

IoT Cloud

Tuur Vanhoutte

16 november 2020

Inhoudsopgave

1 Cloud computing	1
1.1 Enkele eigenschappen	1
1.2 On premise (eigen servers)	1
1.3 Hybrid Cloud (On Premise & Cloud)	2
1.4 IaaS: Infrastructure as a Service	2
1.4.1 Voorbeelden	2
1.5 PaaS: Platform as a Service	2
1.5.1 Talen	3
1.5.2 Voorbeelden	3
1.6 SaaS	3
1.6.1 Voorbeelden	3
1.7 Belangrijkste vendors	3
2 Microsoft Azure	4
2.1 Azure Portal	4
2.2 Azure subscription	4
2.3 Resource Group	5
2.4 Azure Virtual Machines	5
2.4.1 Waarom?	6
2.4.2 Maintenance	6
2.5 Azure Web App	6
2.5.1 Free	7
2.5.2 Shared	7
2.5.3 Basic, Premium	7
2.6 Scaling	7
2.6.1 Scale Up	7
2.6.2 Scale Out	8
2.7 Azure SQL	8
2.7.1 SQL Server Cloud Based	9
2.7.2 Eigenschappen	9
2.7.3 Security	9
2.7.4 Azure DTU	9
2.7.5 Throttling	10
2.7.6 Resources	11
2.7.7 Azure Database for MySQL	11
2.8 Azure & Internet of Things	11
2.8.1 Waarom is Azure belangrijk voor IoT?	11
2.9 Azure CLI	12
2.9.1 Azure Portal	12
2.9.2 Oplossing: Azure CLI 2.0	12
2.10 Bash/Powershell	12
2.11 Andere Azure onderdelen	13
2.12 Samenvatting	13
3 Back-end services met Azure functions	13
3.1 The Internet of	13
3.1.1 The Internet of Information	13
3.1.2 The Internet of Services	14
3.1.3 The Internet of Things	14
3.1.4 Volgende stap: The Internet of Value	15

3.2	Herhaling: Hoe werkt HTTP?	16
3.2.1	Wat is HTTP?	17
3.2.2	HTTP Verbs	17
3.2.3	HTTP status code	17
3.2.4	HTTPS	18
3.2.5	HTTP Request	18
3.2.6	HTTP Response	18
3.2.7	HTTP/2	19
3.3	Webservices	19
3.3.1	Open data	19
3.3.2	Cognitive services	20
3.3.3	Internet of Things	20
3.3.4	JSON	20
3.3.5	URL	21
3.3.6	Fouten	21
3.3.7	Samenvatting	21
3.3.8	Hoe maken we webservices?	22
3.4	Azure Functions	22
3.4.1	Waarom?	22
3.5	Azure Functions security	23
3.5.1	Voordelen van key security	23
3.5.2	Nadelen van key security	24
3.6	Andere Azure Functions	24
3.7	What's next	24
3.8	Azure met Raspberry Pi	25
3.8.1	4 scenario's	25
3.8.2	GET	25
3.8.3	POST	25
3.9	Azure via .NET	26
3.10	Samenvatting	26
4	Azure Storage	26
4.1	Azure Storage account	27
4.2	Azure Files	27
4.3	Azure Disk Storage	28
4.4	Azure Blob Storage	29
4.4.1	Pricing	29
4.4.2	Hot Access	29
4.4.3	Cool Access	29
4.4.4	Static website	30
4.5	Azure Storage Queues	30
4.5.1	Voorbeeld	30
4.5.2	Load leveling	31
4.5.3	Load balancing	31
4.5.4	Temporal decoupling	31
4.6	Azure Table Storage	32
4.6.1	Entities (rijen)	33
4.6.2	Table Storage Data Access	33
4.6.3	Queries	33
4.6.4	Kolom types:	33
4.6.5	Waar gebruiken?	34
4.7	Azure Storage Tools	34

4.8	Programmeren van Azure Storage	34
4.8.1	Wanneer?	34
4.9	Enkele scenario's	34
4.9.1	Post binary file naar Azure Functions en opslag in op Azure Blob	34
4.9.2	Scenario 2	36
4.9.3	Scenario 3	37
4.10	Good practices	39
4.11	Configuratiefiles lokaal of in de cloud?	39
4.12	Samenvatting	39
5	MQTT	39
5.1	Broker	40
5.1.1	Unmanaged services	40
5.1.2	Managed services	40
5.2	Doel	40
5.3	Voordeel	40
5.4	Eigenschappen	41
5.5	Sectoren	41
5.6	Topics	42
5.6.1	Wildcards	42
5.7	Quality Of Service (QoS)	42
5.7.1	Level 0: Fire and forget	43
5.7.2	Level 1: Delivered at least once	43
5.7.3	Level 2: Delivered exactly once	43
5.8	Communicatie	43
5.8.1	Payload	43
5.8.2	Eigenschappen	43
5.8.3	Security	43
5.9	MQTT Tools	44
5.10	MQTT.NET & Python	44
5.10.1	In C#	44
5.10.2	In Python	45
5.11	MQTT en Azure Functions	45
5.11.1	Opstellingen	46
5.12	Samenvatting	48
6	IoT Hub	48
6.1	Werking	49
6.1.1	Wat als device niet krachtig genoeg is?	49
6.2	Hoe data verwerken die binnenkomt op IoT Hub?	50
6.3	IoT Hub Device Twin	50
6.3.1	Reporting properties	51
6.3.2	Device methodes	51
6.4	IoT Hub Trigger voor Azure Functions	52
6.5	Ondersteuning	52
6.5.1	SDK's	52
6.5.2	Devices	52
6.6	IoT Edge	53
6.6.1	IoT Hub issues:	53
6.6.2	Doelstelling	53
6.6.3	Voorbeelden	53
6.6.4	Oppbouw	54

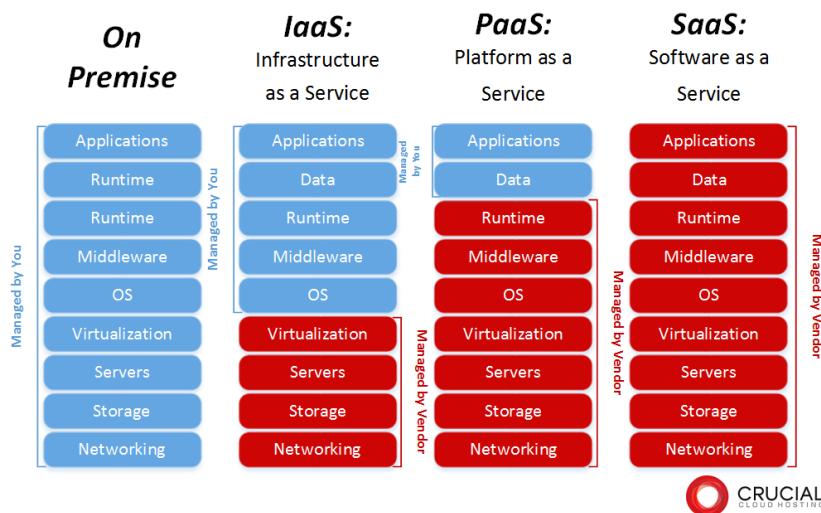
6.6.5	Azure IoT Edge Runtime	54
6.6.6	Modules	54
6.7	IoT in de Cloud vs IoT on the edge	55
6.7.1	IoT in the cloud	55
6.7.2	IoT on the Edge	55
6.8	Samenvatting	55
7	Azure CosmosDB	55
7.1	Relationale databases	55
7.1.1	Waarom ontstaan?	56
7.2	NoSQL databases	56
7.2.1	Waarom?	56
7.2.2	Scaling	57
7.2.3	Availability and always-on	57
7.2.4	Global deployment	58
7.2.5	Gedistribueerde systemen	58
7.3	Soorten NoSQL Databases	58
7.3.1	Key/Value	59
7.3.2	Document databases	59
7.3.3	Column store	60
7.3.4	Graph Database	60
7.4	Voorbeeld SQL	61
7.5	Voorbeeld NoSQL	61
7.6	CosmosDB	62
7.6.1	CosmosDB API	63
7.6.2	CosmosDB API Account	64
7.6.3	Kiezen van een API	64
7.6.4	Container toevoegen	65
7.6.5	Manueel data toevoegen	65
7.6.6	Firewall	66
7.6.7	Aanspreken vanuit Azure Functions met .NET	66
7.6.8	CosmosDB extra's	68
7.7	Extra's	68
7.8	Samenvatting	69
8	Examen	69

