- Public Access Level (wie kan welke bestanden lezen)
  1. Private (default): extern bekijken is niet mogelijk
  2. Blob: extern lezen per blob is mogelijk
  3. Container: extern lezen van volledige container is mogelijk

The screenshot shows the Azure Blob container creation interface. At the top, there are buttons for 'Open in Explorer', '+ Container', 'Refresh', 'Move', and 'Delete storage account'. Below these are tabs for 'New container' and 'Existing container'. A 'Name' field is filled with 'fotos'. Under 'Public access level', a dropdown menu is open, showing options: 'Private (no anonymous access)' (selected), 'Private (anonymous access)', 'Blob (anonymous read access for blobs only)', and 'Container (anonymous read access for containers and blobs)'. A green checkmark icon is visible next to the name input field.

Figuur 23: Azure Blob container toevoegen in Azure Portal

- Bestanden uploaden via storage explorer (<https://azure.microsoft.com/en-us/features/storage-explorer/>)
- Connecteren met accountnaam/access key
- We kunnen bestanden opvragen via HTTP Requests

#### 4.5.1 Pricing

- Prijs is in GB/maand
- Hot of cool access
- We betalen ook de lees en schrijfoperaties

Data storage prices pay-as-you-go			
All prices are per GB per month.			
	PREMIUM	HOT	COOL
First 50 terabyte (TB) / month	€0.16445 per GB	€0.0166 per GB	€0.00844 per GB
Next 450 TB / month	€0.16445 per GB	€0.0159 per GB	€0.00844 per GB
Over 500 TB / month	€0.16445 per GB	€0.0153 per GB	€0.00844 per GB

Figuur 24: Prijstabel Azure Blob Storage

#### 4.5.2 Hot Access

- Data die nu in gebruik is en waar we veel lezen en schrijver
- Data die klaar staat om eventueel later naar vool storage te verplaatsen

#### 4.5.3 Cool Acces

- Backups voor lange termijn
- Data die we niet frequent nodig hebben

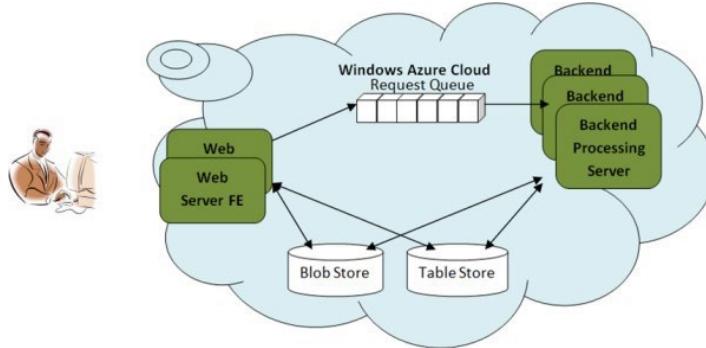
#### 4.5.4 Static website

- Eenvoudige HTML website
- Ideaal voor VueJS, Angular, Reactor, Blazor, . . .

- Eventueel backend in Azure Functions en via JavaScript aangeroepen
- Zeer goedkoop
- <https://docs.microsoft.com/en-us/azure/static-web-apps/overview>

## 4.6 Azure Storage Queues

- Een **wachtrij** in de Cloud
- Een applicatie zal berichten op de wachtrij plaatsen (**Sender**)
- Een tweede applicatie kan deze berichten op een zelf gekozen moment ophalen en verwerken (**Receiver**)
  - We spreken over een **decoupled** applicatie: zender en ontvanger werken onafhankelijk van elkaar
- Zeer **schaalbaar** voor meerdere queues op te starten
- **Resilience:** buffer van berichten op queue zorgt ervoor dat er niks verloren gaat als back-end wegvalt



Figuur 25

### 4.6.1 Voorbeeld

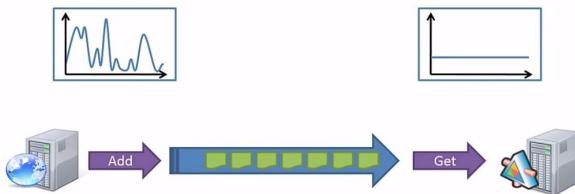
- Sender = webshop
- Ontvanger = stuk software (bv Azure Functions) die order zal controleren en naar een nieuwe wachtrij zal sturen



Figuur 26: Voorbeeld webshop

### 4.6.2 Load leveling

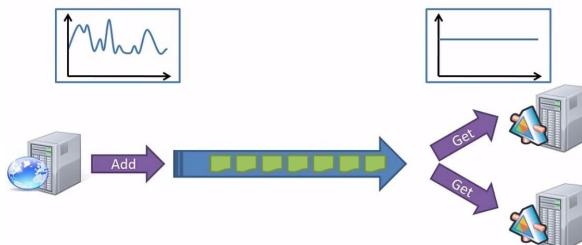
- Constante load op applicatie die verwerkt moet worden
- Werkt zolang we niet meer berichten aanmaken dan wat de applicatie kan verwerken



Figuur 27: Load leveling

#### 4.6.3 Load balancing

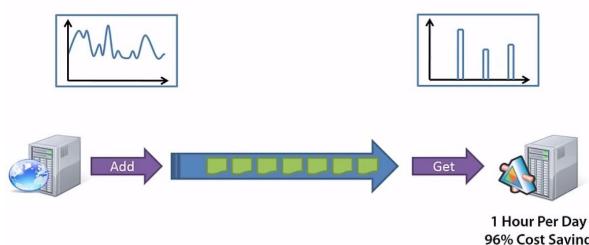
- Teveel berichten per uur
  - Applicatie die verwerkt schalen
  - Nieuwe instantie  $\Rightarrow$  verdubbeling verwerking
- Wanneer terug minder berichten per uur: 1 applicatie stoppen •  
High Availability  $\Rightarrow$  bij crash 1 app, ligt systeem **niet** plat



Figuur 28: Load balancing

#### 4.6.4 Temporal devoupling

- We slaan alles op in Queue voor later
- Applicatie voor werking start om 23u op
  1. Verwerkt alles
  2. Shutdown app

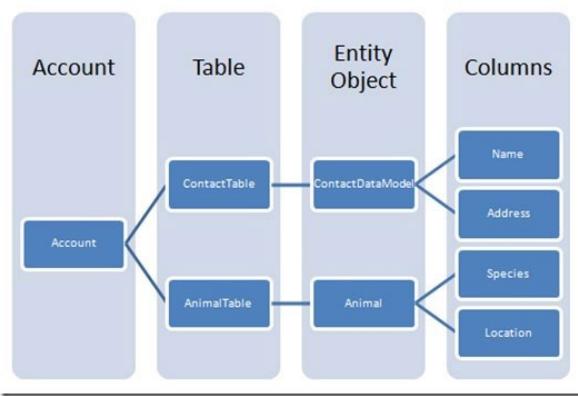


Figuur 29: Temporal decoupling

### 4.7 Azure table storage

- No-SQL key/value store
- Opslag van PetaBytes aan data
- Geo-redundant storage

- Flexibel dataschema, aantal kolommen per rij moet niet hetzelfde zijn
- Vergelijk het met een spreadsheet die je invult
- Geen relaties, geen joins, geen stored procedures
- Eenvoudig in gebruik
- Voorbeeld: <https://www.troyhunt.com/working-with-154-million-records-on/>



Figuur 30: Structuur Azure Table Storage

- Partitions
  - Verdelen van tabel over partities
  - Iedere partitie op eigen server
  - Zal beter schalen en load verdelen op server
- Load balancing over 3 servers
  - Azure zal data repliceren op 3 servers
  - Verdelen van load over deze servers

## PARTITIONING TABLES

PartitionKey	RowKey	Title	
2010	1000	Blogtitle1	Partition 1
2010	1001	Blogtitle2	
2010	1002	Blogtitle3	
2009	1003	Blogtitle4	Partition 2
2009	1004	Blogtitle5	

Figuur 31: Partitions

### 4.7.1 Entities (rijen)

- Max 1MB
- 255 properties (kolommen), waarvan 3 verplicht:
  - Partition key
  - Row key
  - Timestamp (automatisch)
- Niet iedere rij moet evenveel kolommen hebben

#### 4.7.2 Table Storage Data Access

- Via REST API ⇒ cross platform HTTP requests (.NET, PHP, Android, Objective C)
- Storage Client API (via Nuget Package Azure Storage)
- Voor andere platformen zijn er ook libraries (iOS, Android, Python, . . . )

#### 4.7.3 Queries

- Max 1000 items returnen
- Indien > 1000 ⇒ Continuation token
- Query > 30 sec ⇒ cancelled
- Geen index mogelijk
- Query op partition key & row key ⇒ snel

#### 4.7.4 Mogelijke kolom types:

- Byte[] (=bytearray)
- Bool
- DateTime
- Double
- GUID
- Int32 of int
- Int64 of long
- String

#### 4.7.5 Waar gebruiken?

- Profiel info (zal niet veel wijzigen)
- Device info (bv: alle IMEI nummers van GSM toestellen binnen een netwerk)
- Telemetry data (bv: sensor netwerken die om de 5sec waarde van sensor doorsturen)
- Data voor AI modellen
- Alles waar je zeer veel niet-wijzigende data wenst op te slaan

### 4.8 Azure Storage Tools

Met behulp van de Storage Explorer:

<https://azure.microsoft.com/en-us/features/storage-explorer/>

### 4.9 Programmeren van Azure Storage

#### 4.9.1 Wanneer?

- Uploaden van bestanden naar Blob storage
- Versturen van berichten naar wachtrijen (Queues)
- Wegschrijven van records naar Table Storage

Dit kan je allemaal programmeren via C#, Python of JavaScript

- In .NET Nuget Package toevoegen (zoals pip install bij Python)

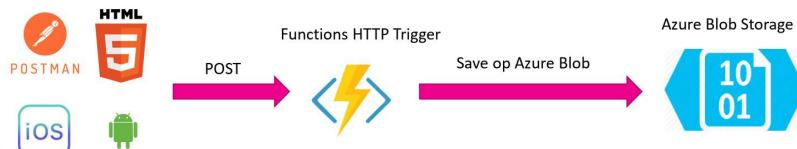
- Rechtermuisknop op dependencies ⇒ Manage Nuget packages
- Zoeken naar juiste package, vb: Azure Storage



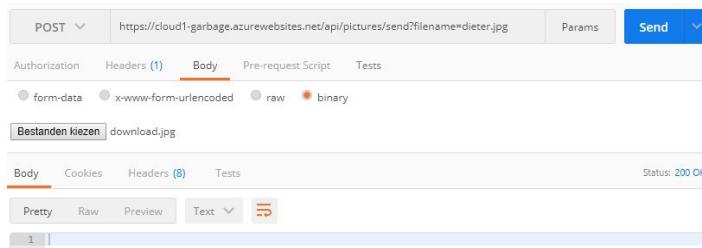
Figuur 32: Azure Storage toevoegen in Visual Studio 2019

## 4.10 Enekele scenarios' (meer info: zie labo's)

### 4.10.1 Scenario 1: Post binary file naar Azure Functions en opslage in Azure Blob



Figuur 33: POST binary file naar Azure Functions, opslag in Azure blob



Figuur 34: De Binary data komt in de body van het HTTP Request via Postman

```

[FunctionName("FileUpload")]
public static async Task<IActionResult> FileUpload(
    [HttpTrigger(AuthorizationLevel.Anonymous, "post", Route = "upload/{fileName}")] HttpRequest req,
    string fileName,
    ILogger log)
{
    try
    {
        log.LogInformation("Upload start");
        CloudStorageAccount storageAccount = CloudStorageAccount.Parse(Environment.GetEnvironmentVariable("BlobStorage"));
        CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();
        CloudBlobContainer container = blobClient.GetContainerReference("uploads");
        CloudBlockBlob blockBlob = container.GetBlockBlobReference(fileName);
        await blockBlob.UploadFromStreamAsync(req.Body);
        log.LogInformation("Upload done");
        return new OkObjectResult("");
    }
    catch (Exception ex)
    {
        log.LogError(ex.Message);
        return new StatusCodeResult(500);
    }
}

```

Figuur 35: Daarna slaan we de file op in de Blob Storage

```

public static async Task<IActionResult> FileUpload(
    [HttpTrigger(AuthorizationLevel.Anonymous, "post", Route = "upload/{fileName}")] HttpRequest req,
    string fileName,
    ILogger log)

```

Figuur 36: Via de parameter in URL geven we de naam van het bestand mee aan service

```

CloudStorageAccount storageAccount = CloudStorageAccount.Parse(Environment.GetEnvironmentVariable("Storage"));

```

Figuur 37: Connectie maken met storage via connectiestring die we ophalen uit localsettings.json file  
(Connectiestring Storage staat op Azure)

```
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();
```

Figuur 38: Via CloudBlobClient kunnen we praten met de storage

```
CloudBlobContainer container = blobClient.GetContainerReference("uploads");
```

