

IoT Cloud

Tuur Vanhoutte

5 oktober 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

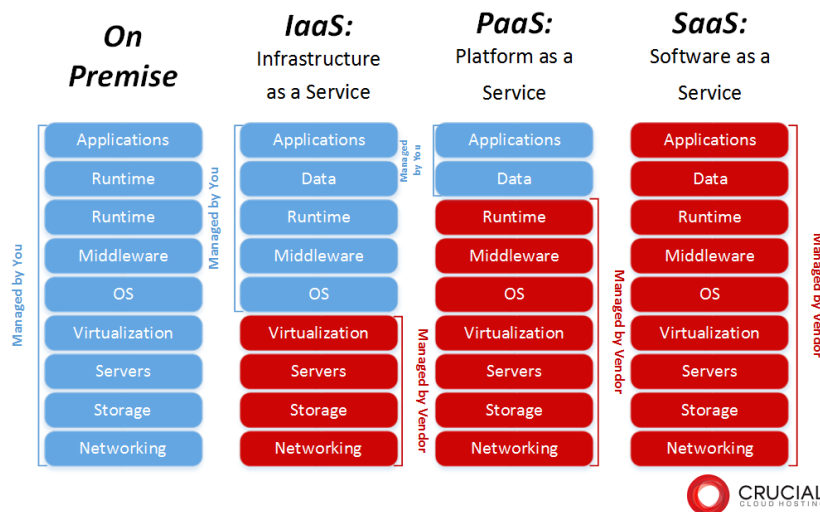
4.8	Programmeren van Azure Storage	34
4.8.1	Wanneer?	34
4.9	Enkele scenario's	34
4.9.1	Post binary file naar Azure Functions en opslag in op Azure Blob	34
4.9.2	Scenario 2	36
4.9.3	Scenario 3	37
4.10	Good practices	39
4.11	Configuratiefiles lokaal of in de cloud?	39
4.12	Samenvatting	39
5	Examen	39

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)
- Ongelimeerde computing power
- Ongelimeerde 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

Figure 1. Magic Quadrant for Cloud Infrastructure and Platform Services



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

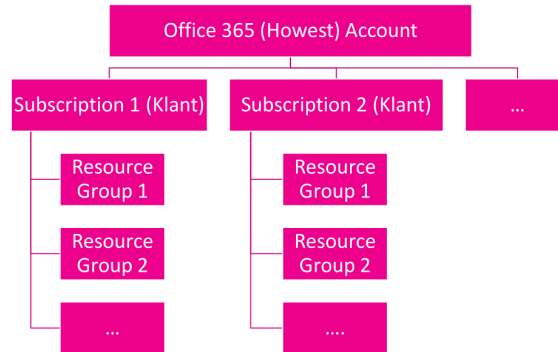
- Beheren van uw Cloud services en applicaties
- Inloggen met Howest Account
- Kies een voldoende sterk wachtwoord en gebruik 2FA

2.2 Azure subscription

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

- Meerdere subscriptions mogelijk

2.3 Resource Group



Figuur 3: Resource groups

- Logische container
- Per project
- Per soort services
 - Storage
 - Webservers
 - Fileservers
 - SQL, Blob, ...
- Je kiest dit zelf
- Makkelijk om te testen, je kan een volledige container verwijderen zonder dat je alles apart moet verwijderen
- Toegangsrechten per container zijn mogelijk
 - via Office 365 account
- Je kies datacenter locatie van je resource group, niet alle resources moeten op dezelfde locatie staan als je resource group.

2.4 Azure Virtual Machines

- Eigen server op Azure omgeving
- Meestal on-premise (server op bedrijf) die we virtueel verplaatsen naar Azure
- Heel wat mogelijkheden
 - File servers
 - Database servers
 - Webservers

- Application servers
- ...
- Toegang tot resources op server die niet mogelijk zijn via een web applicatie
 - Bv. communicatie met oude software pakketten
 - Bv. Genereren van Word/Excel documenten
 - ...

2.4.1 Waarom?

Wanneer je volledige controle wenst over je machine

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

2.4.2 Maintenance

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

Hoe kan ik mijn VM 'up and running' houden?