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

1.2 On premise (eigen servers)

- IT Afdeling is verantwoordelijk voor ALLES
- Back-up & recovery
- Aankoop hardware
- Scaling...
- Meeste vrijheid om te werken
- Soms investeren in hardware die je maar beperkt aantal dagen nodig hebt
- Vb: webshop en kerstperiode
- Soms verplicht door wetgeving

- Medische data
- Financial 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: medical
- 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

We zullen vooral dit gebruiken in IoT Cloud 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



Figuur 2

2 Microsoft Azure

We kiezen voor Microsoft Azure omwille van:

- Zowel .NET als Open Source (PHP, Java, Node, Python, Flask, Docker...)
- Meeste opties en mogelijkheden
- Zeer lage instapdrempel voor studenten

2.1 Azure Portal

Inloggen via portal.azure.com

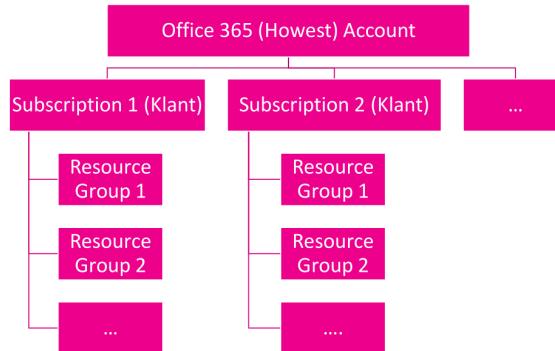
- Beheren van uw Cloud services en applicaties
- Inloggen met Howest Account
- Kies een voldoende sterk passwoord en gebruik 2FA

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 (stabilitet, 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, 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
 - Node
 - ...
- High Availability
 - Replicatie over 3 servers (default)
 - Automatische Back-ups
- Database draait op een server
- Verschillende pricing mogelijkheden

2.7.2 Eigenschappen

- Database draait op een server
- Verschillende pricing mogelijkheden

2.7.3 Security

Azure FireWall: IP adres van je netwerk toelaten

2.7.4 Azure DTU

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

2.7.5 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	AllowAll - No throttling, all queries permitted.	No statements disallowed	All statements allowed
0x01	RejectUpsert - Updates and Inserts will fail.	INSERT, UPDATE, CREATE TABLE INDEX	DELETE, DROP TABLE INDEX, TRUNCATE
0x02	RejectAllWrites - All writes (including deletes) will fail.	INSERT, UPDATE, DELETE, CREATE, DROP	SELECT
0x03	RejectAll - All reads and writes will fail.	All statements disallowed	No statements allowed

Figuur 5: Throttling modes

```

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!

Microsoft Services

Devices	Device Connectivity	Storage	Analytics	Presentation & Action
	Event Hub	SQL Database	Machine Learning	App Service
	IoT Hub	Table/Blob Storage	Stream Analytics	Power BI
	Service Bus	DocumentDB	HDInsight	Notification Hubs
	External Data Sources	3rd party Databases	Data Factory	Mobile Services
			Data Lake	BizTalk Services

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

All services	Search Everything	How likely are you to recommend portal.azure.com to a friend or colleague?										
		0	1	2	3	4	5	6	7	8	9	Extreme
General	GENERAL [1]											
Compute	All resources	★	Report	★	Management groups	★	Subscriptions	★	Resource groups	★	★	
Networking	Cost Management + Billing	★	Reservations	★	Marketplace	★	Help + support	★	Service Health	★		
Storage	Templates	review ★	Tags	★	What's new	★	Quickstart Center	★	Shared dashboards	★		
Web	Free services	★										
Mobile	connect [0]											
Containers	Virtual machines (classic)	★	Virtual machine scale sets	★	Container services (deprecated)	★	Function App	★				
Databases	App Services	★	Container instances	★	Batch accounts	★	Machine learning	★				
Analytics	Cloud services (classic)	★	Kubernetes services	★	Availability sets	★	Disk (classic)	★				
AI + machine learning	Snapshots	★	Images	★	Image definitions	★	Image versions	★	Shared image galleries	★		
Internet of things	OS images (classic)	★	VM images (classic)	★	Classic Virtual Desktops Essentials	★	Civic Virtual Apps Essentials	★	SAP HANA on Azure	review ★		
Mixed reality	CloudSimple Virtual Machines	★	CloudSimple Services	★	CloudSimple Nodes	★						
Integration												
Identity												
Security	NETWORKS [1]											
DevOps	Virtual networks	★	Virtual networks (classic)	★	Load balances	★	Application Gateways	★	Virtual network gateways	★		
Migrate	Local network gateways	★	DNS zones	★	CDN profiles	★	Traffic Manager profiles	★	ExpressRoute circuits	★		
Management + governance	Network Watcher	★	Network security groups	★	Network security groups (classic)	★	Network interfaces	★	Public IP addresses	★		
Intune	Public IP prefixes	★	Reserved IP addresses (classic)	★	Connections	★	On-premises Data Gateways	★	Route tables	★		
Other	Route filters	★	Application security groups	★	DDoS protection plans	★	Firewalls	★	Front Doors	★		
	Service endpoint policies	review ★	Private DNS zones	review ★	Front Door WAF policies	★						
STORAGE [4]												
Storage accounts	Storage accounts (classic)	★	Recovery Services vaults	★	Storage Sync Services	★	Storage Device Managers	★	Data Lake Storage Gen1	★		
Storage explorer	Storage Explorer Data Managers	review ★	ImportExport jobs	★	Data Shares	★	Data Share invitations	★	Data Box Edge / Data Box Gateway	★		
Data Box	Azure NetApp Files	★										
WAF [1]												
App Services	API Management services	★	CDN profiles	★	Search services	★	Notification Hubs	★				
Notification Hub Namespaces	App Service plans	★	App Service Environments	★	API Connections	★	App Service Certificates	★				
App Service Domains	Media services	★	SignalR	★								
MOBILE [2]												
App Services	Notification Hubs	★										
CONTAINERS [7]												
Container services (deprecated)	Container instances	★	Kubernetes services	★	Container registries	★	Batch accounts	★				
Service Fabric clusters	App Services	★										
CONTINUOUS INTEGRATION/DEPLOYMENT [1]												
CI/CD Pipelines	Build pipelines	★	Release management	★	Deployment slots	★	Artifacts	★	Logs	★		

Figuur 8

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?
- 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

- Het draait om webpagina's met content
 - Opzoeken van informatie

- Raadplegen van informatie
- E-commerce
- Social networks
- Technologie
 - HTML
 - CSS
 - JavaScript
 - PHP/ASP/JAVA
- Voorbeelden
 - Google
 - Facebook
 - Amazon
 - Stackoverflow
 - Nieuwssites

3.1.2 The Internet of Services

- Connectiviteit
 - Driving factor mobile applications
 - Cross platform
 - Data op makkelijke manier uitwissen
 - 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
- Device 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, ...)

Internet of Things & Internet of services overlappen

3.1.4 Volgende stap: The Internet of Value

"Value"verhandelen via het internet

- Bitcoin
- Andere cryptocurrencies

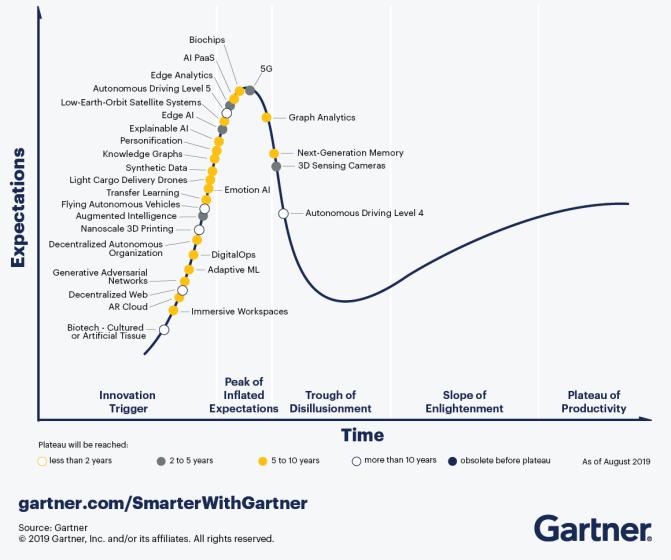
Onderliggende technologie veel belangrijker: Blockchain

- 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
 - Ethereum
 - Smart contracts
- IoT & blockchain zeer veel potentieel
- Bachelor-proef (project 4)

Gartner Hype Cycle for Emerging Technologies, 2019



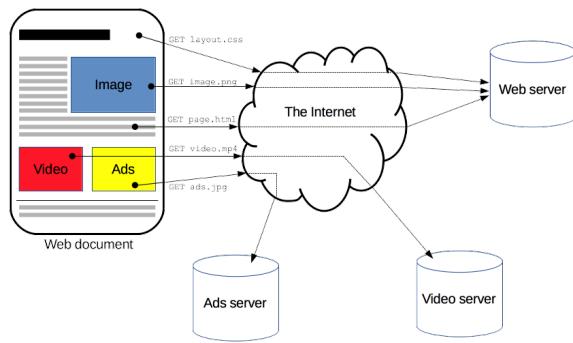
Figuur 9

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 Tekst 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 dus 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 => zelfde effect (UPDATE)
- **DELETE** - Idempotent => meerdere requests => zelfde effect (DELETE)

3.2.3 HTTP status code

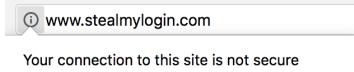
HTTP Response bevat naast data (html of andere) ook een status code:

- 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.howest.be/ HTTP/1.1
Host: www.howest.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: 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/2-compatibele 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 tss 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/>
 - Spraak
 - 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 ongeveer 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
- Daarbinnen 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 key kan terug als value 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 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: operatiernaam 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 terugkeren ?
- Stuur NOOIT maar dan ook NOOIT Exceptions of intern 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 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)
- PUT & DELETE
 - DELETE = DELETE in SQL
 - PUT = UPDATA in database
 - Idempotent methodes ⇒ 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
 - Laravel, 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/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 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://myazuredemos.azurewebsites.net/api/securedfunctionkey?code=8bNn5StOGluAsjqoxgyHcQh05gjXL7F1PlIYn/fcmpWPFk0GgvjmA==>

Figuur 16: De key zit in de URL

3.5.1 Voordelen van key security

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

3.5.2 Nadelen van key security

- Key moet in de applicatie zitten die we verspreiden
 - Vb publieke mobile app's 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 What's 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 terugkeren
 - 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 onderzoeks vragen voor Project 4 in 3MCT

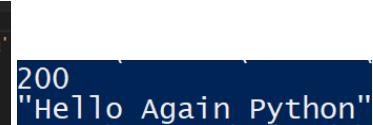
3.8 Azure met Raspberry Pi

3.8.1 4 scenario's

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



200
"Hello Again Python"

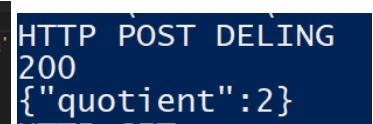
Figuur 17: HTTP GET request en response

Werking:

- Importeren requests library
- url opstellen die we willen aanroepen
- Get() methode uitvoeren met url
- status_code = HTTP Statuscode
- text = inhoud van response

3.8.3 POST