Figuur 39: We halen referentie op naar de container waar we de files willen opslaan

```
CloudBlockBlob blockBlob = container.GetBlockBlobReference(fileName);
```

Figuur 40: We maken 'lege' file aan

```
await blockBlob.UploadFromStreamAsync(req.Body);
```

Figuur 41: We uploaden de stream van de body in de 'lege' file

#### 4.10.2 Scenario 2: Stuur e-mail wanneer een blob verschijnt op Azure Blob



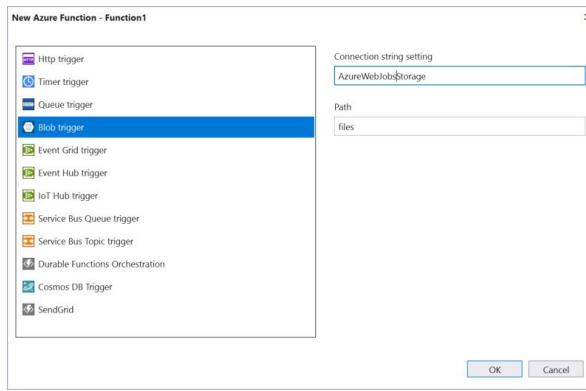
Figuur 42: Stuur e-mail wanneer een blob verschijnt op Azure Blob

##### Blob trigger

- Functie die automatisch uitvoert als er een Blob file aangemaakt wordt
- Vb:
  - Uploaden van foto's naar blob voor betere verwerking
  - Uploaden CSV files voor verwerking

```
{  
    "IsEncrypted": false,  
    "Values": {  
        "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;AccountName=...;AccountKey=...",  
        "FUNCTIONS_WORKER_RUNTIME": "dotnet",  
        "Storage": "DefaultEndpointsProtocol=https;AccountName=...;AccountKey=..."  
    }  
}
```

Figuur 43: In de settings file plaatsen we bij AzureWebJobsStorage de connectionstring naar onze storage account



Figuur 44: Connectionstring = de key uit de settings file, Path = de naam van de container

- Wat kunnen we doen met Blob?
  - Sturen naar wachtrij
  - Sturen naar Deep Learning netwerk voor training – . . .
- Dit voorbeeld: e-mail met link
- Hiervoor maken we gebruik van SendGrid en MailKit

### SendGrid

= Platform om emails te sturen

1. In Azure Portal: nieuwe resource: 'SendGrid Email Delivery'
2. Kies voor het 'Free' pricing tier
3. SendGrid package: via Nuget Package manager in Visual Studio 2019

```
var apiKey = Environment.GetEnvironmentVariable("SENDGRID_API_KEY");
var client = new SendGridClient(apiKey);
var from = new EmailAddress("dieter.de.preester@howest.be", "Blog upload");
var subject = $"File upload done {name}";
var to = new EmailAddress("dieter.dp@mail.com", "Example User");
var plainTextContent = "Er is een nieuw bestand beschikbaar";
var htmlContent = "<strong>Er is een nieuw bestand beschikbaar</strong>";
var msg = MailHelper.CreateSingleEmail(from, to, subject, plainTextContent, htmlContent);
var response = await client.SendEmailAsync(msg);
```

Figuur 45: API key nodig en plaatsen in config file Azure Functions

### MailKit

- Alternatief voor SendGrid library (geen mailservice)
- Versturen van mails via Mailkit package
- SMTP mogelijkheden

#### 4.10.3 Scenario 3: We vullen een Azure Storage Queue voor verwerking door Azure Functions



Figuur 46: We vullen een wachtrij voor verwerking door Azure Functions

## Queue Trigger

- Functie zal actief worden bij ontvangst van een bericht op de Queue
- Ideaal voor het bufferen bij pieken
- Eenvoudig in gebruik
- In Visual Studio:
  - Nieuwe Azure Function ‘Queue Trigger’ met:
  - Connectionstring = de connection key die bij de Queue storage hoort
  - Queue name = naam van de wachtrij waarvan je berichten wenst te ontvangen

## Table Storage

- We ontvangen bericht en we schrijven naar Azure Table Storage
- We maken gebruik van de Nuget package ‘Microsoft.Azure.Cosmos.Table’
- We gaan temperatuur wegschrijven naar Azure Table Storage

```
[FunctionName("SensorQueue")]
0 references
public static async Task SensorQueue([QueueTrigger("sensorlogs", Connection = "StorageAccount")] string myQueueItem, ILogger log)
{
    SensorLog temperatureLog = JsonConvert.DeserializeObject<SensorLog>(myQueueItem);
    if (temperatureLog.Temperature > 10)
    {
        Microsoft.Azure.Cosmos.Table.CloudStorageAccount storageAccount = Microsoft.Azure.Cosmos.Table.CloudStorageAccount.Parse(Environment.GetEnvironmentVariable("StorageAccount"));
        Microsoft.Azure.Cosmos.Table.CloudTableClient tableClient = storageAccount.CreateCloudTableClient(new Microsoft.Azure.Cosmos.Table.TableClientConfiguration());
        Microsoft.Azure.Cosmos.Table.CloudTable table = tableClient.GetTableReference("logs");
        SensorLogEntity entity = new SensorLogEntity()
        {
            PartitionKey = temperatureLog.Location,
            RowKey = Guid.NewGuid().ToString(),
            Temperature = temperatureLog.Temperature.ToString()
        };
        Microsoft.Azure.Cosmos.Table.TableOperation insertOrMergeOperation = Microsoft.Azure.Cosmos.Table.TableOperation.InsertOrMerge(entity);
        Microsoft.Azure.Cosmos.Table.TableResult result = await table.ExecuteAsync(insertOrMergeOperation);
    }
    log.LogInformation($"C# Queue trigger function processed: {myQueueItem}");
}
```

Figuur 47: Temperatuur wegschrijven naar Table Storage

```
public class SensorLog
{
    1 reference
    public string Location { get; set; }
    2 references
    public decimal Temperature { get; set; }
}

4 references
public class SensorLogEntity : TableEntity
{
    1 reference
    public SensorLogEntity()
    {
    }

    0 references
    public SensorLogEntity(string location, string id)
    {
        PartitionKey = location;
        RowKey = id;
    }
    1 reference
    public string Temperature { get; set; }
}
```

Figuur 48: Objecten voor Table Storage moeten erven van TableEntity

## Hoe de Queue vullen? Meerdere mogelijkheden:

- Via website formulier
- Mobile app
- Python script via Raspberry Pi
- ...

- <https://docs.microsoft.com/en-us/azure/storage/queues/storage-quickstart-queues-python>

```

import os, uuid
import json
import base64
from azure.storage.queue import QueueServiceClient, QueueClient, QueueMessage
connect_str = "DefaultEndpointsProtocol=https;AccountName=theorieles3demosstorage;Ac
try:
    print("Azure Queue storage v12 - Python quickstart sample")
    queue_client = QueueClient.from_connection_string(connect_str, "sensorlogs")
    dict = {'temperature': 13}
    message = json.dumps(dict)
    message_bytes = message.encode('ascii')
    base64_bytes = base64.b64encode(message_bytes)
    base64_message = base64_bytes.decode('ascii')
    queue_client.send_message(base64_message)
except Exception as ex:
    print('Exception:')
    print(ex)

```

Figuur 49: Python script die queue opvult

## 4.11 Good practices

- Iedere Azure Function heeft zijn eigen taak
- Veelgemaakte fout: 1 trigger die alles doet (bv: HTTP trigger die file upload doet en daarna ook mail verstuurt)
- Beter: HTTP trigger ⇒ File naar Blob ⇒ Blob Trigger ⇒ Bericht op Queue ⇒ Queue Trigger ⇒ Mail sturen
  - 3 triggers
  - Schaalbaar
  - Hogere performantie

## 4.12 Configuratiefiles lokaal of in de cloud?

The screenshot shows the Azure Functions Configuration blade. On the left, there is a local JSON configuration file:

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "DefaultEn",
    "FUNCTIONS_WORKER_RUNTIME": "dotn",
    "StorageAccount": "DefaultEndpoi",
    "SENDGRID_API_KEY": "SG.FKKTFsIFS"
  }
}
```

On the right, the 'Application settings' tab of the configuration blade is shown, listing environment variables:

Name	Value
APPINSIGHTS_INSTRUMENTATIONKEY	Hidden value
APPLICATIONINSIGHTS_CONNECTION_STRING	Hidden value
AzureWebJobsStorage	Hidden value

Figuur 50: Lokaal (links) VS in de cloud (rechts)

## 4.13 Samenvatting

- Welke zijn de verschillende soorten Azure Storage en wat is hun doel?
- Wat is Azure Blob storage en wanneer gebruik je deze?
- Wat is Azure Storage Queues en wanneer gebruik ik deze?
- Wat is Azure Table Storage en wanneer gebruik ik deze?
- Wat zijn partitions in Azure Table Storage?
- Hoe werken Azure Functions Blob Triggers?
- Waarvoor kan ik Azure Functions Blob Trigger gebruiken?

- Wat is SendGrid?
  - Good practices

## 5 Azure Security

### 5.1 Governance

- Proces (framework, regels) om uw Azure account, applicaties en resources te beheren en te controleren
- Development team moet dit proces volgen zodat alles onder controle blijft
  - Subscription management
  - Cost management
  - Security
  - Resources Consistency
  - Identity Baseline
  - Deployment acceleration
- Subscription Management
  - Niet iedereen kan/mag subscriptions aanmaken (billing management)
  - Niet iedereen mag/kan de kosten zien
  - Via RBAC kan je rollen instellen wie wat mag zien

#### 5.1.1 Cost management

- Beheren van de kost
- Voorspelen van de kost is mogelijk
- Triggers instellen als de kosten te hoog zouden worden
- Maak gebruik van de Azure Calculator om een inschatting te maken

#### 5.1.2 Security

- Niet iedereen mag toegang hebben tot de subscription data
- Aanmaken van resource moet volgens bepaald proces verlopen
- Azure Policy kan ervoor zorgen dat regels automatisch toegepast worden
- Zeer belangrijk voor compliance

#### 5.1.3 Resource consistency

- Consistentie in het uitrollen van applicaties vb: naamgeving
- Zorgt ervoor dat iedereen zijn weg kan vinden binnen de omgeving
- Kunnen we terug opleggen via Azure Policy

#### 5.1.4 Identity Baseline

- Gaat samen met Security Policy
- Definieren van authentication en authorization requirements doormiddel van Azure AD

#### 5.1.5 Deployment Acceleration

- Het uitrollen van resources & services via scripts en op een consistentie manier

#### 5.1.6 Azure Blue Prints

- Documentatie en scripts om Azure Governance op een goede manier toe te passen
- <https://learn.microsoft.com/nl-nl/azure/governance/blueprints/overview>

## 5.2 Azure KeyVault

### 5.2.1 Probleem

- Confidetiale informatie geheim houden
  - Connectionstrings naar databases, storage buckets, ...

- API keys
- Logins van FTP Server of mailservers
- Certificaten
- Encryptionkeys
- Veel developers stoppen deze info in configuratie files
- Gevaar onstaat als we deze gaan inchecken in source control
- Mensen mailen deze informatie ook rond
  - Wat bij een hack van een mailbox?
- Per ongeluk streamen tijdens video call met opname...

### 5.2.2 Azure KeyVault

- Kluis waarin we confidentiële informatie kunnen opslaan
  - Connectionstrings, logins van services, API Keys, ...
- Deze kluis staat in de cloud en niet op ons systeem of op onze servers
- De informatie kunnen we raadplegen vanuit onze code op een veilige manier
- 3 belangrijke onderdelen
  - Secrets Management
  - Key Management
  - Certificate Management
- Hardware Security Module (HSM) is hardware die keys voor authentication and crypto keys kan beheren
- International standard FIPS 40
- Enkel Premium Tier

#### 5.2.2.1 *Secrets Management*

- API Key, Connectionstrings, .. alles wat geheim is en die je nodig hebt in je applicatie
- Worden encrypted opgeslagen
- Doelstelling is deze los te koppelen van de applicatie die we bouwen
- Enkel wie de juiste Azure rechten heeft kan deze **lezen** of **schrijven** in de KeyVault
- Version management
- Altijd in Software opgeslagen geen HSM

#### 5.2.2.2 *Keys Management*

- Encryption Keys (public/private)
- Storage Account Keys with auto sync
- Key rotation is mogelijk
- Version Management
- Opslag in software Of HSM (premium tier)

#### 5.2.2.3 *Certificates*

- X.509 Certificaten
- Lifecycle van certificaten beheren
- HTTPS Certificaten beheren

### 5.2.3 Uitlezen van Azure KeyVault

- We voegen volgende nuget package toe
  - Dotnet add package Azure.Security.KeyVault.Secrets –version 4.3.0
- Het enige wat we nodig hebben is de URL van de KeyVault

#### 5.2.3.1 *KeyVault Security*

- Vault Access Policy (oude systeem)
- Azure Role Based Access Control
  - Nieuwe
  - Gradueel controle over rechten

- Rechten mogelijk specifiek voor
  - 1 secret
  - 1 key
  - ...

Toch is er nog een probleem. Developer kan tijdens het **debuggen** van de applicatie de waardes uit de KeyVault lezen. Oplossing 2 KeyVaults:

- 1 development
- 1 production

Zelfde keyname gebruiken maar dan moet je wel alles dubbel hebben SQL, Sendgrid, storage

## 5.3 Azure Active Directory (AzureAD)

### 5.3.1 Probleem

- Opslaan van login/paswoord combinatie is gevaarlijk
- Als developer of bedrijf deze verantwoordelijkheid drage is niet eenvoudig en sterk af te raden!!!
- Bij een hack of datalek straalt dit negatief af op het bedrijf (meldingsplicht)
- We moeten allemaal zeer veel login/paswoorden bijhouden daarom kiest men soms dezelfde paswoordem...

### 5.3.2 Azure AD

- Wat?
  - A multitenant service voor identity and access management in de Cloud
  - Global scale, reliability van available
  - 99.99 SLA
- Wat kan je ermee doen?
  - Centraal beheren van gebruikers (intern en extern)
    - Identity
    - Toegang tot resources
  - Single Sign On (SSO)
  - MFA
- Wie gebruikt dit?
  - Office 365
  - Azure Portal
  - Veel andere SaaS applicaties
- Soorten
  - Azure AD B2B Collaboration
    - Inviteren van gebruikers in je eigen AD
    - Delen van applicaties met deze gebruikers
    - Vooral in scenario waar externe mensen eigen (interne) beschrijfsapplicaties moeten gebruiken. Vb: consultant moet toegang hebben tot ERP systeem van bedrijf of waar je eigen mensen toegang wil geven tot applicaties in de Cloud
  - Azure AD B2C
    - Ideaal voor public web app's, Spa's, etc...
    - Ondersteuning voor Microsoft Account, Gmail, Twitter, ...
    - Eigen login mogelijk → dataopslag in AzureAD we moeten dus zelfs niks bijhouden
    - Controle over de login flow en schermen is mogelijk (vb branding)
    - Vooral gebruikt bij publieke applicatie's en voor mensen die niet in je Active Directory zitten
  - Alternatieven buiten Azure
    - Okta
    - Auth0

- AWS IAM Identity Center
- Google Cloud Identity

## 5.4 Azure Managed Identity

### 5.4.1 System Assigned Managed Identity

- Verbonden aan de resource
- Een resource kan maar 1 **System Assigned Managed Identity** hebben
- Een **System Assigned Managed Identity** kan maar aan **1** resource gekoppeld worden
- Verwijderen we de resource dan is ook de System Managed Identity weg

## 5.5 User Assigned Managed Identity

- De **User Assigned Managed Identity** is zelf een resource
- Niet gekoppeld aan een andere resource dus eigen life cycle
- We kunnen deze koppelen aan meerdere andere resources met hun rollen
- Je kan deze identities niet beheren
- Beheer is managed door Azure AD

### 5.5.1 Hoe implementeren

- Voeg een nuget package toe voor identity
  - Dotnet add package Azure.Identity –version 1.6.1
- Nu moeten we nog zorgen dat ons system weet wie de ingelogde gebruiker is.
  - Open de command prompt en tik **az login**
  - De browser zal openen en inloggen in de portal van Azure
- Lokaal op je pc
  - De gebruiker van de Azure Account moet je toevoegen aan de resource die je wenst te gebruiken. We kunnen rechten nauwkeurig afregelen en toestaan per gebruiker

## 5.6 Samenvatting

- Wat is Azure Governance
- Wat is KeyVault en waarom gebruiken we dit?
- Welke 3 services bevat KeyVault en wat lossen ze op?
- Wat is Azure AD en welke versies zijn er en wanneer gebruiken we deze?
- Wat is System Managed Identity?
- Wat is User Managed Identity?

# 6 Cosmos DB

## 6.1 Relationale databases

= Opslag van data op gestructureerde manier

- We slaan data op in verschillende tabellen
  - Via normalisatie bepalen we de tabellen die we nodig hebben
  - We stellen schema op waarin de data moet passen
- Een tabel bestaat uit
  - Rijen (record), die de data voorstellen
  - Kolommen (welke info slaan we op)
- We kunnen tabellen met elkaar verbinden via relaties
  - We maken hiervoor gebruik van vreemde sleutels (kolom die zal verwijzen naar record in een andere tabel)

- We maken gebruik van de taal SQL voor
  - Data opvragen (SELECT)
  - Data toevoegen (INSERT)
  - Data wijzigen (UPDATE)
  - Data verwijderen (DELETE)
- Via DDL kunnen we tabellen aanmaken (CREATE TABLE)
- Wij kennen:
  - SQL Database (Op Azure)
  - MySQL Lokaal op de RPi maar ook op de Azure Cloud

### 6.1.1 Waarom ontstaan?

- Ontstaan in client/server periode
- Er was nog geen sprake van Internet, Cloud, Mobile data etc . . .
- Was vooral gemaakt om op 1 server te werken, meestal centraal in bedrijf
- Enige manier om te schalen was krachtiger CPU/Network/Storage ⇒ Scale up
- **Opgepast:** blijft zeer belangrijk binnen bedrijven, veel software moet integreren met bestaande systemen waaronder SQL

## 6.2 NoSQL databases

= ‘Non-SQL’ of ‘non-relational SQL’

- Databases die gemodelleerd zijn zonder tabelrelaties te gebruiken

### 6.2.1 Waarom?

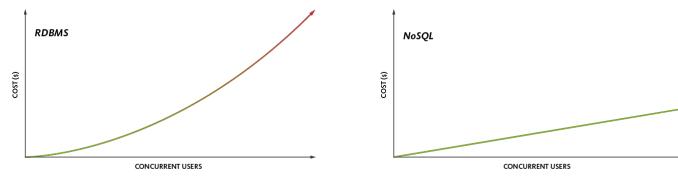
- Hoeveelheid data neemt enorm toe jaar na jaar
  - Social Networks
  - IoT Devices
  - AI (Artificial Intelligence) & Deep Learning systemen
- Concurrency: veel gebruikers op hetzelfde moment: 10000 tot miljoenen
- Globaal bereikbaar en responsive
- Connectivity: veel gebruikers maar ook toestellen schrijven nu data weg of moeten data kunnen lezen
- Kan verschillende soorten data opslaan: structured, semi-structured en unstructured data
- Horizontal scaling, we kunnen eenvoudige meerdere servers toevoegen

⇒ **Bovenstaande zaken zijn niet altijd mogelijk met relationele databases**

### 6.2.2 Scaling

- Met eenvoudige hardware kunnen we scale-out doen door gewoon servers bij te plaatsen
- De hardware mag goedkoop zijn

- Geen downtime
- Eenvoudig te installeren en configureren

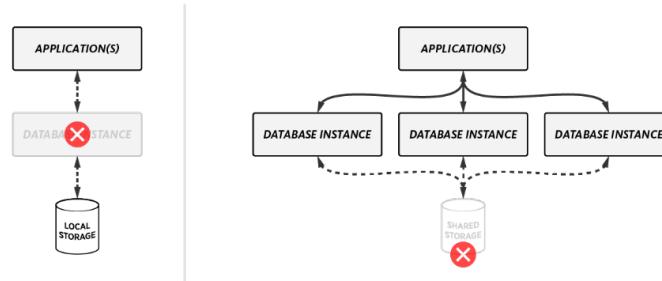


Figuur 51: Relationale DBMS vs NoSQL cost per concurrent user

### 6.2.3 Availability and always-on

#### Bij Relationale (RDBMS)

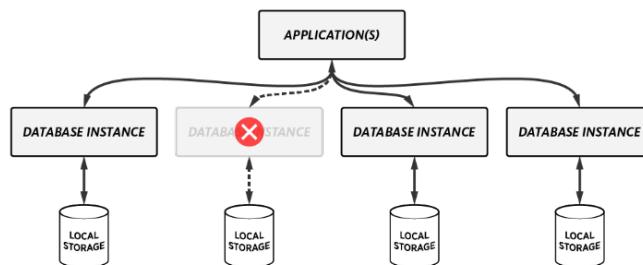
- Gedeeltelijke storage failure
- ⇒ Volledige applicatie down



Figuur 79: Beschikbaarheid bij RDBMS: bij beide scenario's een single point of failure

#### Bij NoSQL

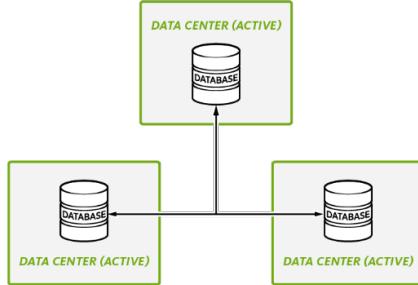
- Data zal op verschillende partities staan
- Geen gedeelde data
- Data zal naar alle storage weggeschreven worden, high availability (kan ook met RDBMS maar complex en extra software)
- We spreken hier over **nodes**



Figuur 52: Beschikbaarheid bij NoSQL

#### 6.2.4 Global deployment

- Database in verschillende datacenters
- Zo dicht mogelijk bij de klant draaien ⇒ latency zo < mogelijk
- Automatische replicatie van data tussen de verschillende datacenters



Figuur 53

#### 6.2.5 Gedistribueerde systemen

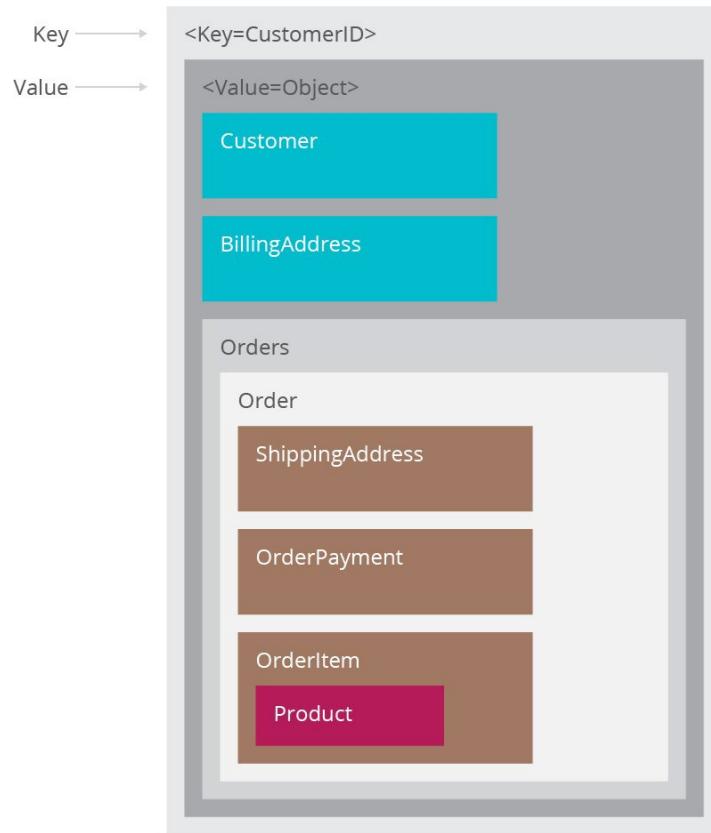
- NoSQL is een gedistribueerd systeem
- We hebben meerdere servers (we noemen dit ook soms nodes)
- Deze nodes communiceren met elkaar over een netwerk
- Data zal op meerdere machines staan
- We merken dit niet, is transparant voor de applicatie

### 6.3 Soorten NoSQL Databases

1. Key/Value database (bv Azure table storage)
2. Document database
3. Column store (zien we niet)
4. Graph (zien we niet)

#### 6.3.1 Key/Value

- Meest eenvoudige NoSQL database
- Bestaat uit een unieke Key
- Voor iedere Key is er één waarde
- Value kan JSON zijn of ander document
- De NoSQL Database weet niet wat soort data het is, de database slaat enkel op, we moeten dus GEEN datatype opgeven
- De applicatie die data zal ophalen op basis van de Key moet de waarde kunnen interpreteren **Wanneer gebruiken?**
- Sessie-informatie gebruiker, user profiles, shopping basket
- Data waar je minder moet in gaan zoeken via queries • Data zonder relaties



Figuur 54: Key-value database

### 6.3.2 Document databases

- Basis is een document, geen record
- Het formaat is meestal JSON, maar XML of BSON (binary JSON) zijn ook mogelijk
- Het document zal zichzelf beschrijven, geen schema nodig zoals bij een SQL database
- Niet iedere document moet volgens hetzelfde schema zijn
- Er is een query taal om de data op te vragen en te doorzoeken **Wanneer gebruiken?**
- Real-time analytics data
- IoT applications
- Vooral data die niet zoveel zal wijzigen, maar tegenwoordig meer en meer voor alle soorten data



Figuur 55: Document Database

### 6.3.3 Column store

- Dataopslag zal niet gebeuren op basis van records maar op basis van kolom
- Veel sneller bij zeer complexe query's
- Veel gebruikt voor data analyse systemen

Rowid	EmpId	Lastname	Firstname	Salary
001	10	Smith	Joe	40000
002	12	Jones	Mary	50000
003	11	Johnson	Cathy	44000
004	22	Jones	Bob	55000

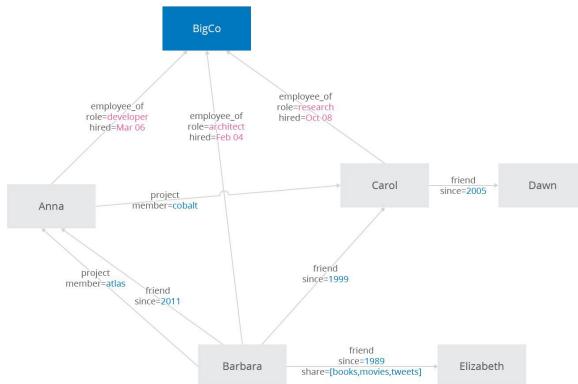
```
[001:10,Smith,Joe,40000;
002:12,Jones,Mary,50000;
003:11,Johnson,Cathy,44000;
004:22,Jones,Bob,55000;