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

2.5 Azure Web App

- Azure Platform Service
- Laat ons toe om webapps online te plaatsen
- We moeten GEEN eigen server aanmaken
- We moeten GEEN webserver configureren
- Click & go
- Ondersteuning voor:

- ASP.NET (Core)
- PHP
- Python Flask
- Java
- Node.JS
- ...
- Makkelijkste manier om uw applicatie online te krijgen, easy deployment
- Eenvoudige te koppelen aan een GitHub Repository
- Autoscale (not free)
- Monitoring
- Staging mode

2.5.1 Free

= gratis plan voor Azure Web App

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

2.5.2 Shared

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

2.5.3 Basic, Premium

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

2.6 Scaling

2.6.1 Scale Up

- = vertical scaling
- Server krachtiger maken, meer memory en CPU

Pros:

- Minder energie dan scale out
- Eenvoudiger te implementeren
- Minder licenties (n.v.t. op Azure Web App)

Cons:

- Duurder
- Indien we maar 1 machine gebruiken: \Rightarrow 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	3 rd 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?

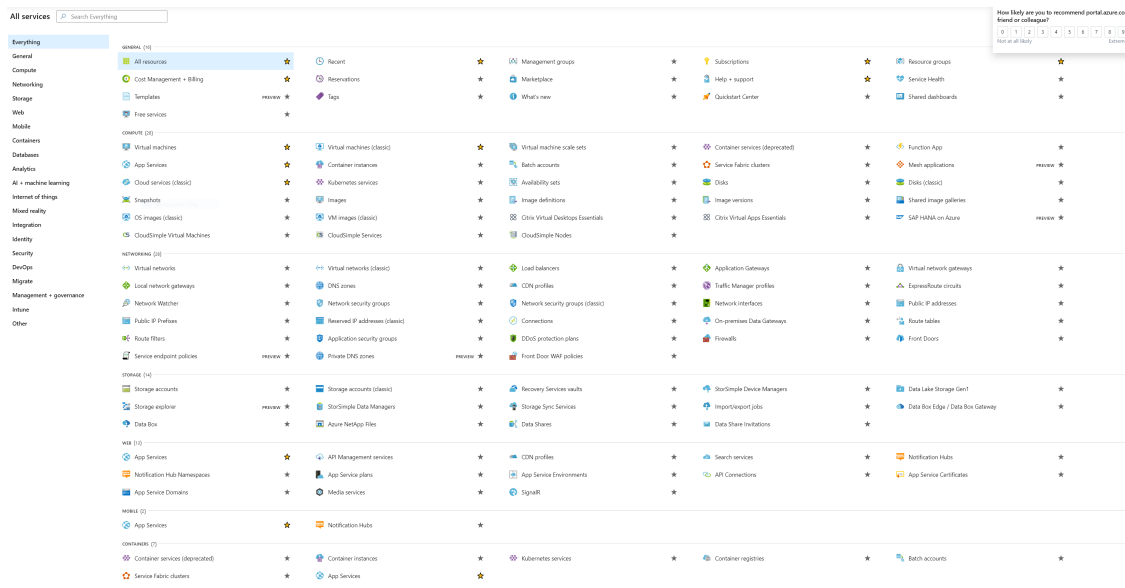
2.9.2 Oplossing: Azure CLI 2.0

- 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



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 uitwisselen
 - Functionaliteit makkelijk aanroepen
 - Centraal staan Cloud platforms
 - REST Based
- Voorbeelden
 - Netflix
 - Azure
 - Amazon Web Services (AWS)
 - IBM Bluemix