```
import requests
url = 'https://myazrefunctionsdemos.azurewebsites.net/api/HelloWorld/Python'
ret = requests.get(url)
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 load in een dictionary
- Daarna kunnen we de waarden uit de dictionary opvragen zoals normaal

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, ...

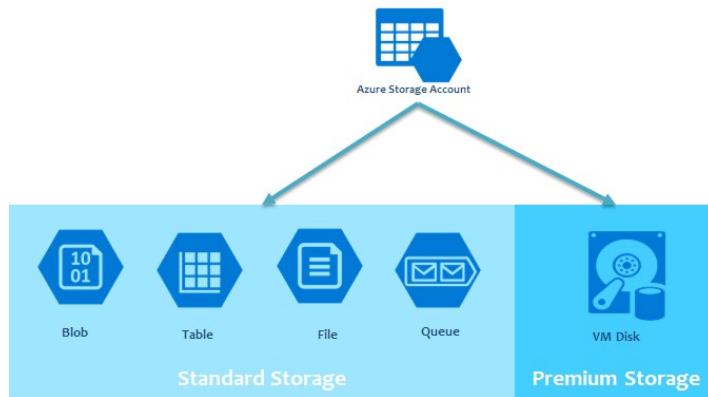
Soorten storage

- File (zien we niet in labo)

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

4.1 Azure Storage account

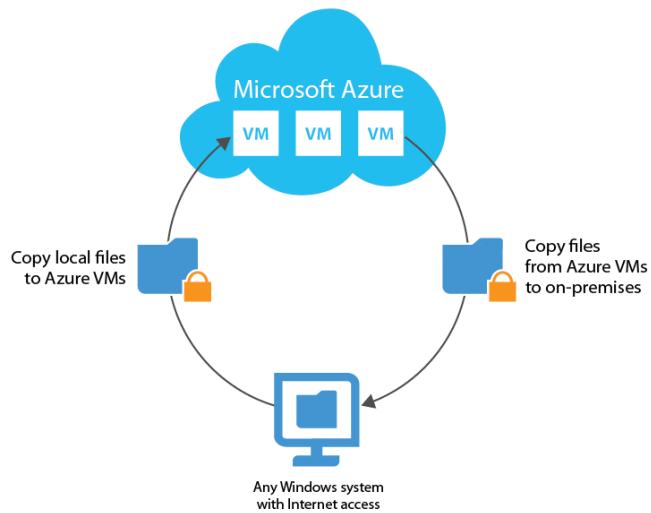
- Alles komt hierin samen
- Dit moeten we aanmaken in Azure
- Daarbinnen maken we dan de soorten storage aan



Figuur 20: Azure Storage account

4.2 Azure Files

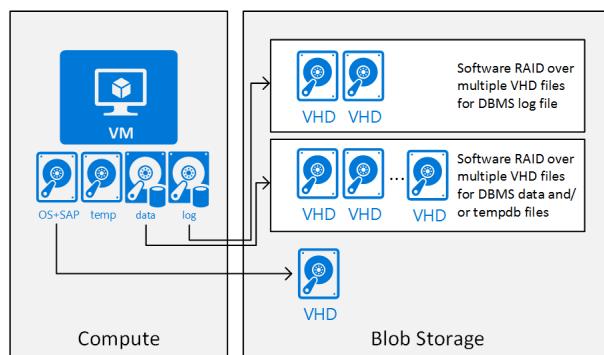
- Hard Disk maar in de Cloud
- Je kan deze mappen naar eigen pc, vb Z:..
- Handig als je veilig files wil kopiëren tussen Cloud server en on premise server
- Zal de gekende protocollen volgen zoals SMB 3.0
- Veel gebruikt in Hybrid Cloud
- Werkt niet op school (Blocked)
- Kan je thuis eens proberen



Figuur 21

4.3 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

4.4 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
- 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 Request

4.4.1 Pricing

- Prijs is per GB/per 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 23: Prijstabel Azure Blob Storage

4.4.2 Hot Access

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

4.4.3 Cool Access

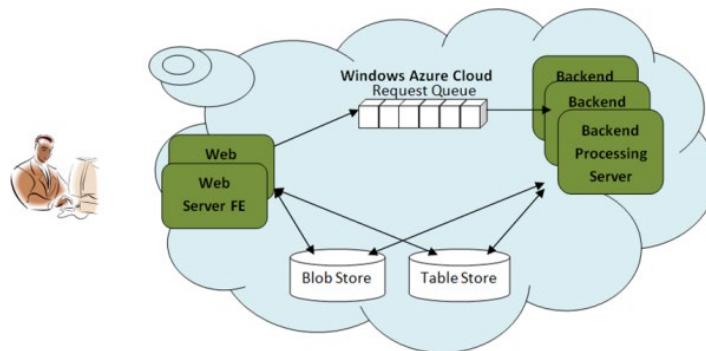
- Backups voor lange termijn
- Data die we niet frequent nodig hebben