10:[001,12:002,11:003,22:004;
Smith:[001],Jones:[002],Johnson:[003],Jones:[004];
Joe:[001],Mary:[002],Cathy:[003],Bob:[004];
40000:[001],50000:[002],44000:[003],55000:[004];
```

Figuur 56: Column store

### 6.3.4 Graph Database

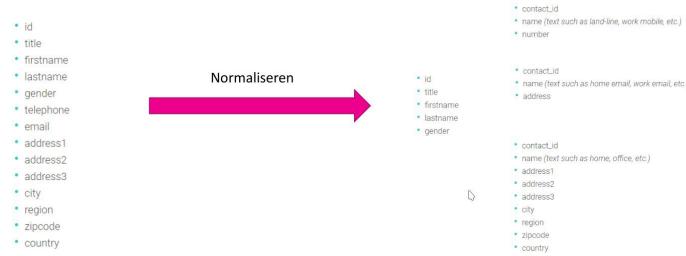
- Opslaan van entiteit en relaties tussen verschillende entiteiten •  
Een entiteit is een soort node (object instantie) met properties
- Een relatie bevat ook properties, vb: friend relation, project relation . . .
- De kracht zit in de relaties die je gaat opstellen tussen de entiteiten **Wanneer gebruiken?**
- Social networking apps
- Spatial data apps • Routing apps



Figuur 57: Graph database

## 6.4 Normalisatie

### 6.4.1 Voorbeeld SQL



Figuur 58: Normaliseren van data naar tabellen voor gebruik in SQL

- Normalisatie lukt wel maar blijft lastig
- Er zullen zaken bijkomen, vb. bedrijf, job role, . . .
- We moeten terug schema aanpassen
- We kunnen niet alles voorzien
- De data zit ook verspreid over de database ⇒ complexe SELECT en JOIN nodig
- Full Text search is lastig, we moeten alles kolommen checken bij het zoeken

### 6.4.2 Voorbeeld NoSQL

- Een record zal een document worden
- Makkelijk nieuwe velden toepassen
- Formaat is JSON
- We slaan data op als JSON in de database zelf
- We moeten data niet meer uitlezen en INSERT opbouwen

**Nadeel:** dubbele data, geen relaties tussen data

```
{
  name: [
    "Billy", "Bob", "Jones"
  ],
  company: "Fake Goods Corp",
  jobtitle: "Vice President of Data Management",
  telephone: {
    home: "0123456789",
    mobile: "9876543210",
    work: "2244668800"
  },
  email: {
    personal: "bob@myhomeemail.net",
    work: "bob@myworkemail.com"
  },
  address: {
    home: {
      line1: "10 Non-Existent Street",
      city: "Nowhere",
      country: "Australia"
    }
  },
  birthdate: ISODate("1980-01-01T00:00:00.000Z"),
  twitter: '@bobsfakeaccount',
  note: "Don't trust this guy",
  weight: "200lb",
  photo: "52e86ad749e0b817d25c8892.jpg"
}
```

Figuur 58: Voorbeeld NoSQL

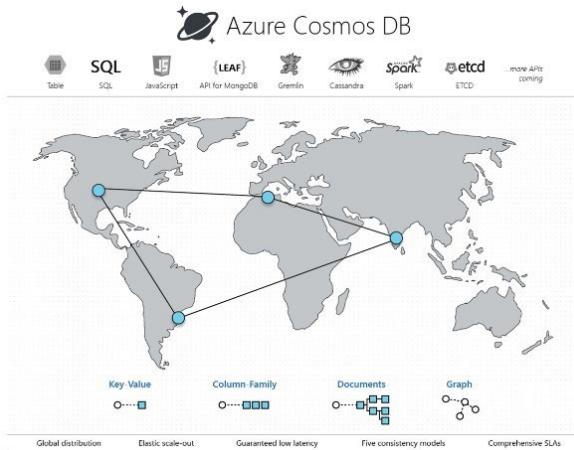


Figuur 59: NoSQL technologieën

## 6.5 CosmosDB

= CosmosDB is managed NoSQL database service op Azure

- Geen onderhoud
- Geen patching
- Eenvoudig op te zetten
- Pay what you use • Krachtige API's

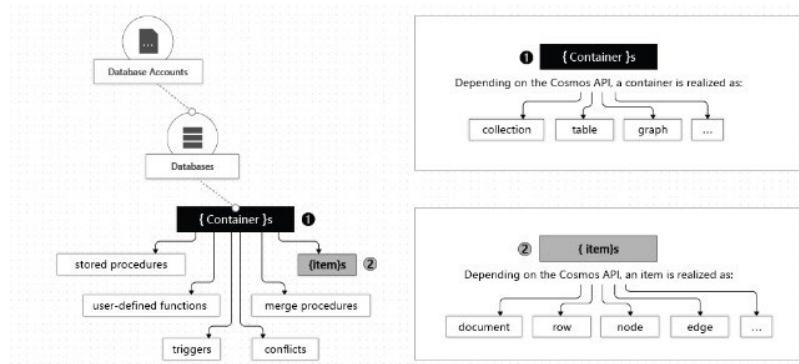


Figuur 60: Azure Cosmos DB

- Globaal gedistribueerde database
  - Op eenvoudige manier de data zo dicht mogelijk bij de klant brengen
  - Op eenvoudige manier activeren in datacenter dicht bij de gebruiker
  - Geen complex configuratie voor nodig
- Unlimited scalability
  - Eenvoudig up- and down scaling wanneer je maar wilt
- Low latency
- Meerdere data modellen en API's beschikbaar
  - Document DB API
  - MongoDB API
  - Gremlin API – ...
  - Zeer belangrijk bij het migreren van bestaande toepassingen naar CosmosDB
- Service Level Agreement (SLA)
  - = uptime-garantie aangeboden door de vendor (hier: Azure)
  - Vanaf 99,99% in een region
  - Meerdere regions = hogere SLA

### 6.5.1 CosmosDB API

- Cosmos DB Account
- Database
  - Groep die meerdere Containers (Collection) bevat
- Container == Tabel (in klassieke SQL)
  - Bevat een verzameling van verschillende JSON documenten
- Item == Record (in klassieke SQL)
  - JSON content



Figuur 61: Structuur CosmosDB API

### 6.5.2 cosmosDB API Account

- Wij gebruiken ‘SQL API’
- Capacity Node:
  - Kies hier ‘serverless’, we betalen enkel voor wat we gebruiken
  - Kies ‘non production’ om te testen

Home > Resource groups > VoorbeeldExamenOefening > New > Azure Cosmos DB >

### Create Azure Cosmos DB Account

For a limited time, create a new Azure Cosmos DB account with multi-region writes in any region, and receive up to 33% off for the life of your account. [Try it for free](#)

[Basics](#) [Networking](#) [Backup Policy](#) [Encryption](#) [Tags](#) [Review + create](#)

Azure Cosmos DB is a globally distributed, multi-model, fully managed database service. [Try it for free](#), for 30 days with unlimited renewals. Go to production starting at \$24/month per database, multiple containers included. [Learn more](#)

**Project Details**  
Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* [Azure for Students](#)  
Resource Group \* [VoorbeeldExamenOefening](#) [Create new](#)

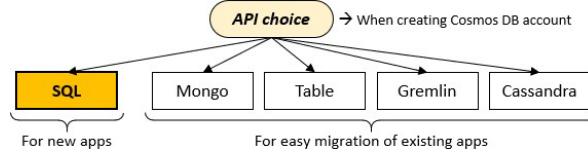
**Instance Details**

Account Name \*  Enter account name  
API \* [Core \(SQL\)](#)  
Notebooks (Preview) [On](#) [Off](#)  
Location \* [\(US\) West US](#)  
Capacity mode [Provisioned throughput](#) [Serverless \(preview\)](#) [Learn more about capacity mode](#)  
Account Type [Production](#) [Non-Production](#)

Figuur 62: CosmosDB Account aanmaken in Azure Portal

### 6.5.3 Kiezen van een API

- Bepalen bij aanmaak Account
- Je kan deze niet wijzigen achteraf
- Wij zullen SQL API gebruiken voor document databases
- **Opgepast:** SQL API is niet zo krachtig als de SQL taal die we kennen



Figuur 63: Kiezen van een API

#### 6.5.4 Container toevoegen

= vergelijkbaar met de klassieke tabel uit RDBMS

- Unieke database naam
- Unieke container naam (soort table)
- Partition Key (zelfs concept als table storage)

**Add Container**

\* Database id ⓘ  
 Create new  Use existing

\* Container id ⓘ

\* Partition key ⓘ  
  
 My partition key is larger than 100 bytes

Unique keys ⓘ

Figuur 64: Toevoegen van een container

#### 6.5.5 Manueel data toevoegen

testdietercosmos | Data Explorer

SQL API

DATA

mctshop

customers

id	city
1	Kortrijk

Items

```

SELECT * FROM c
    
```

1
 "id": "1",
 "name": "De Preester",
 "firstName": "Dileter",
 "city": "Kortrijk",
 "id": "14c4a11c14c4aaaaaaad++",
 "level": "db:/p44+A==coll/24+4Jtvl4+docs/rP4+A3tvl4Caaaaaaaah++/",
 "type": "\\"c300ca31-0000-0000-0000-5f928ee40000\\",
 "attachments": "",
 "size": 169344935

Figuur 65: Toevoegen van data

## 6.5.6 Firewall

The screenshot shows the Azure portal interface for managing a Cosmos DB account named 'testdiertercosmos'. Under the 'Firewall and virtual networks' tab, there's a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Notifications, Data Explorer, Settings, Default consistency, Backup & Restore, Firewall and virtual networks (which is selected), Private Endpoint Connections, and VNet. In the main area, there's a search bar and a 'Virtual networks' section with tabs for Virtual Network, Subnet, and Address range. Below that is a 'Firewall' section where you can add IP ranges or subnet ranges. An IP range '172.119.127.71' is listed under 'IP (Single IPv4 or CIDR range)'. There are also sections for 'No IP range configured', 'Exceptions' (with 'Accept connections from within public Azure datacenters' checked), and 'Allow access from Azure Portal'.

Figuur 66: Firewall instellen

## 6.5.7 Aanspreken vanuit Azure Functions met .NET

- Nuget Package: Microsoft.Azure.Cosmos
- 2 belangrijke parameters: URI en PRIMARY KEY

The screenshot compares the parameters in the Azure Portal with the corresponding code in C#. Both show the same values for the primary and secondary keys, as well as the connection strings. The C# code uses environment variables to store the connection string.