3.1.3 The Internet of Things

= Connecting hardware

- 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

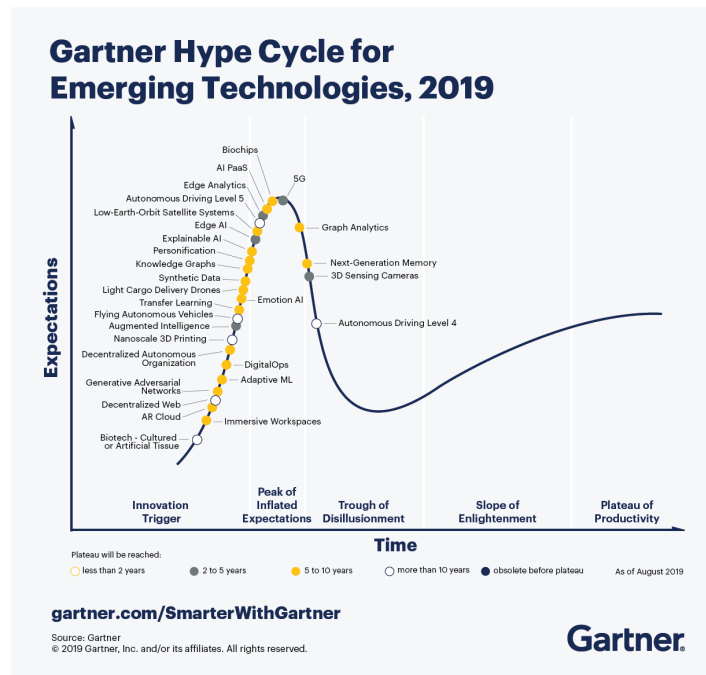
- Bitcoin
- Andere cryptocurrencies

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)



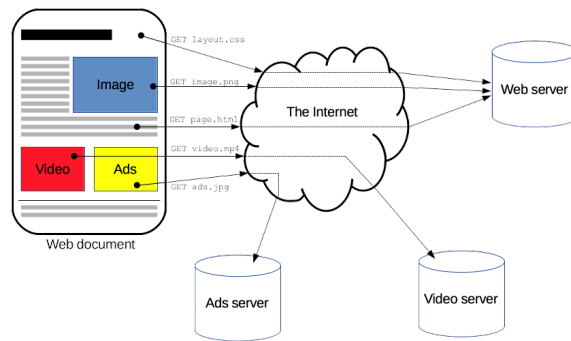
Figuur 9

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

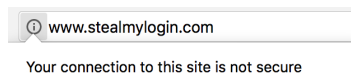
HTTP Response bevat naast data (html of andere) ook een status code:

- 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 terugkrijgt 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 geschieden door een dubbelpunt :
 - Key:value koppels geschieden 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 \Rightarrow 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: operatiennaam 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 \Rightarrow voer je ze 1x uit of 100x, het resultaat blijft hetzelfde
- HTTP POST

- INSERT in SQL

Webservices zijn stateless

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

3.3.8 Hoe maken we webservices?

- Aanwezig op ieder platform
- PHP
 - Laraval, Lumen, Wave
- Python
 - Eve, Flask-Restfull
- .NET
 - ASP.NET Core met C#
 - Azure Functions
 - * Met C#, NodeJS of Python

3.4 Azure Functions

= Serverless functions

3.4.1 Waarom?

- Eenvoudig concept, zeer eenvoudig aan te maken
- Ondersteuning voor verschillende talen
 - Wij gebruiken C# maar Python kan ook: <https://docs.microsoft.com/en-us/azure/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://myazurefunctions demos.azurewebsites.net/api/secureapifunctionkey?code=8bNn5StOGluAsJqoxgyHHcQh05gIXL7F1PIIYn/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 onderzoeksvragen voor Project 4 in 3MCT

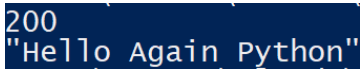
3.8 Azure met Raspberry Pi

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



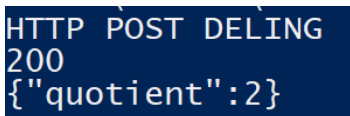
Figuur 17: HTTP GET request en response

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



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, ...