4.4.4 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>

4.5 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 24

4.5.1 Voorbeeld

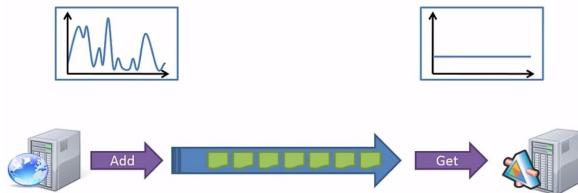
- Sender = webshop
- Ontvanger = stuk software (bv Azure Functions) die order zal controleren en naar een nieuwe wachtrij zal sturen



Figuur 25

4.5.2 Load leveling

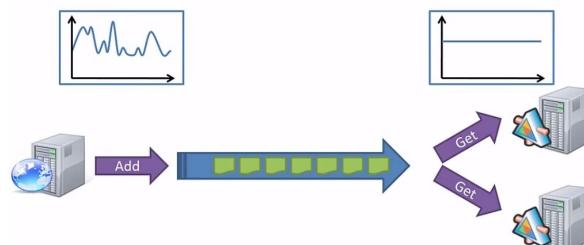
- Constante load op applicatie die verwerkt moet worden
- Werkt zolang we niet meer berichten aanmaken dan wat de applicatie kan verwerken



Figuur 26

4.5.3 Load balancing

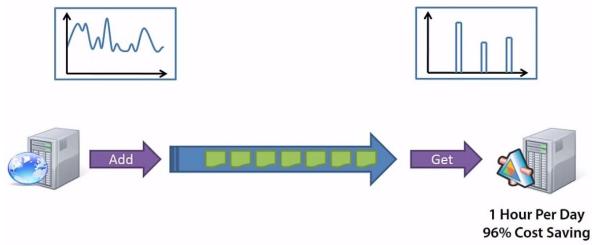
- Teveel berichten per uur
 - Applicatie die verwerkt schalen
 - Nieuwe instantie \Rightarrow verdubbeling verwerking
- Wanneer terug minder berichten per uur: 1 applicatie stoppen
- High Availability \Rightarrow bij crash 1 app, ligt systeem **niet** plat



Figuur 27

4.5.4 Temporal decoupling

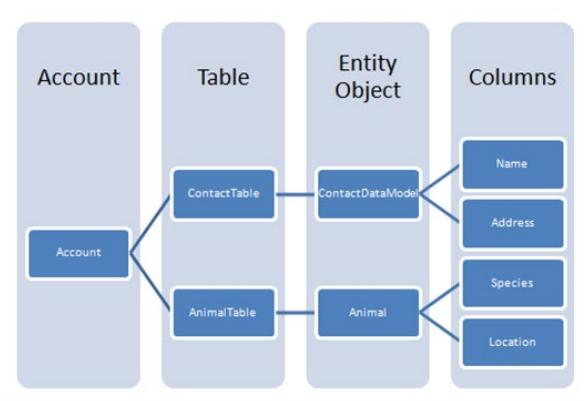
- We slaan alles op in Queue voor later
- Applicatie voor werking start om 23u op
 1. Verwerkt alles
 2. Shutdown app



Figuur 28

4.6 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 29

- Partitions
 - Verdelen van table 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 30

4.6.1 Entities (rijen)

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

4.6.2 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, ...)

4.6.3 Queries

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

4.6.4 Kolom types:

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

- Int64 of long
- String

4.6.5 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

4.7 Azure Storage Tools

Met behulp van de Storage Explorer:

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

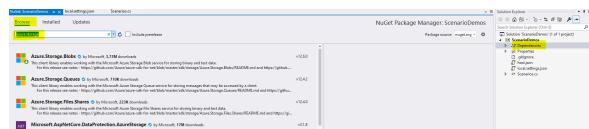
4.8 Programmeren van Azure Storage

4.8.1 Wanneer?

- Opladen van bestanden op 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 achter juiste package, vb: Azure Storage



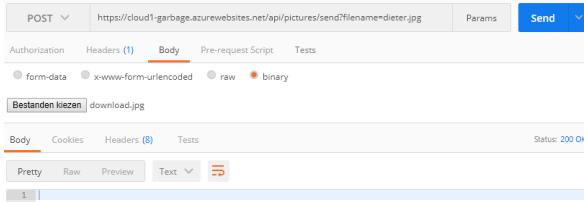
Figuur 31: Azure Storage toevoegen in Visual Studio 2019

4.9 Enkele scenario's

4.9.1 Post binary file naar Azure Functions en opslag in op Azure Blob



Figuur 32: POST binary file naar Azure Functions, opslag in Azure blob



Figuur 33: De Binary data komt in de body van het HTTP Request via Postman

```
[FunctionName("FileUpload")]
[HttpTrigger(AuthorizationLevel.Function, "post", Route = "upload/{fileName}")]
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"));
        CloudBlockBlob blob = storageAccount.CreateCloudBlobClient();
        CloudBlockContainer container = blob.ContainerReference("uploads");
        CloudBlockBlob blockBlob = container.GetBlockBlobReference(fileName);
        await blockBlob.UploadFromStreamAsync(req.Body);
        log.LogInformation("Upload done");
        return new OkObjectResult("OK");
    }
    catch (Exception ex)
    {
        log.LogError(ex.Message);
        return new StatusCodeResult(500);
    }
}
```

Figuur 34: 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 35: Via de parameter in URL geven we de naam van het bestand mee aan service

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

Figuur 36: Connectie maken met storage via connectiestring die we ophalen uit localsettings.json file
(Connectiestring Storage staat op Azure)

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

Figuur 37: Via CloudBlobClient kunnen we praten met de storage

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

Figuur 38: We halen referentie op naar de container waar we de files willen opslaan

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

Figuur 39: We maken 'lege' file aan

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

Figuur 40: We uploaden de stream van de body in de 'lege' file

4.9.2 Scenario 2



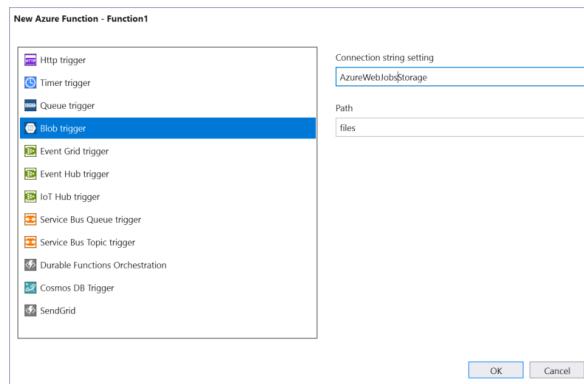
Figuur 41: Send e-mail wanneer een blob verschijnt op Azure Blob

Blob trigger

- Functie uitvoeren als er een Blob file aangemaakt is in een container
- Vb:
 - Uploaden van foto's naar blob voor betere verwerking
 - Uploaden CSV files voor verwerking



Figuur 42: In de settings file plaatsen we bij AzureWebJobStorage de connectionstring naar onze storage account



Figuur 43: 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

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@gmail.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 44: 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

4.9.3 Scenario 3



Figuur 45: 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 schrijven we 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 46: 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 47: Objecten voor Table Storage moeten erven van Table Entity

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 48: Python script die queue opvult

4.10 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 stuurt)
- Beter: HTTP trigger ⇒ File naar Blob ⇒ Blob Trigger ⇒ Bericht op Queue ⇒ Queue Trigger ⇒ Mail sturen
 - 3 triggers
 - Schaalbaar
 - Hogere performantie

4.11 Configuratiefiles lokaal of in de cloud?

The screenshot shows two side-by-side views of Azure Functions configuration.

Left (Local): A JSON configuration file snippet:

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

Right (Cloud): The Azure portal's Application settings page for a function app.