```
"CosmosKey": "WkRQ9HGxLljchlwIP46166wloMEIp...  
"CosmosEndPoint": "https://cloud1.documents..
```

Figuur 67: Parameters in Azure Portal en in C#

### Stap 1: Connectie maken met CosmosDB

- CosmosClient object aanmaken met parameters uit local.settings.json
- Connectionstring meegeven
- ConnectionMode op Gateway (Alternatief is direct maar werkt niet in Howest)

```
CosmosClientOptions clientOptions = new CosmosClientOptions();  
clientOptions.ConnectionMode = Microsoft.Azure.Cosmos.ConnectionMode.Gateway;  
  
cosmosClient = new CosmosClient(Environment.GetEnvironmentVariable("CosmosConnectionString"), clientOptions);
```

Figuur 68: Verbinden met database via Environment Variables

### Stap 2: Model van data die we wensen weg te schrijven

- [JsonProperty] gebruiken om lowercase property te bekomen in de database
- Id voorzien die uniek is
- PartitionKey moet ook aanwezig zijn in model

```

8 references
public class TemperatureLog
{
    [JsonProperty("deviceid")]
    2 references
    public string DeviceId { get; set; }
    [JsonProperty("temperature")]
    3 references
    public int Temperatuur { get; set; }
    [JsonProperty("id")]
    1 reference
    public string Id { get; set; }
}

```

Figuur 69: Datamodel

### Stap 3: Deserialisatie

```

string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
var tempLog = JsonConvert.DeserializeObject<TemperatureLog>(requestBody);

```

Figuur 70: Deserialiseren van de JSON naar een object van het Datamodel

### Stap 4: Wegschrijven naar database

- Ophalen database en container waarnaar we wensen weg te schrijven
- Id invullen met unieke waarde
- CreateItemAsync aanroepen met model

```

CosmosClientOptions clientOptions = new CosmosClientOptions();
clientOptions.ConnectionMode = Microsoft.Azure.Cosmos.ConnectionMode.Gateway;

cosmosClient = new CosmosClient(Environment.GetEnvironmentVariable("CosmosConnectionString"), clientOptions);

```

Figuur 71: Wegschrijven van data

### Stap 5: Bekijk data in Azure Portal of in Azure Storage Explorer

	id	deviceid
1	c7b553..._podiet	"podiet",
2	681c13..._podiet	"Temperatuur": 7,
3	"_id": "881199-1daa-407d-874a-ea594f51824f",	"_etag": "'12c3031ed4-0000-0000-5d33db200000'",
4	"_etag": "'886d40dca0585aa3a-0000-0000-5d33db200000'",	"_ts": 1574170834
5	0d0bcf..._podiet	
6	6022bc..._podiet	
7	0605d9..._podiet	
8	93b0e2..._podiet	
9	295d00..._podiet	
10	564979..._podiet	

Figuur 72: Azure Portal / Azure Storage Explorer

### Stap 6: Opvragen van data

```

QueryDefinition query = new QueryDefinition("SELECT * FROM logs f WHERE f.temperature < @maxtemp")
    .WithParameter("@maxtemp", maxTemp);

List<TemperatureLog> results = new List<TemperatureLog>();
FeedIterator<TemperatureLog> resultSetIterator = container.GetItemQueryIterator<TemperatureLog>(query, requestOptions: new QueryRequestOptions() { PartitionKey = new PartitionKey(deviceId) });
while (resultSetIterator.HasMoreResults)
{
    FeedResponse<TemperatureLog> response = await resultSetIterator.ReadNextAsync();
    results.AddRange(response);
    if (response.Diagnostics != null)
    {
        Console.WriteLine($"\\nQueryWithSqlParameters Diagnostics: {response.Diagnostics.ToString()}");
    }
}

```

Figuur 73: Opvragen van data

### 6.5.8 CosmosDB extra's

- Azure Search
  - Krachtige zoekmachine boven CosmosDB
  - Text search
  - Indexing
- Azure Functions
  - CosmosDB Trigger
  - Bij toevoegen documenten ⇒ functie activatie
- Time To Live
  - Automatisch verwijderen van items na x-aantal tijd
  - Container niveau
  - Item niveau

## 6.6 Extra's

- Stored Procedures
  - Functies in de database
  - Schrijven van complexe queries
  - Werken op collection niveau
  - Aanroepen via API of portal
  - Javascript
- UDF (User Defined Functions)
  - Functies die we aanroepen vanuit queries
  - Vb: berekening maken
  - JS
- <https://docs.microsoft.com/en-us/azure/cosmos-db/programming>

```
var taxUdf = {
    id: "tax",
    serverScript: function tax(income) {

        if(income == undefined)
            throw 'no input';

        if (income < 1000)
            return income * 0.1;
        else if (income < 10000)
            return income * 0.2;
        else
            return income * 0.4;
    }
}
```

Figuur 74: Voorbeeld UDF aanmaken