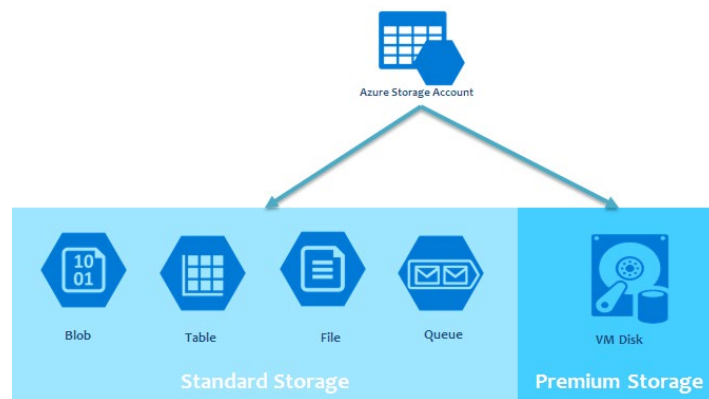
Soorten storage

- File (zien we niet in labo)

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

4.1 Azure Storage account

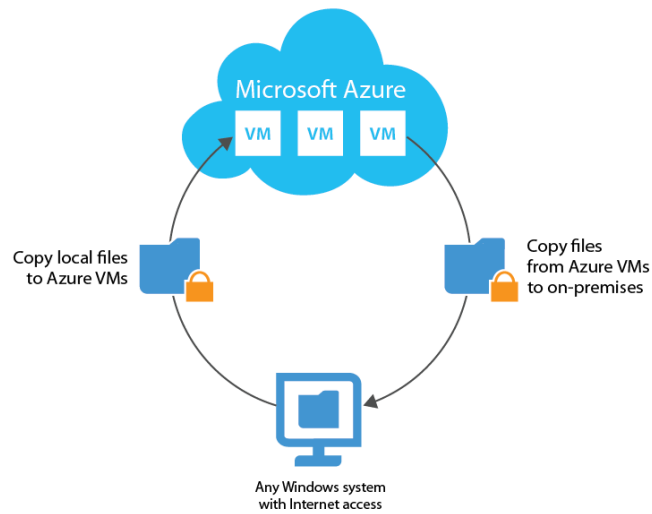
- Alles komt hierin samen
- Dit moeten we aanmaken in Azure
- Daarbinnen maken we dan de soorten storage aan



Figuur 20: Azure Storage account

4.2 Azure Files

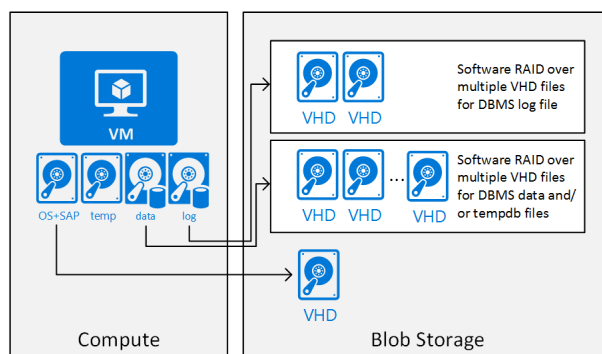
- Hard Disk maar in de Cloud
- Je kan deze mappen naar eigen pc, vb Z:..
- Handig als je veilig files wil kopiëren tussen Cloud server en on premise server
- Zal de gekende protocollen volgen zoals SMB 3.0
- Veel gebruikt in Hybrid Cloud
- Werkt niet op school (Blocked)
- Kan je thuis eens proberen



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

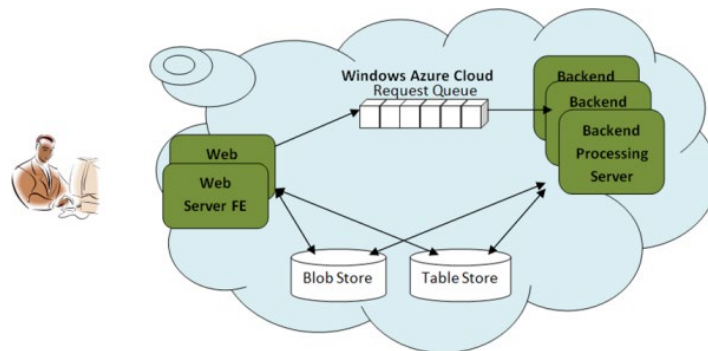
- Backups voor lange termijn
- Data die we niet frequent nodig hebben