Name	Value
APPINSIGHTS_INSTRUMENTATIONKEY	<Hidden value>
APPLICATIONINSIGHTS_CONNECTION_STRING	<Hidden value>
AzureWebJobsStorage	<Hidden value>

Figuur 49: Lokaal (links) VS in de cloud (rechts)

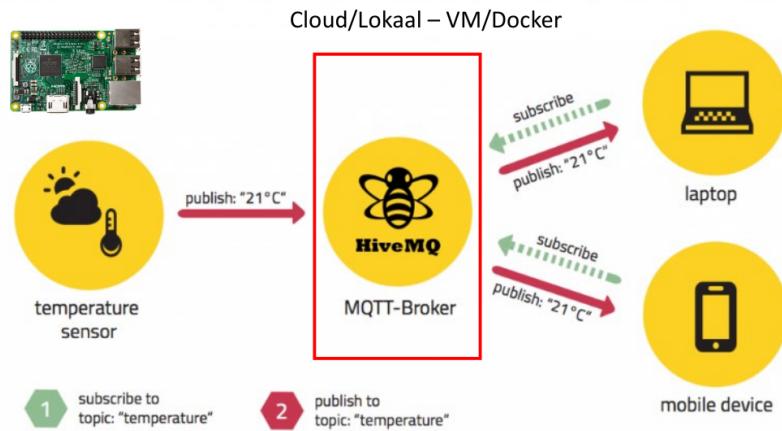
4.12 Samenvatting

- 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 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 50: MQTT-broker

5.1 Broker

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

5.1.1 Unmanaged services

- MQTT via eigen server
- Zowel VM als docker mogelijk

5.1.2 Managed services

- Amazon IoT
- Google Cloud IoT
- Azure IoT Hub

5.2 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, ...

5.3 Voordeel

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

- Eenvoudig in gebruik

5.4 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 niets van elkaar 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 51: Mogelijke MQTT brokers die je kan gebruiken

5.5 Sectoren

- Telemetry
- Automotive
- Smart phone
- Energy Monitoring (vb Flukso)
- Chat Applications
- Notification services
- LoRa Gateway (multitech)
- Facebook Messenger

- Olie & Gas

5.6 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 52: Topic levels

5.6.1 Wildcards

- ✓ myhome / groundfloor / livingroom / temperature
- ✓ myhome / groundfloor / kitchen / temperature
- ✗ myhome / groundfloor / kitchen / brightness
- ✗ myhome / firstfloor / kitchen / temperature
- ✗ myhome / groundfloor / kitchen / fridge / temperature

Figuur 53: Single-level wildcard: +

- ✓ myhome / groundfloor / livingroom / temperature
- ✓ myhome / groundfloor / kitchen / temperature
- ✓ myhome / groundfloor / kitchen / brightness
- ✗ myhome / firstfloor / kitchen / temperature

Figuur 54: Multi-level wildcard: #

5.7 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

5.7.1 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.

5.7.2 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.

5.7.3 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.

5.8 Communicatie

5.8.1 Payload

= Wat zit in het bericht?

- Vrij te kiezen
- JSON
- XML
- CSV
- ...

5.8.2 Eigenschappen

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

5.8.3 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/>

5.9 MQTT Tools

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



Figuur 55: MQTT Tools

5.10 MQTT.NET & Python

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

5.10.1 In C#

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

Naam van de broker
Uniek ID voor identificatie bij de broker

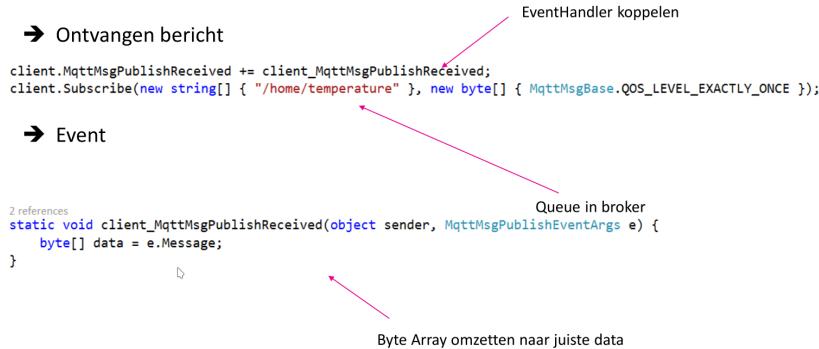
Figuur 56: Connectie maken

```
string strValue = Convert.ToString("23");
localClient.Publish("/home/temperature", Encoding.UTF8.GetBytes(strValue)
, MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, false);
```

Waarde die je wil versturen
ALTIJD als byte array

QoS
Bijhouden op broker als laatste bericht
Queue in broker

Figuur 57: Versturen bericht



Figuur 58: Ontvangen bericht via een Event

```

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

Figuur 59: Meerdere topic publishen

```

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

Figuur 60: Receive (met multi-level wildcard #)

5.10.2 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 Python code for an MQTT client. It includes two event handlers: 'on_connect' and 'on_message'. The 'on_connect' handler prints the connection result and subscribes to the '/dieter' topic. The 'on_message' handler prints the topic and payload of received messages. The client is initialized, connected to a broker at port 1883, and then enters a loop that runs forever.

Figuur 61

5.11 MQTT en 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>

```

Topic waar berichten aankomen   Ontvangen MQTT Bericht   Uitgaande Topic MQTT (niet verplicht)
[FunctionName("ReceiveInMessageFunction")]
public static void Run([MqttTrigger("/in")] IMqttMessage message, [Mqtt] out IMqttMessage outMessage, ILogger logger)
{
    var body = message.GetMessage();
    var bodyString = Encoding.UTF8.GetString(body);
    logger.LogInformation($"[{DateTime.Now}] Message for topic {message.Topic}: {bodyString}");
    var newMessage = $"{bodyString} from server";
    outMessage = new MqttMessage("/out", Encoding.ASCII.GetBytes(newMessage), MqttQualityOfServiceLevel.AtLeastOnce, true);
}

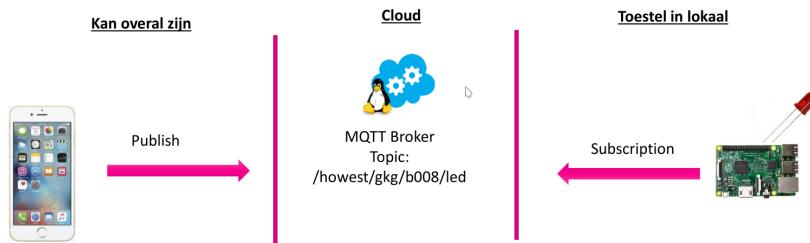
```

Figuur 62: MQTT Azure Function

5.11.1 Opstellingen

Directe communicatie (one-way)

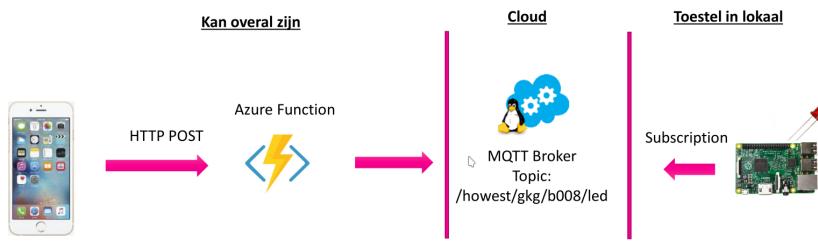
- Vanuit App direct communiceren met MQTT broker via internet
- Voor de meeste platformen (iOS, Android, Windows is dit mogelijk)
- Niet altijd beste oplossing (poorten niet altijd open)



