Introductie

Vak = **Projectwerk** -> Gebruikmakende van bestaande backend **ASP.Net Core** een web/mobile **city game** application maken.

**Azure cloud** te gebruiken **-> Continous integration/deployment** met **pipelines**

.Net appliactie => Drie lage structuur, uitgelegd door Sven Mariën.

Gastspreker: **Vragen voor credits**

Projectwerk:

* Aanmaken analyse document
  + Online docs zodat iedereen kan bijdragen
  + Meer op sides…
* Functioneel design (Mockups, user stories)
* Technisch design

Sessie: Gastspreker Microsoft Azure

Mike Martin – cloud solution architect – “technical evangelist” – ex-MVP, Advisor, User group->azug  
Social - @techmike2kx – [mimar@microsoft.com](mailto:mimar@microsoft.com)

Vragen:

Uitleggen niet uitputtend?

#Cloudflare: root cause analysis facebook down

# Azure overlook – a look under the hub

65+ regions van Azure over heel de wereld – altijd in pairs voor availabilty  
-> elk jaar uitbreiding op datacenters voor verbetering en meer data availability, meer failsafe  
SLA garanteed – gericht op 99.999% - meestal gewoon 99.99%

Zo gebouwd voor zo min mogelijk rondlopend personeel – enkel wachters, kuisploeg en rackbeheer.

**Availability zone** – connected with azure backbone om vrij data door te sturen zodat services nooit down hoeven zijn bij het neergaan van 1 datacenter.

**Modular Datacenters** – voor datacenters close to home, te installeren door gewoon electriciteit en data aan te sluiten.

**Project Natick** – datacenter modules voor onder de zee – selfsustaining, only needs connection. Provides heat for fish.

**Energy problem –** trying to become ecological friendly -> zonnepanelen arrays, hydrogen fuel cell generators in datacentra en zelfs carbon capture systemin.

**Azure orbital** – ground station as a service – earth observation and global communication services – sends data to ground stations for computing.

# Advancing Microsoft Azure reliability

Azure active directory and authentication services gaan soms nog wel down. Tried solutions:

* AIOps: no manual feedback anymore and automatic service control
* New Azure brain – Project Narya – predictive and adaptive failure prevention – to solve problems and test edgecases before they ever become a problem.

# Inside Servers

+-10miljoen draaiende servers rond de wereld. Memory optimized servers:

* 2014: Godzilla – insanely powerfull computer servers
* 2019 AMD Gen7
* 2020 Intel Gen7
* Nog zotter 2017 – The Beast – 4x18 core, 4TB, 4x2TB NVMe + 1x980GB SSD + 40gb/s
* 2019 Beast V2 – zotter
* 2020 Mega-Godzilla-Beast – 448 Cores, 24TB RAM

NDv4 – massively scalable AI supercomputer  
GPU support nodig voor data science modellen.

Xbox Cloud Gaming – seamless cloud gaming on for XBOX on mobile or PC devices too

Active liquid submersion of motherboard -> for next step of data:  
**Quantum computing** – quantum computing chipsets, already al zeer klein.

# Inside Azure Resource Manager

Azure portal – CLI and powershell – SDKs – REST API  
Azure bestaat uit een aantal API’s -> ARM – bestaat uit:

* Security: RBAC and Activity logs and telemetry
* ARM configurations – identity management plane: don’t right them yourself anymore! Use a service.
* ARM Resource Providers

Reddog (op markt voor kort nano) = fabrik controller – top ellement of server rack – controlls all provisions for subscriptions.

**Project Bicep** – simple declarative language to provision infrastructure on Azure   
Bicep Language -> bicep build -> ARM templates -> az deployment group create

**Azure Chaos studio –** new service – test environment to test all fallout of failure

**Azure storage portfolio**

* Disk storage
* Blob storage/ object storage
* File storage
* Backup
* Data Transport
* Hybrid storage
* Future technology
* +a lot more of services around data

**New methods of datastorage**: DNA, Holographic, quartz glass – extremely dense and high throughput.

# Technology

Azure devops en github gaan in de toekomst samen service worden. Dus -> files in git en ci/cd pipeline in azure devops maken.

Mentions: Rider tech – zeer positief beschreven.

Serverless zeer interessant maar niet alles kan dat gewoon simpel.

Architecture: Reliabability, resilience, Cost, .. -> iets dat vooral moeilijk te leren valt maar vooral ervaring nodig.