```
var query = 'SELECT * FROM TaxPayers t WHERE udf.tax(t.income) > 20000';
```

Figuur 75: UDF uitvoeren in query

## 7 Azure Service bus

- Azure Service Bus
- Azure Event Hub
- Azure Event Grid

### 7.1 Azure Service Bus

= Azure Service Bus is een volledig beheerde berichtenmakelaar voor ondernemingen met berichtenwachtrijen en onderwerpen voor publiceren en abonneren (in een naamruimte). Service Bus wordt gebruikt om toepassingen en services van elkaar te ontkoppelen, wat de volgende voordelen biedt

- Versturen van data tussen verschillende applicaties op basis van berichten
- Een bericht is een soort “container” met daarin data en bevat ook “metadata”
- De data kan gelijk welk soort informatie zijn en kan volgende formaten zijn
  - JSON
  - XML
  - Apache Avro
  - Plain Text
- Eenvoudige scenario versturen (**producer**) van berichten vb orders, stock, veranderingen et... naar wachtrij om dan opgepikt te worden door andere applicatie (**consumer**)
- Decoupled Applications, schaalbaarheid en beschikbaarheid verhogen, producer en consumer moeten niet op hetzelfde moment online zijn
- Topic & Subscriptions, 1 consumer zal naar meerdere topics (queue) schrijven
- Transactions: Bericht ophalen, verwerken en verwijderen van de queue allemaal in 1 transaction

#### 7.1.1 Queues

- Wachtrij in de Cloud die berichten zal opslaan tot een ontvanger deze zal uitlezen, verwerken en verwijderen uit de queue.
- Message zijn gesorteerd en krijgen een timestamp bij aankomst in de wachtrij
- Er is terug 3x opslag om niks te laten verloren gaan bij problemen in een datacenter
- We gebruiken dit meestal in **point to point communicatie, 1 verzender, 1 ontvanger**

#### 7.1.2 Topics

- Hier hebben we een sender die naar 1 of meerdere topics een bericht zal sturen
- Er kunnen 1 of meerdere subscriptions gekoppeld zijn aan een topic
- Een subscriber (receiver) zal een copy van het bericht ontvangen
- Er kunnen regels ingesteld worden op een topic

#### 7.1.3 Namespaces

- Dit is een container waarin queues en topics actief zijn, er kunnen er meerdere per namespace zijn
- Een namespaces kan actief zijn over verschillende datacenter en availability zones
- **Message Session:** garantie dat berichten volgens FIFO concept opgeslagen zijn
- **Auto Forwarding:** automatisch doorsturen van berichten kunnen worden of afgeleverd worden
- **Dead Lettering Queue:** bevat berichten die niet verwerkt kunnen worden of afgeleverd worden
- **Scheduled Delivery:** uitgesteld verwerken van berichten
- **Message deferral:** langs de kant zetten van berichten tot de ontvanger deze kan verwerken. Dit kan nodig zijn als de ontvanger even offline is
- **Filtering and actions:** subscribers kunnen opgeven welke berichten ze al dan niet wensen te ontvangen van een topic
- **Auto-delete on idle:** automatisch queue verwijderen als deze 5 min idle is
- **Duplication detection:** detecteren van dezelfde dubbele berichten
- **Transactions:** enkel berichten verwijderen als andere taken in de transactie goed verlopen zijn

- **Geo disaster recovery:** we kunnen Azure Service Bus clusteren over verschillende datacenters op verschillende datacenters op verschillende locaties om de downtime zo minimaal te houden bij problemen
- **Standard protocols:** standaard is het protocol AMQP 1.0, dit is een open ISO/IEC standaard wat zorgt dat we vlot kunnen samenwerken met andere producten zoals ActiveMQ of RabbitMQ
- Azure Service Bus Premium is compatible met Java Message Service JMS 2.0

#### 7.1.4 Implementeren?

- Azure Service Bus SDK voor .NET, Python, Javascript, ...



Fig 76

#### 7.1.5 Verbinden?

- Connectionstring maken met beperkte rechten
- Managed identity

### 7.2 Azure Event Hubs

= Azure Event Hubs is een platform voor het streamen van big data en een dienst voor het opnemen van gebeurtenissen. Het kan miljoenen gebeurtenissen per seconde ontvangen en verwerken. Gegevens verzonden naar een Event Hub kan worden getransformeerd en opgeslagen met behulp van een real-time analyseprovider of batching-/opslagadapters.

- Azure Service waar we miljoenen events per seconde kunnen ontvangen
- Gebruiken we vooral voor:
  - Sensordata
  - Live dashboard data
  - Telemetry data van machines of software
  - Logging van applicaties
- Startpunt van data pipeline, data zal binnenkomen via EventHubs. We noemen dit ook de **event ingestor**
- Verdere werking mogelijk via
  - Azure Streaming Analytics
  - Azure Data Explorer
- Directe opslag naar
  - Azure Storage
  - Azure Datalake
- Basis voor Azure IoT Hub

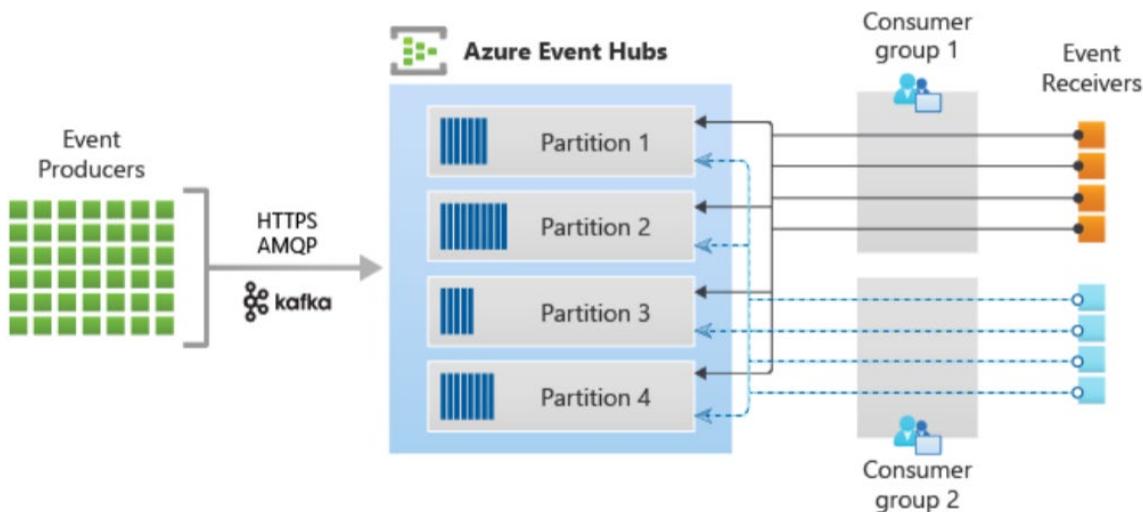


Fig 77

### 7.3 Azure Data Explorer

- Real Time Data Analytics Server op basis van Time Series Database
- Wanneer je moet zoeken in miljoenen berichten per seconde
- Krachtige query taal Kusto
- Dashboarding opties
- Duur !!!

### 7.4 Azure Event Grid

= Event Grid is een zeer schaalbare, serverloze eventbroker die u kunt gebruiken toepassingen integreren met behulp van gebeurtenissen. Evenementen worden door Event Grid geleverd aan abonneebestemmingen zoals toepassingen, Azure-services of een willekeurig eindpunt waartoe Event Grid netwerktoegang heeft. De bron van die gebeurtenissen kan zijn andere applicaties, SaaS-services en Azure-services.

- Via Event Grids kunnen we applicaties binnen Azure met elkaar verbinden
- De applicaties communiceren via Events

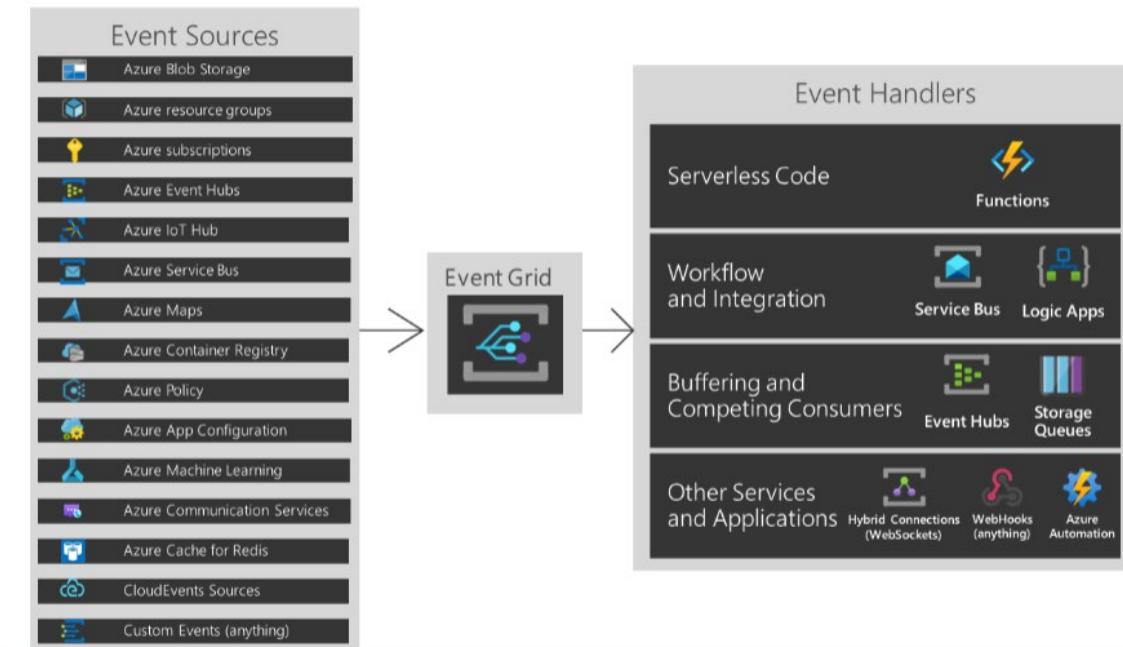


Fig 78

#### 7.4.1 Event source

- Publisher van events naar de Event Grid
  - Onze eigen applicatie kan custom event publishen
  - We kunnen beroep doen op ingebouwde Event publishers in Azure
  - Derde partijen vb Microsoft Graph of SaaS software zoals Teams, ...
- Event Handler (receiver)
  - Azure Functions
  - Webhooks
  - Event Hubs
  - Queues
  - Logic Apps
  - ...

### 7.5 Azure Logic Apps

- Workflow Designer
- We programmeren visueel met blokken de workflow
- Basis zijn connectoren om te verbinden met:
  - Data
  - Webservices
  - Externe services
  - ...

#### 7.5.1 Activeren?

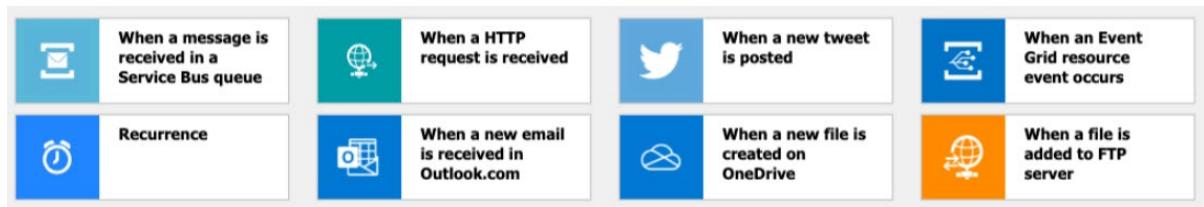


Fig 79

- Geen server nodig (serverless)
- Pay By Use, we betalen per keer Azure de workflow moet uitvoeren (0,000025 cent per run)
- Verbinding met veel andere azure services via connectors

### 7.6 Azure Messaging Services

#### 7.6.1 Event vs Message

##### 7.6.1.1 Events

- Een event is een notificatie die ontstaat als er iets gewijzigd is of iets ontstaan is
- Het maakt niet uit voor de verzender van het event wat er zal gebeuren met het event
- De ontvanger van het event zal beslissen wat er zal gebeuren met het event
- Vb: we melden dat er een file is aangemaakt, maar de file zelf zit NIET in het event

##### 7.6.1.2 Message

- Een message bevat ruwe data aangemaakt door een service of app en die door een ontvanger verwerkt zal worden. De verzender van het bericht gaat er vanuit dat de ontvanger dit bericht zal inlezen en verwerken
- Vb: order

### 7.6.2 Azure event Grid

- Wanneer we berichten tussen Azure Service willen verturen om ze op de hoogte te brengen van veranderingen: vb aanmaken resource group, file drop op een blob, ...
- Zowel zender als ontvanger zijn een Azure Service (of office 365)
- Zeer schaalbaar, goedkoop, serverless

### 7.6.3 Azure Event Hubs

- Wanneer we miljoenen events moeten ontvangen en verwerken en dit in realtime
- We hebben zeer veel clients (10.000+) dit berichten versturen naar Event hub
- We wensen te integreren met data platformen zoals **Azure Data Explorer** of Azure Streaming Analytics
- Wanneer we een data pipeline wensen op te zetten

### 7.6.4 Azure Service Bus

- Wanneer we berichten wensen te verturen tussen verschillende applicaties los van elkaar
- Veel gebruikt in enterprise business systemen voor orders, transactions etc...
- Wanneer we geen duplications mogen hebben, forwarding nodig hebben, of 1 van de andere opties

### 7.6.5 Logic Apps

- Workflow zonder code
- Integraties met andere platformen
  - Office 365, Adobe
  - SAP of andere ERP systemen

Service	Purpose	Type	When to use
Event Grid	Reactive programming	Event distribution (discrete)	React to status changes
Event Hubs	Big data pipeline	Event streaming (series)	Telemetry and distributed data streaming
Service Bus	High-value enterprise messaging	Message	Order processing and financial transactions

Fig 80

## 7.7 Samenvatting

- Wat is Azure Service Bus en waarom gebruiken we dit?
- Wat is Event Hub en wanneer gebruiken we dit?
- Wat is Event Grid en wanneer gebruiken we dit?
- Wat is Logic Apps en wanneer gebruiken we dit?

## 8 IoT Hub

Azure IoT Hub is a fully managed service that enables **reliable** and **secure bidirectional communications** between millions of IoT devices and a **solution back end**

- Managed Service: we moeten niks installeren in de cloud
- Bi-Directional Communication
- **Miljoenen** toestellen
- Support voor meerdere talen (C#, Python, C, NodeJS)
- HTTPS/AMQP/MQTT protocol support

- Versturen telemetrie data ⇒ Device To Cloud
- Ontvangen van commands ⇒ Cloud To Device
- Beheer van devices via **digital twin**
- Queries op devices
- End-to-end security
- Certificaten per device
- TLS support
- X.509 support
- IP Whitelisting/blacklisting van devices
- Firmware/software update support
- Eenvoudiger dan bij MQTT

## 8.1 Werking

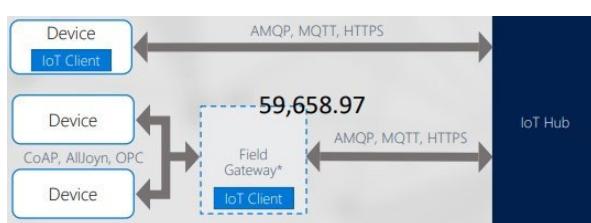
- Device zal rechtstreeks communiceren met IoT Hub
  - Device To Cloud (D2C)
  - Cloud To Device (C2D)
- 3 Protocollen mogelijk:
  - AMQP
  - MQTT (beperkingen t.o.v. eigen MQTT Server)
  - HTTPS
- Meest eenvoudige opstelling
- Hardware moet wel krachtig genoeg zijn

### 8.1.1 Wat als device niet krachtig genoeg is?

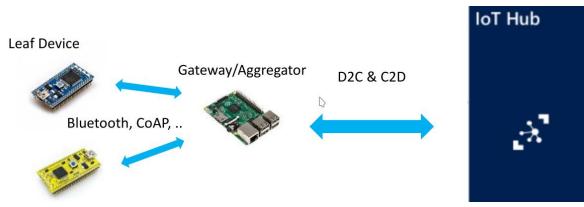
- Bv: Arduino, Mbed Cortex M0,M1, . . . (= "Leaf device")
- Goedkope hardware maar niet altijd direct mogelijk om te connecteren op het internet **Oplossing:**

- **Field gateway gebruiken**

- = Device dat krachtig genoeg is om veilig met het internet te communiceren
  - Kan Raspi zijn, maar ook industriële PC
  - Field gateway zal lokaal met de minder krachtige device praten via CoAP, Bluetooth, AllJoyn,
- ...



Figuur 81: Bovenaan: rechtstreeks communiceren met IoT client, onderaan: via Field gateway naar minder krachtige devices



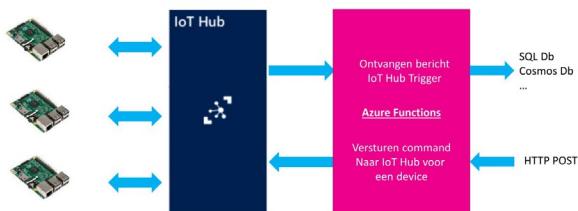
Figuur 82: Opstelling Field gateway

**D2C:** Low powerd device zal communiceren met Raspi (Gateway). Gateway zal data doorsturen naar Cloud.

**C2D:** IoT hub zal commando sturen naar Raspi (Gateway). Raspi zal commando naar juiste device sturen.

## 8.2 Hoe data verwerken die binnenkomt op IoT Hub?

- Streaming analytics