Figuur 63

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



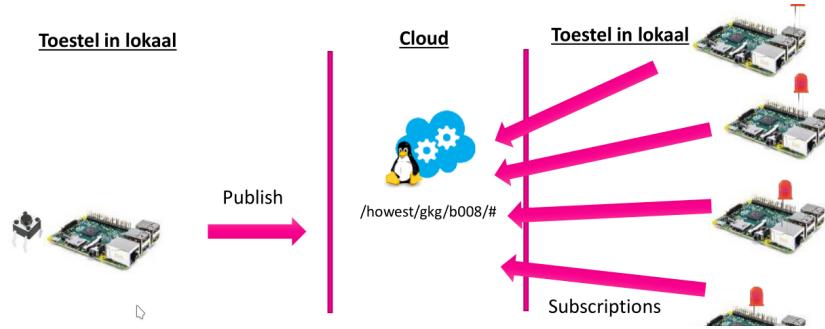
Figuur 64

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

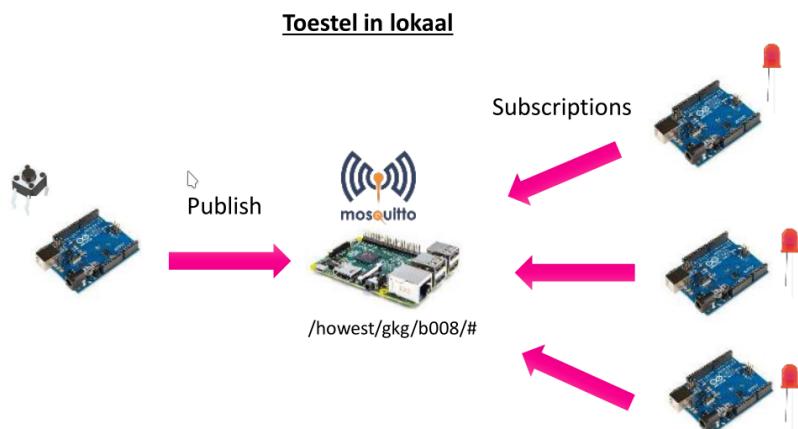
- Iedereen met subscription op /howest/gkg/b008/# zal dit ontvangen



Figuur 65

Communicatie tussen devices lokaal (zonder cloud)

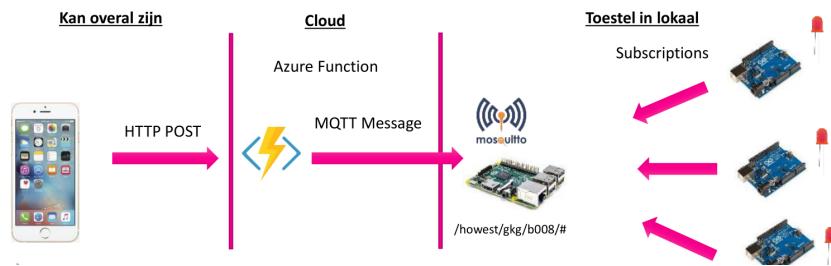
- We kunnen de MQTT broker installeren op de Raspberry Pi
- Devices connecteren met elkaar via de lokale browser op de Pi
- Geen link met de cloud nodig



Figuur 66

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 67

5.12 Samenvatting

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

6 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,Node)
- 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

6.1 Werking

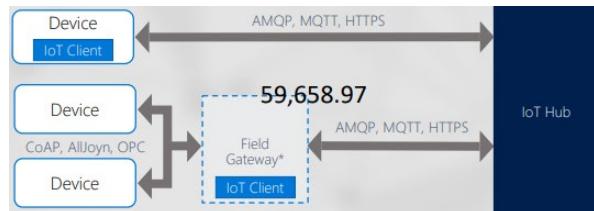
- Device zal rechtstreeks communiceren met IoT Hub
 - Device To Cloud (D2C)
 - Cloud To Device (C2D)
- 3 Protocollen mogelijk:
 - AMQP
 - MQTT (beperkingen t.o.v. eigen MQTT Server)
 - HTTPS
- Meest eenvoudige opstelling
- Hardware moet wel krachtig genoeg zijn

6.1.1 Wat als device niet krachtig genoeg is?

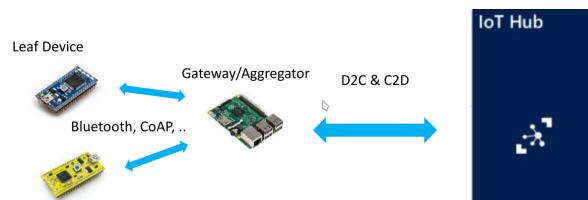
- Bv: Arduino, Mbed Cortex M0,M1, ... (= "Leaf device")
- Goedkope hardware maar niet altijd direct mogelijk om te connecteren op het internet

Oplossing:

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



Figuur 68: Bovenaan: rechtstreeks communiceren met IoT client, onderaan: via Field gateway naar minder krachtige devices



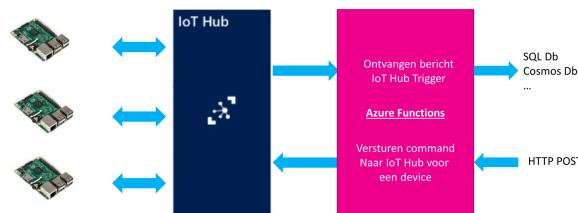
Figuur 69: Opstelling Field gateway

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

6.2 Hoe data verwerken die binnenkomt op IoT Hub?

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



Figuur 70: Schematische voorstelling

6.3 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 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)

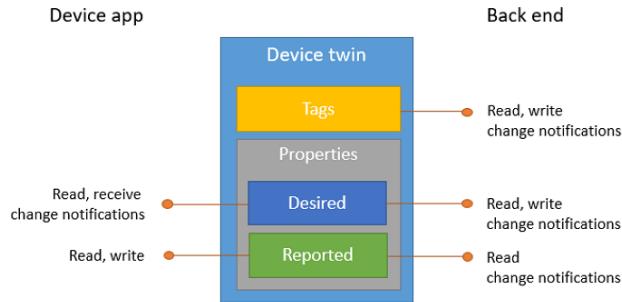
Voorbeelden

- We wensen enkel temperaturen boven een bepaalde waarde te loggen
- Instellen van parameter tijdstip van doorsturen
- Alles wat configurerbaar moet zijn op afstand

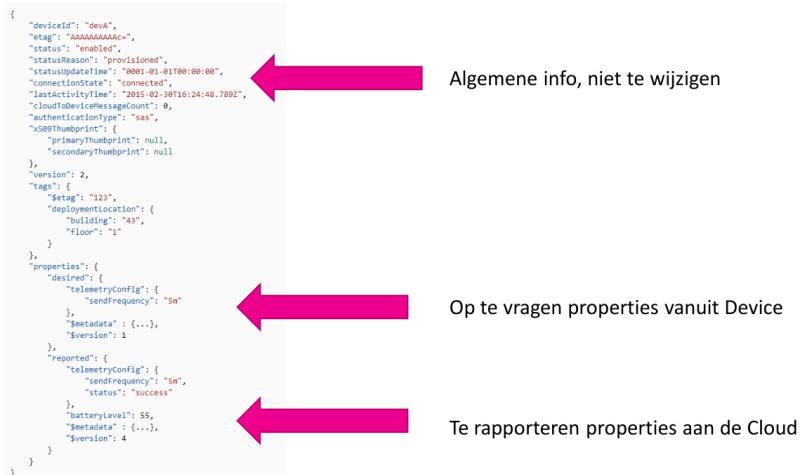
Dit is een belangrijk verschil met MQTT, daar hebben we deze infrastructuur NIET. We moeten dit zelf maken door berichten te sturen naar toestel en omgekeerd

6.3.1 Reporting properties

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



Figuur 71

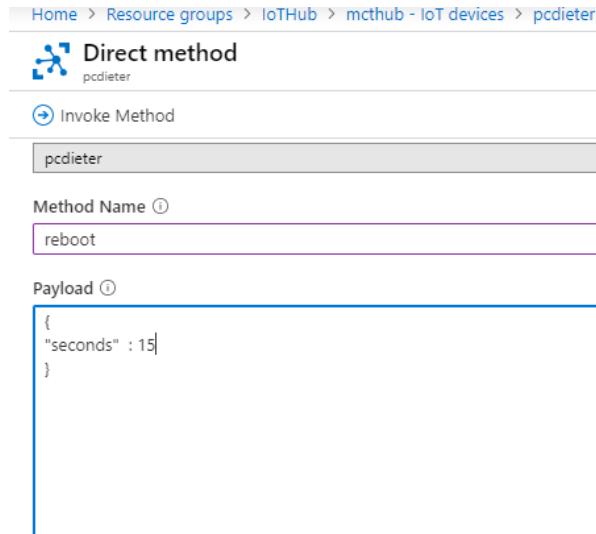


Figuur 72: Properties opvragen en rapporteren in JSON

6.3.2 Device 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 73: In Azure Portal de methode activeren met een payload

6.4 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)

6.5 Ondersteuning

6.5.1 SDK's

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

6.5.2 Devices

- Raspberry Pi
- Arduino
- Intel Edison

- MXCHIP
- Sparkfun electronics
- Adafruit
- Particle

6.6 IoT Edge

= service **bovenop** IoT Hub

6.6.1 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)