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

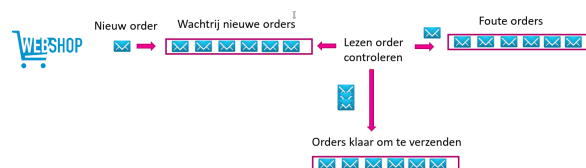
- Een **wachtrij** in de Cloud
- Een applicatie zal berichten op de wachtrij plaatsen (**Sender**)
- Een tweede applicatie kan deze berichten op een zelf gekozen moment ophalen en verwerken (**Receiver**)
- We spreken over een **decoupled** applicatie: zender en ontvanger werken onafhankelijk van elkaar
- 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

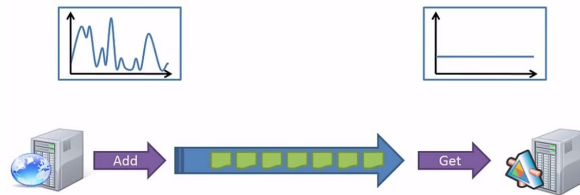
- Sender = webshop
- Ontvanger = stuk software (bv Azure Functions) die order zal controleren en naar een nieuwe wachtrij zal sturen



Figuur 25

4.5.2 Load leveling

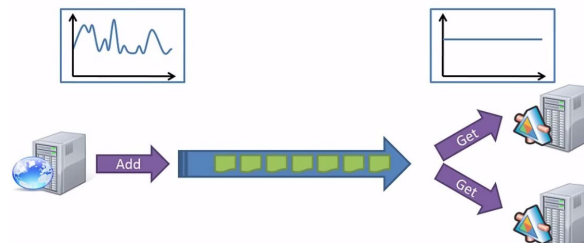
- Constante load op applicatie die verwerkt moet worden
- Werkt zolang we niet meer berichten aanmaken dan wat de applicatie kan verwerken



Figuur 26

4.5.3 Load balancing

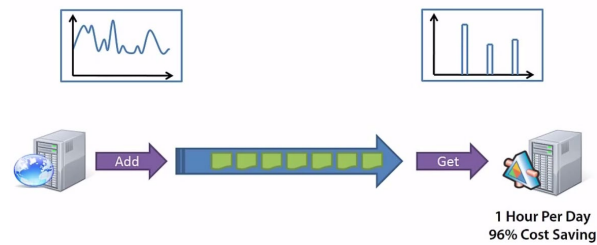
- Teveel berichten per uur
 - Applicatie die verwerkt schalen
 - Nieuwe instantie \Rightarrow verdubbeling verwerking
- Wanneer terug minder berichten per uur: 1 applicatie stoppen
- High Availability \Rightarrow bij crash 1 app, ligt systeem **niet** plat



Figuur 27

4.5.4 Temporal decoupling

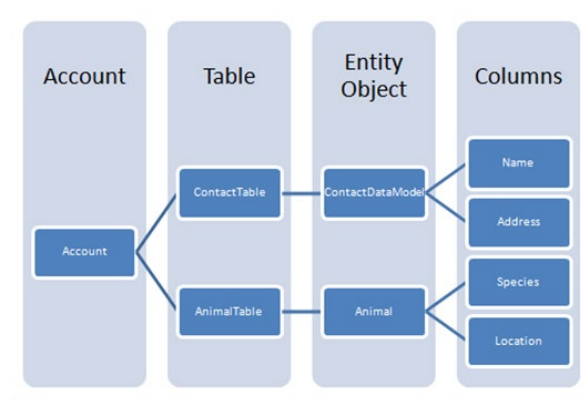
- We slaan alles op in Queue voor later
- Applicatie voor werking start om 23u op
 1. Verwerkt alles
 2. Shutdown app



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
2010	1001	Blogtitle2
2010	1002	Blogtitle3
2009	1003	Blogtitle4
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 \Rightarrow 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 $> 1000 \Rightarrow$ Continuation token
- Query > 30 sec \Rightarrow cancelled
- Geen index mogelijk
- Query op partition key & row key \Rightarrow 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/>