**What to use for cloud services** - App services is your friend, start as simple as possible. Also containers are very useful.

Azure serverless compute

Azure portal – app services (10 gratis per regio) , ook gratis webapps services. -> advanced setting:  
Also subdomain of appservices url -> add .scm. (source control management) before azurewebsites.net -> stuurt je naar **Kudu** = deployment engine aanspreekbaar via api.  
Andere extension: **Monaco** = code als appservice – berijkbaar door /dev/wwwroot/ achter url toe te voegen.  
-> monaco bevat ook git portal – gratis logica is 1 app-service gebrijken als git base en een andere als git-landingpage.  
**Application insight**: performance, uitgebruikers application uses  
**Clarity**: screenshot en videos van de website die uses van gebruikers kan aantonen.  
**Application map:** hoe de logica van de website draait.  
5gb/maand gratis application services.  
**Pipeline deployment gates:** set up condition for pipeline so service doesn’t go down when there are a lot of users.  
Bij pipeline: always split build and release!  
**Deployment slots:**   
**Swap slots:** bij te veel problemen kan je terug swappen  
Interesting for students:  
App services for containers -> not same Kudu experience. App management is in app not in kudu.  
Autoscale: gives opportunity to go with your userbase. With rules based on metrics easy scalabilty is possible. All services using autoscale: App services, API and VM.  
Azure Portal: **FREE SERVICES shown clearly**   
**Message bus:**  tussen front end en middelware/backup van de applicatie plaatsen zodat de operaties tussen delen loshangen van elkaar en gewoon gebruik maken van de service bus.  
Functions – message, servicebus – webapp => free service  
Truely free services use: **Static web apps:** simple service dat gelinkt kan worden aan github.  
->github.dev of op . drukken brengt je naar code in de browser. (met docker dev? Niet zeker over deze actie.   
Andere gratis service: devops starter – met github of azure devops pipeline  
= gratis wizard voor het opzetten van een applicatie die helpt met waar te landen  
=> start een devops project, met setup van container registry etc.  
- Kost tip: infrastructure as code -> elke live sessie weggooien en gewoon herbouwen, enige nadeel is een beetje langere build.  
container-webapp-template.json-> in arm templates bevat parameters. Als je dit extract en kopiëerd naar een container-webapp-template.parameter.json file kan er in elke andere omgeving die je wilt opzetten andere parameters kan meegeven.

CI/CD – gratis security features -> github in samenwerking met azure images: scannen van container images. Of miss toch niet gratis??  
Security posure: lees – Hoe veilig heb ik mijn omgeving gemaakt?  
Security center -> container scan bekijken.  
+ security graph.

Project Info

Sjabloon blueprint ook voor stage gaan we gebruiken voor analyze van de app die we gaan maken.  
Ionic of angular -> altijd ook mobile framework included.

Story – pipeline - blueprint

Lab Notities:

# Lab 1: CI

# Lab 2: docker&continuous deployment

# Lab 3: Architecture

**Api** => configure method = “middleware” pipeline: every request goes through these methods.

Hondehok: architectuur simple – weinig tools – zelfstandig haalbaar  
Huis: erchitectuur complex – veel tools - gemaakt door team met specifieke skills

“Creating the overall blueprint of the system and solution”

Nadenken over:

* Welke technologiën/producten/externe componenten – wat is beschikbaar op de markt.
  + Programeertaal, 3rd party componenten, database
* Security, schaalbaarheid, stabiliteit,…
* Opdeling in fysische en logische blokken en modules
  + 2-tie, 3-tier, … - communicatie tussen tiers en modulles: TCP, JSON, …

Architecture first:

* Architecture = first design desisions: hardest to change, most critical
* [slide info]

Old school => “monolithic” big chunk code – very hard to maintain or change.  
On the other end: Service based architecture – api exists out of all different component separate of each other

Voordelen van layering:

* [slides]

**3 – layer model** => presentation layer (ui,input) – business layer ( validation, transformation) – data layer (db, communication) -> data sources/external services (api,…).

* Voordelen: bij een andere ORM (zoals EntityFramworkCore) moet enkel data layer aangepast worden. Code wordt meer overzichtelijk en werkt zoals verwacht.
* Kan nog voorkomen in het “wild”- wordt vaker al uitgebreid over volgende architecturen

**Clean Code Architecture – Onion architecture – Hexagonal architecture**

Presentation / Infrastructure => Appliaction => Domain – Core (presentation heeft ook referentie naar infrastructure)  
=> Infrastructure is naar buiten toe ook -> application layer heeft geen verwijzing meer naar data => application maakt interfaces aan die de infrastructure toepast.