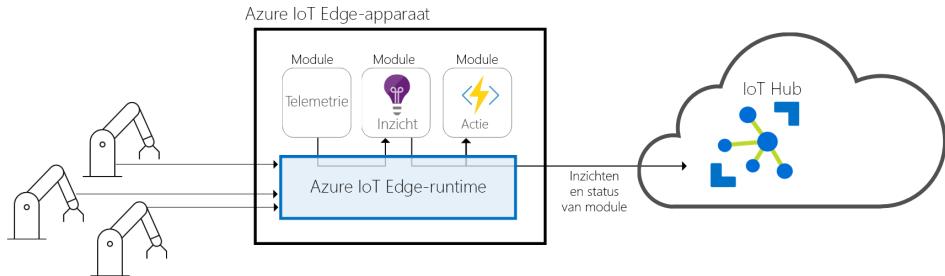
6.6.2 Doelstelling

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

6.6.3 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

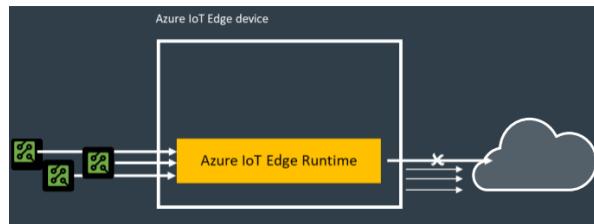
6.6.4 Opbouw



Figuur 74: IoT Edge draait op Docker. Modules zijn Docker Containers

6.6.5 Azure IoT Edge Runtime

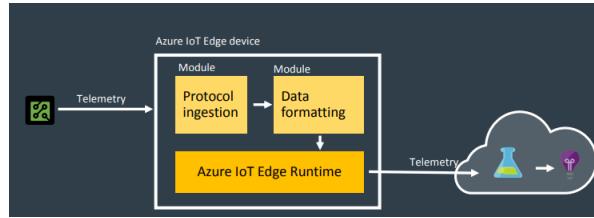
- Draait op bv RPi of industriële PC
- Installatie van workloads & update op device
- Beveiligen van Edge communicatie
- Verantwoordelijk voor het uitvoeren van de modules
- Status health rapporteren aan de Cloud voor remote monitoring
- Zal zorgen voor communicatie met Leaf devices (low power toestellen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



Figuur 75

6.6.6 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 76

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

6.7 IoT in de Cloud vs IoT on the edge

6.7.1 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

6.7.2 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

6.8 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 ?

7 Azure CosmosDB

7.1 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

7.1.1 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

7.2 NoSQL databases

= 'Non-SQL' of 'non-relational SQL'

- Databases die gemodelleerd zijn zonder tabelrelaties te gebruiken

7.2.1 Waarom?

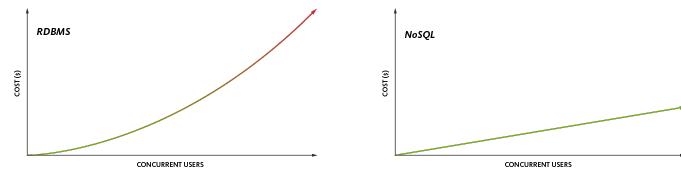
- Hoeveelheid data neemt enorm toe jaar na jaar
 - Social Networks
 - IoT Devices
 - AI (Artificial Intelligence) & Deep Learning systemen
- Concurrency: veel gebruikers op hetzelfde moment: 10000 tot miljoenen
- Globaal bereikbaar en responsive

- 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 eenvoudig meerdere servers toevoegen

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

7.2.2 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

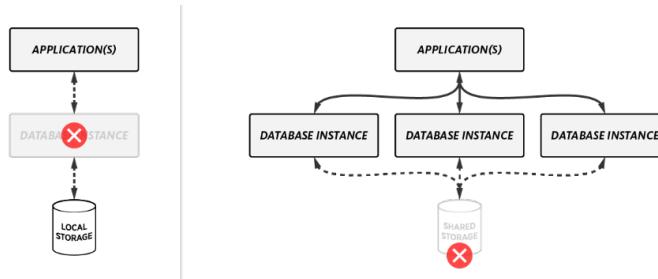


Figuur 77: Relationale DBMS vs NoSQL cost per concurrent user

7.2.3 Availability and always-on

Bij Relationale (RDBMS)

- Gedeelde storage failure
- Volledige applicatie down

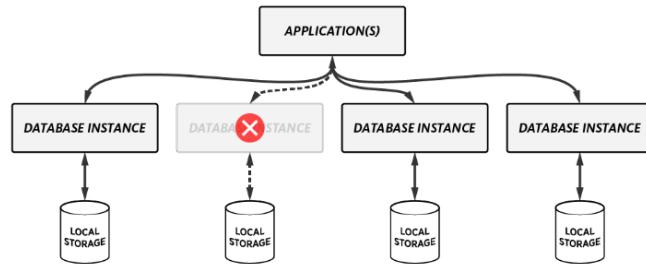


Figuur 78: Beschikbaarheid bij RDBMS: single point of failure

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)

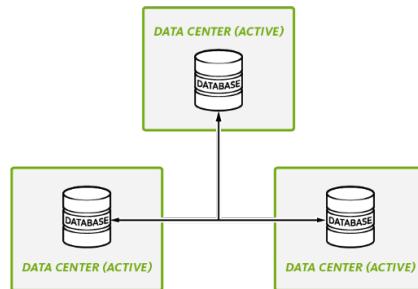
- We spreken hier over **nodes**



Figuur 79: Beschikbaarheid bij NoSQL

7.2.4 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 80

7.2.5 Gedistribueerde systemen

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

7.3 Soorten NoSQL Databases

1. Key/Value (bv Azure table storage)
2. Document database

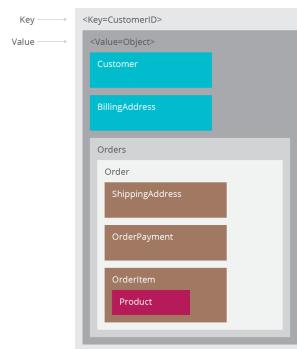
3. Column store (zien we niet)
4. Graph (zien we niet)

7.3.1 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, shoppint basket
- Data waar je minder moet in gaan zoeken via een query
- Data zonder relaties



Figuur 81: Key-value database

7.3.2 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

```

<Key=CustomerID>
{
  "customerId": "fc986e48ca6" +
  "customer": {
    "firstname": "Pramod",
    "lastname": "Sadlage",
    "company": "ThoughtWorks",
    "likes": [ "Biking", "Photography" ]
  }
  "billingaddress":
  {
    "state": "AN",
    "city": "DILLINGHAM",
    "type": "R"
  }
}

```

Figuur 82: Document Database

7.3.3 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	Empld	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;
---	---

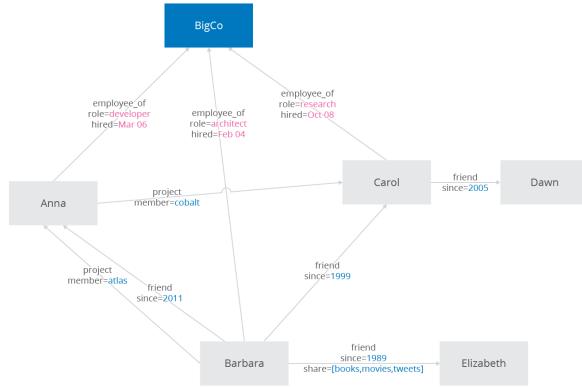
Figuur 83: Column store

7.3.4 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 84: Graph database

7.4 Voorbeeld SQL



Figuur 85: 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 \Rightarrow complexe SELECT en JOIN nodig
- Full Text search is lastig, we moeten alles kolommen checken bij het zoeken