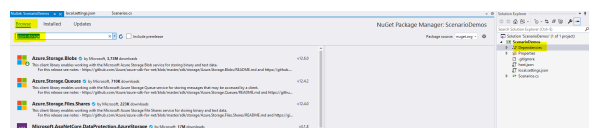
4.8 Programmeren van Azure Storage

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

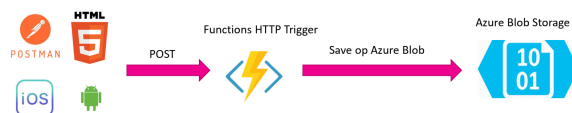
- In .NET Nuget Package toevoegen (zoals pip install bij Python)
- Rechtermuisknop op dependencies ⇒ Manage Nuget packages
- Zoeken achter juiste package, vb: Azure Storage



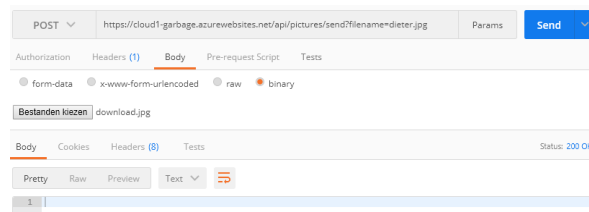
Figuur 31: Azure Storage toevoegen in Visual Studio 2019

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

Figuur 34: Daarna slaan we de file op in de Blob Storage

```
public static async Task<ActionResult> 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



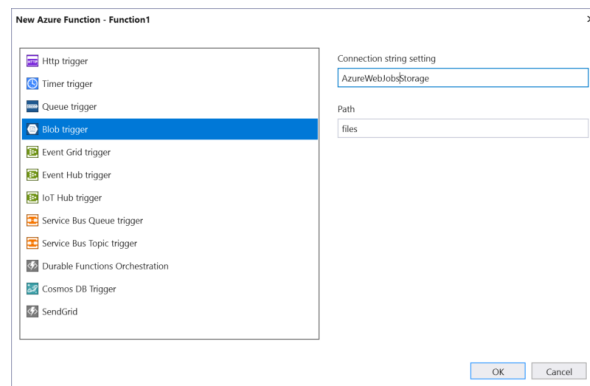
Figuur 41: Send e-mail wanneer een blob verschijnt op Azure Blob

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

```
1  "IsEncrypted": false,  
2  "Values": {  
3    "AzureWebJobsStorage": "DefaultEndpointsProtocol=http;  
4    "FUNCTIONS_WORKER_RUNTIME": "dotnet",  
5    "Storage": "DefaultEndpointsProtocol=https;AccountName=...;  
6  }
```

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")]
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($"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

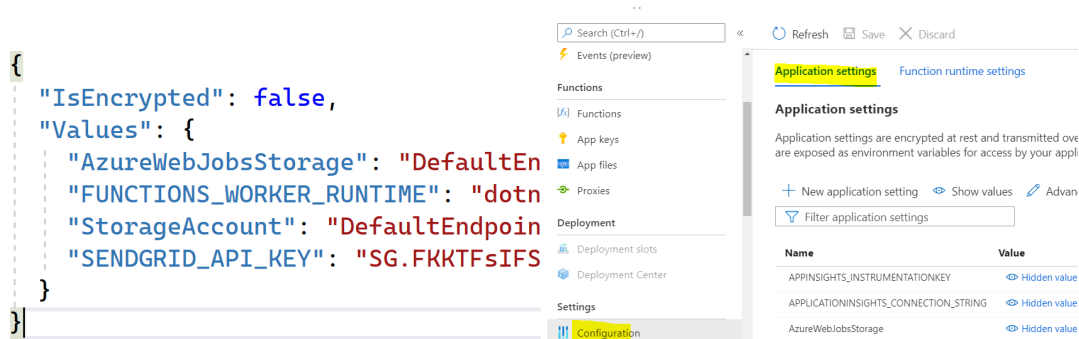
✓ try:
    print("Azure Queue storage v12 - Python quickstart sample")
    queue_client = QueueClient.from_connection_string(connect_str, "sensorlogs")
    dict = {'temperature': 13}
    message = json.dumps(dict)
    message_bytes = message.encode('ascii')
    base64_bytes = base64.b64encode(message_bytes)
    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 verstuurt)
- 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?



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 Examen

- Theorie: 30%
- Labo 70%

- Het examen bevat zeker vragen over ofwel MQTT ofwel IoT