- Message Queues
- Azure Functions
  - Ontvangen van berichten
  - Versturen van berichten (commandos) naar IoT Hub



Figuur 83: Schematische voorstelling

## 8.3 IoT Hub Device Twin

### Probleemstelling:

We hebben 10.000 toestellen verbonden met Azure IoT Hub. Deze toestellen sturen een motor aan. Het aantal RPM zit vast in het Python script vb. 5000. We willen een aanpassing sturen naar 5000 toestellen.

Hoe kunnen we dit gaan doen?

### Oplossing: IoT Hub Device Twin

- Digitale "tweeling"van device in de cloud
- Ideaal om **configuratie** naar een toestel te sturen
- Configuratie zal opgeslagen en gesynchroniseerd worden – In de cloud (IoT Hub)
  - Op het toestel (developer moet dit doen in Python)

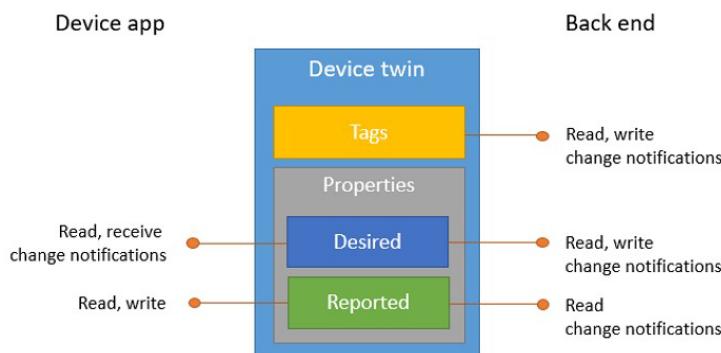
### Voorbeelden

- We wensen enkel temperaturen boven een bepaalde waarde te loggen
- Instellen van parameter tijdstip van doorsturen

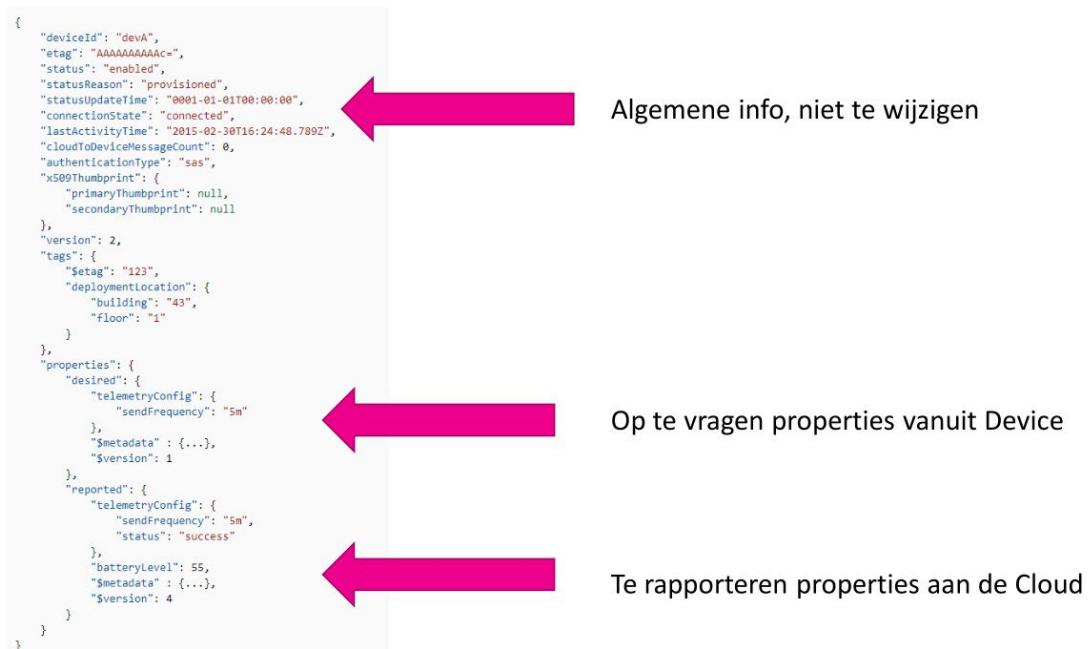
Dit is een belangrijk verschil met MQTT, daar hebben we deze infrastructuur NIET. We moeten dit zelf maken door berichten te sturen naar toestel en omgekeerd

### 8.3.1 Reporting properties

- Device kan rapporteren aan cloud
- Status van batterij
- Status van toestel
- Laatste keer upload van toestel naar cloud
- ...



Figuur 84

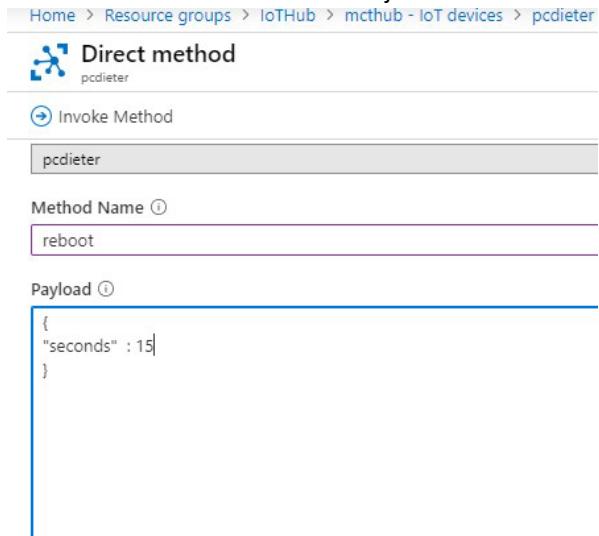


Figuur 85: Properties opvragen en rapporteren in JSON

### 8.3.2 Direct methods

- Methodes op toestel activeren en dit vanop afstand
- We gaan de effectieve Python methode vanop afstand starten

- Vb: Reboot toestel
- Via portal kunnen we de methode activeren
- Via C# kunnen we de methode activeren
- Parameters meeturen ook mogelijk
- Opslaan van device info in JSON document
- Query's mogelijk: vb: SELECT alle devices WHERE batterijstatus > 10%



Figuur 86: In Azure Portal de methode activeren met een payload

## 8.4 IoT Hub Trigger voor Azure Functions

In Visual Studio:

- Opgeven connectiestring uit local settings
- Path ⇒ default laten staan
- Data = array van bytes ⇒ omzetten naar string (JSON)

## 8.5 Ondersteuning

### 8.5.1 SDK's

- Java
- Python
- NodeJS
- Microsoft .NET
- C

### 8.5.2 Devices

- Raspberry Pi
- Adruino
- Intel Edison
- MXCHIP
- Sparkfun electronics
- Adafruit
- Particle

## 8.6 IoT Edge

= service **bovenop** IoT Hub

### 8.6.1 IoT hub issues:

- Alles moet naar de cloud voor verwerking, niks lokaal, soms veel datatrafiek
- We zijn afhankelijk van het Internet
- Het is moeilijk om scripts op een device te updaten
- Script vraagt soms speciale libraries op devices, soms probleem om te installeren
- Versie beheer is lasting, welke device heeft welke software staan?
- Soms wil je bepaalde dat NIET naar Cloud sturen (mag bedrijf niet verlaten)

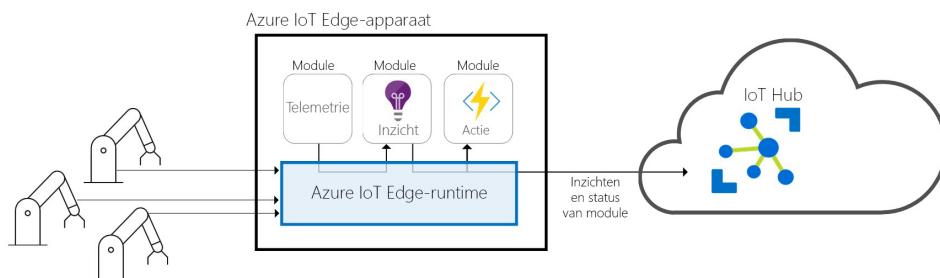
### 8.6.2 Doelstelling

- Cloud workloads lokaal draaien
- Eerste filtering van data op device doen
- Niet alles doorsturen naar Cloud
- Offline scenario voorzien
- Kosten in de cloud doen dalen
- Makkelijk updaten van IoT Edge device

### 8.6.3 Voorbeelden

- Foto analyse op IoT Edge ⇒ we moeten foto niet naar Cloud sturen voor analyse
- Filteren van data op de Edge Device
- Lokale data opslag

### 8.6.4 Opbouw

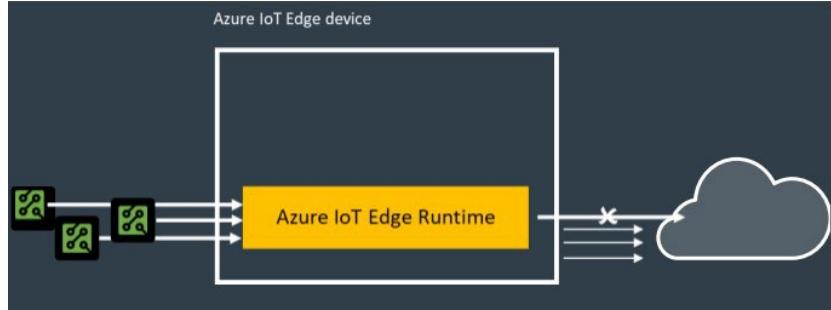


Figuur 87: IoT Edge draait op Docker. Modules zijn Docker Containers

### 8.6.5 Azure IoT Edge Runtime

- Draait op bv RPi of industriële PC
- Installatie van workloads & updates gebeuren op device
- Beveiligen van Edge communicatie
- Verantwoordelijk voor het uitvoeren van de modules
- Status health rapporteren aan de Cloud voor remote monitoring
- Zal zorgen voor communicatie met Leaf devices (low power toestellen zonder Internet)

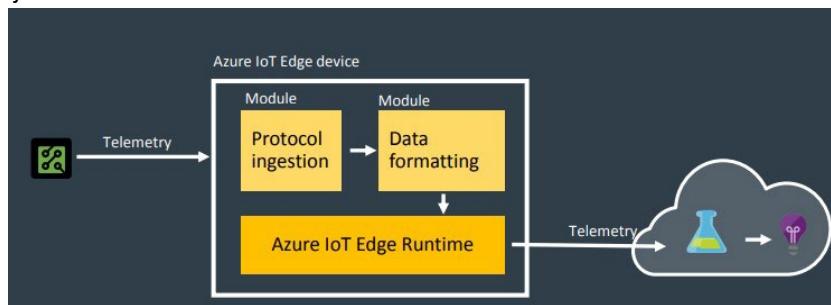
- Zorgt voor communicatie tussen IoT Edge en modules op de device
- Zorgt voor communicatie tussen IoT Edge en Cloud



Figuur 87: Voorstelling Edge device

### 8.6.6 Modules

- Functionaliteit toevoegen op de Edge
- Iedere module zal een actie uitvoeren
- We koppelen modules aan elkaar als een soort data processing pipeline
- We kunnen modules schrijven in C#, Python, . . .
- Modules zijn Docker containers



Figuur 88: Modules die worden toegevoegd aan het Edge device

#### Voorbeelden:

1. Module die data filtert voor deze naar de cloud te sturen
2. Module die data omzet van XML naar JSON voor deze naar de cloud te sturen

## 8.7 IoT in the Cloud vs IoT on the Edge

### 8.7.1 IoT in the Cloud

- Remote monitoring and control
- Merging remote data from across multiple IoT devices
- Near infinite compute and storage to train machine learning and other advanced AI tools

### 8.7.2 IoT on the Edge

- Low latency tight control loops require near real-time response
- Public internet inherently unpredictable

- Privacy of data and protection of IP

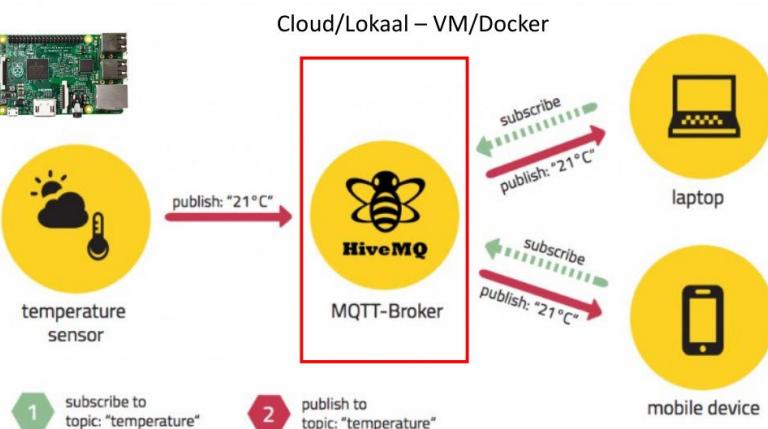
## 8.8 Samenvatting

- Wat en waarom IoT hub?
- Waar zijn digitale twins bij IoT Hub en wat zijn de voordelen?
- Wat zijn direct methods en wanneer gebruiken we dit?
- Wat is IoT Edge?
- Wanneer gebruiken we IoT Edge en wanneer IoT Hub?

## 9 MQTT

= Message Queueing Telemetry Transport

= een machine-tot-machine (M2M) data transfer protocol die ons toelaat om berichten te sturen van een device naar een ander device.



Figuur 89: MQTT-broker

### 9.1 Broker

- = software die instaat voor de communicatie volgens het MQTT protocol
- De broker staat centraal
- Doet niks anders dan berichten doorsturen en ontvangen

#### 9.1.1 Unmanaged services

- MQTT via eigen server
- Zowel VM als docker mogelijk

#### 9.1.2 Managed services

- Amazon IoT
- Google Cloud IoT
- Azure IoT Hub

## 9.2 Doel

- Data verzamelen op device voor transport **naar** cloud
- Data **vanuit** de cloud naar de device sturen
- Communicatie tussen machines (M2M): vb tussen RPi, tussen Pi en ESP32, . . .

## 9.3 Voordeel

- Lightweight (weinig overhead)
- Bi-directioneel (=two-way)
- Standaard (versie 3.1.1)
- Eenvoudig in gebruik

## 9.4 Eigenschappen

- We spreken van een Publishing/Subscriber (Pub/Sub) protocol
- Centraal staat de broker die de bemiddeling doet tussen de clients
- 1/meerdere clients schrijven zich in (subscribe) voor een onderwerp of '**Topic**'
- 1/meerdere clients kunnen een bericht plaatsen (publish) op een '**Topic**'
- Topic = soort wachtrij
- Pub/Sub weten niks van elkaars bestaan af: geen IP, geen ports, geen locatie, . . .
- Pub/Sub moeten niet op hetzelfde moment actief zijn, kan **disconnected** werken

BROKER	DESCRIPTION
mosquitto	mosquitto is an open source MQTT broker written in C. It fully supports MQTT 3.1 and MQTT 3.1.1 and is very lightweight. Due to its small size, this broker can be used on constrained devices.
HiveMQ	HiveMQ is a scalable, high-performance MQTT broker suitable for mission critical deployments. It fully supports MQTT 3.1 and MQTT 3.1.1 and has features like websockets, clustering, and an open-source plugin system for Java developers.
Apache ActiveMQ	ActiveMQ is an open-source multi-protocol message broker with a core written around JMS. It supports MQTT and maps MQTT semantics over JMS.
RabbitMQ	RabbitMQ is a scalable, open-source message queue implementation, written in Erlang. It is an AMQP message broker but has an MQTT plugin available. Does not support all MQTT features (e.g. QoS 2).
mosca	mosca is an open-source MQTT broker written in Node.js. It can operate as standalone or be embedded into any Node.js application. Does not implement all MQTT features (e.g. QoS 2).
RSMB	RSMB is a message broker by IBM available for personal use. It is written in C and is one of the oldest MQTT broker implementations available.
WebsphereMQ / IBM MQ	Websphere MQ is a commercial message- oriented middleware by IBM. Fully supports MQTT.

Figuur 90: Mogelijke MQTT brokers die je kan gebruiken

## 9.5 sectoren

- Telemetry
- Automotive

- Smart phone
- Energy Monitoring (vb Flukso)
- Chat Applications
- Notification services
- LoRa Gateway (multitech)
- Facebook Messenger
- Olie & Gas

## 9.6 Topics

Broker (server) heeft een **Topic/Subject**

- vb: /howest/b008/sensors/temperatuur
- Zelfde vorm als url

Devices hebben **Subscription** op een of meerdere **Topics**



Figuur 91: Topic levels

### 9.6.1 Wildcards



Figuur 92: Single-level wildcard: +



Figuur 93: Multi-level wildcard: #

## 9.7 Quality of Service (QoS)

= een overeenkomst tussen zender en ontvanger om te garanderen dat een bericht geleverd wordt.

**3 niveau's:**

1. Fire and forget
2. Delivered at least once
3. Delivered exactly once

### 9.7.1 Level 0: fire and forget

The sender tries with best effort to send the message and relies on the reliability of TCP. No retransmission takes place in case of failure.

### 9.7.2 Level 1: Delivered at least once

The receiver will get the message at least once. If the receiver does not acknowledge the message or the acknowledgement gets lost on the way, it will be resent until the sender gets an acknowledgement.

Duplicate messages can occur.

### 9.7.3 Level 2: Delivered exactly once

The protocol makes sure that the message will arrive exactly once at the receiver. This increases communication overhead but is the best option when neither loss nor duplication of messages are acceptable.

## 9.8 Communicatie

### 9.8.1 Payload

= Wat zit in het bericht?

- Vrij te kiezen
- JSON
- XML
- CSV

- ...

### 9.8.2 Eigenschappen

- Geen standaard
- Nadeel is interoperability = samenwerking met andere systemen omdat er geen afspraken zijn rond inhoud
- Goede documentatie nodig

### 9.8.3 Security

Login/wachtwoord

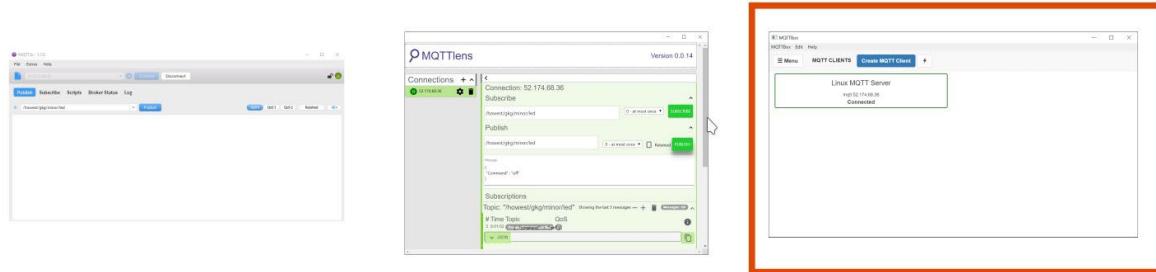
- Eenvoudig maar niet veilig
- Proberen te vermijden

TLS (zie 3MCT)

- Voorkeur, meest veilige
- Beveiligen van communicatiekanaal
- Nadeel is dat je krachtiger IoT device nodig hebt voor encryptie, dus niet overal mogelijk
- <https://paolopatierno.wordpress.com/2015/08/18/gnatmq-and-ssltls-support-make-it-up-and-running>

## 9.9 MQTT Tools

MQTT.FX of MQTT Lens of MQTT Box (Windows Store) installeren om de broker te testen.



Figuur 94: MQTT Tools

## 9.10 MQTT.NET & Python

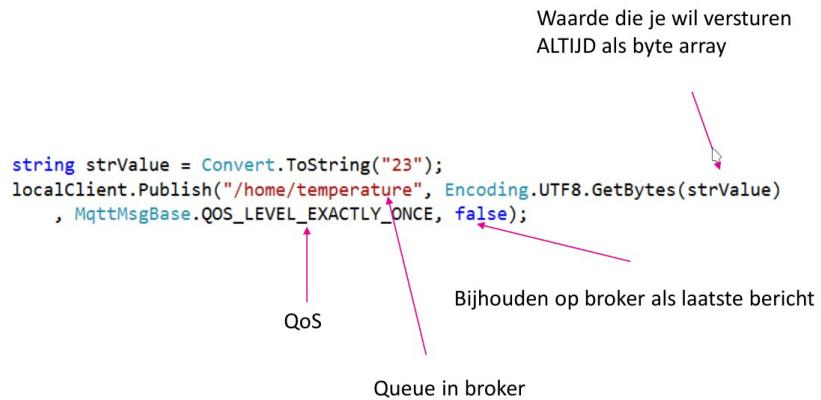
Met MQTT library voor .NET (Nuget package downloaden in Visual Studio)

### 9.10.1 In C#

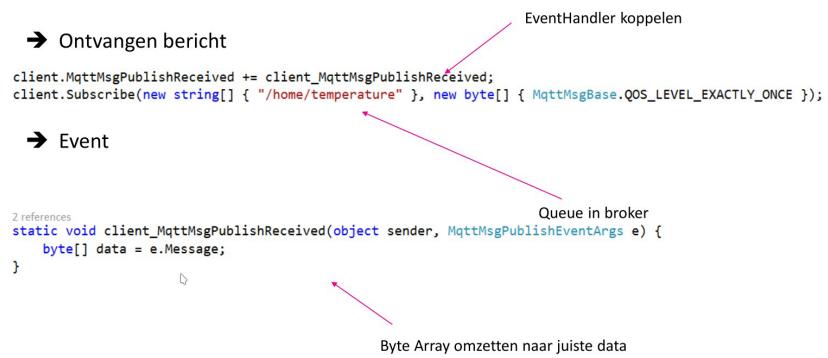
```
Naam van de broker
↑
MqttClient client = new MqttClient("mqtt-howest.northeurope.cloudapp.azure.com");
client.ProtocolVersion = MqttProtocolVersion.Version_3_1;
string clientId = Guid.NewGuid().ToString();
client.Connect(clientId);