7.5 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"
}
```

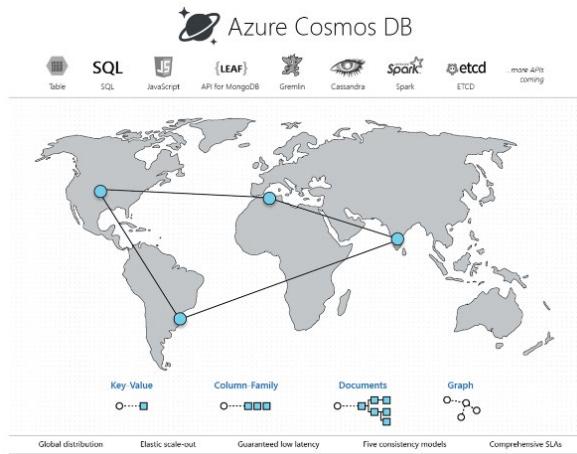
Figuur 86: Voorbeeld



Figuur 87: NoSQL technologieën

7.6 CosmosDB

- = CosmosDB is managed NoSQL database service op Azure
 - Geen onderhoud
 - Geen patching
 - Eenvoudige op te zetten
 - Pay what you use
 - Krachtige API's



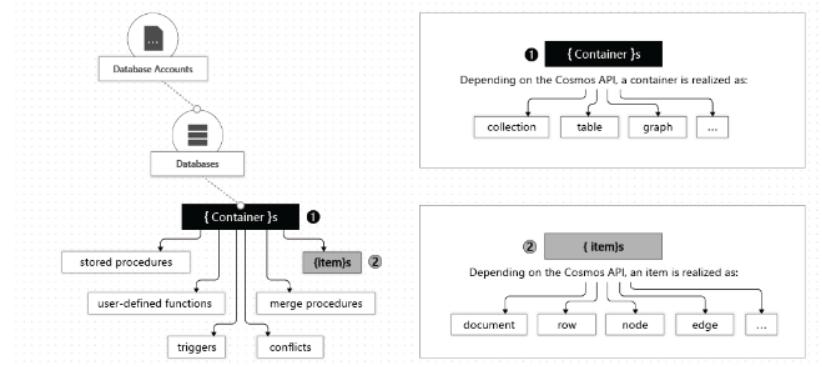
Figuur 88: Azure Cosmos DB

- Globaal gedistribueerde database
 - Op een eenvoudige manier de data zo dicht mogelijk bij de klant brengen
 - Op een 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
- SLA
 - Van 99,99% in een region, meerdere regions = hogere SLA

7.6.1 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 89

7.6.2 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

The screenshot shows the 'Create Azure Cosmos DB Account' wizard with the following steps completed:

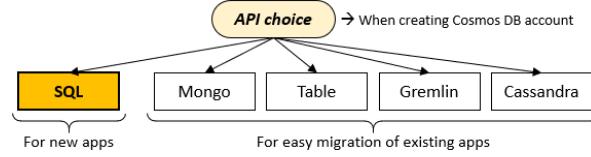
- Step 1: Basics**
 - Subscription: Azure for Students
 - Resource Group: VoorbeeldExamenOefening
- Step 2: Project Details**
 - Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.
- Step 3: Instance Details**
 - Account Name: Enter account name
 - API: Core (SQL)
 - Notebooks (Preview): Off
 - Location: (US) West US
 - Capacity mode: Provisioned throughput (Serverless (preview))
 - Learn more about capacity mode
 - Account Type: Non-Production

Figuur 90

7.6.3 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 91: Kiezen van een API

7.6.4 Container toevoegen

= vergelijkbaar met de klassieke tabel uit RDBMS

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

The screenshot shows the 'Add Container' dialog. It includes fields for 'Database id' (radio buttons for 'Create new' or 'Use existing'), a text input for 'Type a new database id', 'Container id' (text input 'e.g., Container1'), 'Partition key' (text input 'e.g., /address/zipCode'), and a checkbox for 'My partition key is larger than 100 bytes'. Below these are sections for 'Unique keys' and a '+ Add unique key' button.

Figuur 92

7.6.5 Manueel data toevoegen

The screenshot shows the Azure Cosmos DB Data Explorer interface. On the left, the navigation pane includes 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Quick start', 'Notifications', 'Data Explorer', 'Settings', 'Default consistency', and 'Backup & Restore'. The central area is titled 'testdietercosmos | Data Explorer' and shows the 'SQL API' selected. A table view displays data for the 'customers' collection, with columns 'id' and '_city'. The data is as follows:

1	"id": "1", "name": "De Preiter", "city": "Kortrijk", "_rid": "2b2f294a11v14CAAAABAAAAA-", "_self": "https://testdietercosmos.documents.azure.com:443/dbs/2b2f294a11v14CAAAABAAAAA/colls/2b2f294a11v14CAAAABAAAAA/docs/2b2f294a11v14CAAAABAAAAA/", "_attachments": "attachments/", "_ts": 1603440356
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

Figuur 93

7.6.6 Firewall

The screenshot shows the 'Firewall and virtual networks' section of the Azure portal for a Cosmos DB account named 'testdietercosmos'. It displays options for allowing access from 'All networks' or 'Selected networks'. There are tabs for 'Virtual network', 'Virtual Network', and 'Address range'. Under 'Address range', there is a field for 'IP (Single IP or CIDR range)' which is currently empty. Below this, there are sections for 'Exceptions' and 'Allow access from Azure Portal'.

Figuur 94

7.6.7 Aanspreken vanuit Azure Functions met .NET

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

The screenshot shows the 'Connection Keys' section of the Azure portal for a Cosmos DB account. It lists 'Read-write Keys' and 'Read-only Keys'. The 'Read-write Keys' tab is selected, showing fields for 'URI' (https://cloud1.documents.azure.com:443/), 'PRIMARY KEY' (WkRQ9HGxLj1chlwIP46166wloMEIpIpxvgwnmsXCC3ICLd48jAL), 'SECONDARY KEY' (zhDS4GaVvNCFTpJU7gfZGOrEUNTsDqhpVamsPn4POROPGK5), 'PRIMARY CONNECTION STRING' (AccountEndpoint=https://cloud1.documents.azure.com:443/), and 'SECONDARY CONNECTION STRING' (AccountEndpoint=https://cloud1.documents.azure.com:443/). Below these fields, the primary key and endpoint are highlighted in yellow: "CosmosKey": "WkRQ9HGxLj1chlwIP46166wloMEIp" and "CosmosEndPoint": "https://cloud1.documents".

Figuur 95

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 96: 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 97: Datamodel

Stap 3: Deserialisatie

```

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

```

Figuur 98: Deserialisatie

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 99: Wegschrijven van data

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

The screenshot shows the Azure Storage Explorer interface. On the left, there's a tree view of storage accounts and containers. The 'Items' tab is selected in the top navigation bar. A table lists 10 items from a container named 'leesdb'. Each item has a unique ID and contains JSON data representing a TemperatureLog object. The right side of the screen shows a detailed view of one of the items, displaying its properties like deviceid, temperature, and id.

Index	Id	deviceid	temperature	id
1	601313...	"deviceid": "podleter",	17	453a10f8-610f-445c...
2	c75553...	"Temperatuur": 7,		03ed6e9-6141-433...
3	0d0bcf...	"id": "8811990-12a-407c-874a-e594f51824f",		c031a1be-6464-41fe...
4		"x-ms-meta-deviceid": "podleter",		9809b2cd-9404-4000...
5		"x-ms-meta-temperature": 7,		a0332e01-c020-4000...
6	602c0e...	"_etag": "0x80012ed+0000-0000-0000-5d33eb208000\"",		42d65e42-0295-4487...
7		"_attachments": {}		7128639-ef92-4111...
8	060d5e9...	"_attachments": "attachments/",		7894640-d040-4075...
9	9390e2...	"_etag": "0x80012ed+0000-0000-0000-5d33eb208000\"",		f0e0440-ac4d-4e42...
10	56979a...	"_attachments": {}		384c7fb-3f46-46ff...

Figuur 100: 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 101: Opgvragen van data

7.6.8 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

7.7 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 102: Voorbeeld UDF aanmaken

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

Figuur 103: UDF uitvoeren in query

7.8 Samenvatting

- SQL vs NoSQL
- Waarom en wanneer NoSQL databases?
- Welke soorten NoSQL databases zijn er ?
- Weet hoe je kan lezen en schrijven naar CosmosDB ?

8 Examen

- Theorie: 30%
- Labo 70%
- Het examen bevat zeker vragen over ofwel MQTT ofwel IoT