Core: Business logic – Validation logic – data model – interfaces for the infrastructure  
=> has no dependanties + easily testable

Domain: data model

Infrastructure: gateway to the outside world – implementation ( eg. Repository, context)

**How to adapt to this layer:**

-**Step 0-**Put the right code in the right place

-**Step 1**-Entity framework core nuget package mag enkel nog gebruikt worden door de infrastrucre.

-**Step 2**-Create AF migration => stelselmatig uitbreiden van de database – niet telkens nodig om de database dan te verweideren.  
Met Add-migration [naam] voeg je migration toe. En met Update-database [name migration] voer je de update uit op de databank.

-**Step 3**-Repository patern gebruiken => repository class maken (met GetAll, Get, Add, Update en remove functies) => context verwijzingen in controller verplaatsen door calls naar de repository => En vervolgens nog aan het framework meegeven welke repository dat er moet meegegeven worden in statup -> configureServices -> services.AddTransient<StudentRepository, StudentRepository>();  
AddTransient kan ook AddScoped of AddSingleton. Beste opties zijn Transient(controller per call) en scoped(controller per sessie)<.  
Controllers worden aangemaakt bij elke request door het framework zelf.  
=> Nog te verbeteren met: Interfaces, abstractie

-**Step 4**-Async/Await: Make method return async await – Add “async to the returning method” and add await keyword between return and the code. Then in the controller make the methods async tasks and add await before every call to the repo.

-**step 5**- CQRS & Mediator pattern: **business logic** and **validation** to application core -> eg. **Services**/or/ **CQRS** implementation = Queriable … for **every query** and **command** a **class**. (these classes check the validation and business logic)  
Mediator -> makes sure the right class is used for a querry or command from the controller.  
Afbeelding met tekst

Automatisch gegenereerde beschrijvingEvery query also has a handler inside. A handler that the mediator will give query to.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

In the controller we replace the repository with the mediator

Afbeelding met tekst

Automatisch gegenereerde beschrijving

And we tell the mediator the assembly reference so it can work.

Afbeelding met tekst

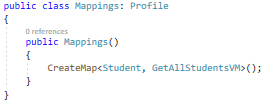
Automatisch gegenereerde beschrijving

-**Step 6**- Viewmodels and mapper – never send complete entities unless asked for – not all properties publicly accessible/relevant – **get all** should give less data than **get by id** -query properties less than command properties.  
-> **Viewmodel / DTO’s** – only contains properties – no logic  
-Add a VM class in queries folder – VM applied in query

The concept

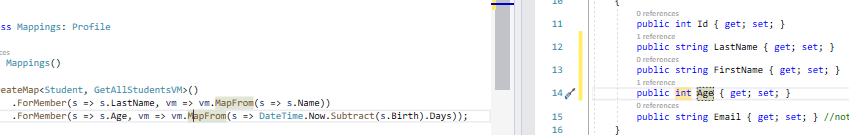
Not final implementation

Veel worden hier mappers voor gebruikt -> automapper package

Add - services.AddAutoMapper(Assembly.GetExecutingAssembly()); - to applicationcoreservices  
Add Mappings folder and class

And in the query map the result.

Mapper can be guided to map different properties on the VM.



-**Step 7**- Status codes – sometimes only 2: Notfound(404) and Ok() – But others could be given back as well: name too long, age too high - exception die niet worden opgevangen geven 500 terug.  
Exception checking doen we meestal in de query sinds deze buiten de normale code flow zitten, deze exception vangen we dan op in de student controller.  
Midelware pipeline helps with checks on the code and handle requests step by step -routing middleware see slides – not caught exception get caught by swagger in our case to view as developers. In deployment just returns a 500. – So let’s add some widdelware to generalize exceptions.

-**Step 8**- Validation – best with model and library – fluentvalidation – validator per command.  
See slides for example.

Validation and mediator – validation happens in mediator - also on slides

**Uitbreidingen:**

-add parameter to student

-asign student to classgroup

-receive email upon deleting a student – add class in infrastructure for communication with outside world.

-as admin see all operation done in system – middleware for logger

-> clean code template – “ASP.net core API – clean code” - codewithmukesh