Uniek ID voor identificatie bij de broker
↑
```

Figuur 95: Connectie maken



Figuur 96: Versturen bericht



Figuur 97: Ontvangen bericht via een Event

```

client.Publish("temperatures/b116", Encoding.UTF8.GetBytes(tempB116));
client.Publish("temperatures/a202", Encoding.UTF8.GetBytes(a202));

```

Figuur 98: Meerdere topic publishen

```

client.Subscribe(new string[] { "temperatures/#" }, new byte[] { MqttMsgBase.QOS_LEVEL_AT_LEAST_ONCE });

```

Figuur 99: Receive (met multi-level wildcard #)

## 9.10.2 In Python

```

import paho.mqtt.client as mqtt
import json

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe("/dieter")

def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("52.174.68.36", 1883, 60)
client.loop_forever()

```

The diagram highlights parts of a Python MQTT client script. It includes two event handlers: `on_connect` and `on_message`. The `on_connect` handler prints the connection result and subscribes to the topic "/dieter". The `on_message` handler prints the topic and payload of each received message. A pink arrow points from the `on_connect` definition to the text "Eventhandler die zal aangroepen worden als we connectie hebben". A pink arrow points from the `on_message` definition to the text "Deze evenhandler zal de berichten opvangen". A pink arrow points from the `client.on_connect = on_connect` line to the text "Nieuw MQTT bericht ontvangen". A pink arrow points from the `client.on_message = on_message` line to the text "Twee event handlers bij connecteren , bij ieder bericht ontvangen". A pink arrow points from the `client.connect` line to the text "Connecteren naar MQTT Broker". A pink arrow points from the `client.loop_forever` line to the text "Loop forever zodat programma niet stop".

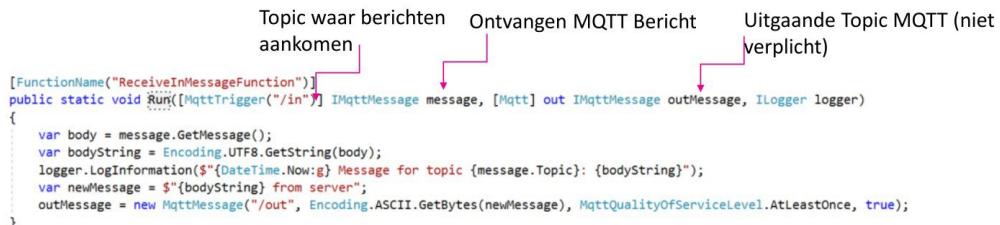
Figuur 100

## 9.11 MQTT & Azure Functions

De MQTT Broker kan Azure Function Triggers activeren met de Nuget Extension:

'CaseOnline.Azure.WebJobs.Extension.Mqtt'

<https://github.com/keesschollaart81/CaseOnline.Azure.WebJobs.Extensions.Mqtt>

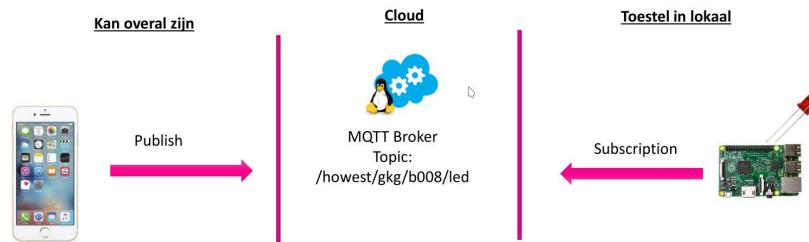


Figuur 101: MQTT Azure Function

### 9.11.1 Opstelling

#### 9.11.1.1 Directe communicatie (one-way)

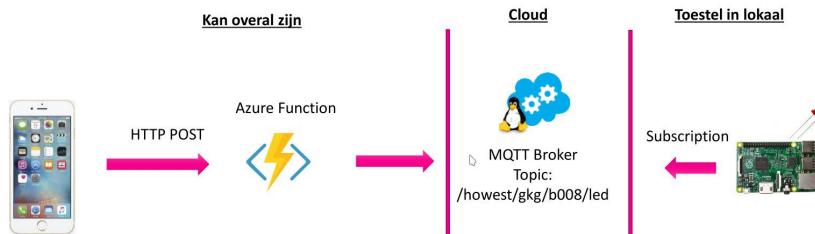
- Vanuit App direct communiceren met MQTT broker via internet
- Voor de meeste platformen (iOS, Android, Windows) is dit mogelijk
- Niet altijd beste oplossing (poorten niet altijd open)



Figuur 102

#### 9.11.1.2 Bericht via Azure Function naar MQTT (one-way)

- Vanuit App HTTP POST naar Azure Functions
- In de Azure function maken we bericht aan om te versturen naar MQTT Broker Topic



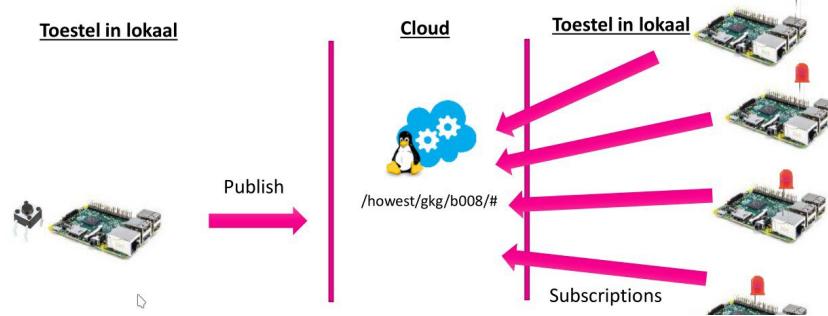
Figuur 102

#### 9.11.1.3 Communicatie tussen devices via de cloud

Als we op een drukknop aangesloten aan de RPI drukken:

- Publish bericht in /howest/gkg/b008/#

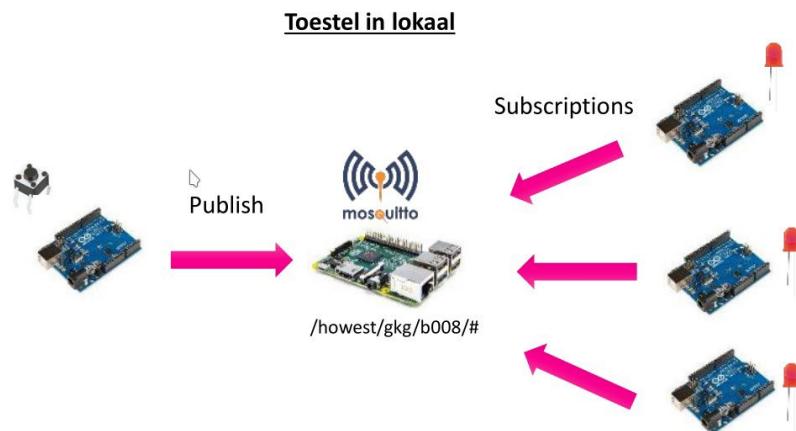
- Broker draait in Cloud
- Iedereen met subscription op /howest/gkg/b008/# zal dit ontvangen



Figuur 104

#### 9.11.1.4 Communicatie tussen devices lokaal (zonder cloud)

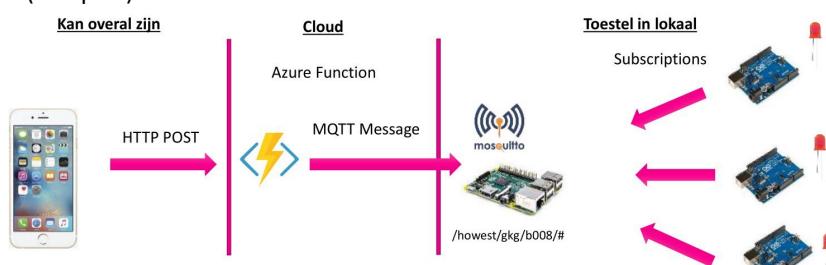
- We kunnen de MQTT broker installeren op de Raspberry Pi
- Devices connecteren met elkaar via de lokale browser op de Pi • Geen link met de cloud nodig



Figuur 105

#### 9.11.1.5 Communicatie tussen devices lokaal via gateway met cloud link

- We kunnen een MQTT Broker installeren op de Raspberry Pi
- Devices communiceren met elkaar via de lokale broker op de Pi
- Via een mobile app sturen we bericht naar Azure Function LED AAN
- Function plaatst bericht in Topic /howest/gkg/b008/#
- Vraagt VPN (complex)



Figuur 106

### 9.12 Samenvatting

- Wat is MQTT?
- Wat zijn topics & subscriptions?
- Welke vormen van topics zijn er, wildcards etc
- Wat is QoS en welke zijn er?
- Welke opstellingen zijn er mogelijk?