31/03/2024

BetterBoard Praktikrapport

ANSLAG: 14.338

ANDERS NIELSEN

Indholdsfortegnelse

Introduktion

BetterBoard® ApS er en lille virksomhed, på omkring 13 medarbejdere, med lokation i Kolding. Firmaet ligger i et nichemarked inden for softwareverdenen. Men finder vækst i at levere cloud-baseret softwareprodukter som digital bestyrelsesportal og -datarum.

Bestyrelsesportal By BetterBoard

Datarum By BetterBoard

Virksomhedens hovedprodukt er bestyrelsesportalen. Og indfanger kunder ved at effektivisere bestyrelsesarbejde. Samt lette processen for at overholde gældende lovgivning omkring IT-sikkerhed og GDPR.

En stor faktor for softwareprodukternes salgbarhed, er det faktum at de er hundrede procent cloudbaseret. Således at endebrugerne kan fokusere på funktionalitet, frem for integration eller installation.

BetterBoards primære indtægtsstrøm kommer fra kunder, der tilkøber sig forskellige niveauer af abonnementer eller licenser til bestyrelsesportalen eller datarum.

Teknologi

BetterBoard er hvad man kan klassificere som et Microsoft-hus, hvad software techstack angår. *Azure* er den valgte *cloud provider*, som dermed påvirker udviklingsafdelingens valgte værktøjer og programmeringssprog.

Eksempelvis benytter udviklingsafdelingen sig af *bicep*: Et domæne-specifikt programmeringssprog der bruger syntaks til at udrulle *Azure* ressourcer:

```
param location string = resourceGroup().location

param storageAccountName string =
'toylaunch${uniqueString(resourceGroup().id)}'


resource storageAccount
'Microsoft.Storage/storageAccounts@2021-06-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
  }
}
```

Figur 1: Bicep eksempel[1]

Når infrastrukturen skal udrulles, med en *bicep* fil som skabelon, kompileres det til en *Azure Resource Manager (ARM) template*. Som er den endelige sandhedskilde for den infrastruktur der bliver udrullet til cloud provideren. *ARM templaten* er en *json* fil, indeholdende alle konfigurationer som ressourcen på *Azure* kommer til at have – Om det er deklareret i *bicep* eller ej.

[1] https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/overview?tabs=bicep

Opsummeret set, består BetterBoards softwareprodukter af en monolitisk konjektur, mellem en frontend- og backend applikation. Bestående henholdsvis af *Vue.js* og *ASP.NET*.
Backendapplikationen gør brug af en *MongoDB* database til data persistens. Ydermere er der en række *Azure Functions* (*AZF*) applikationer til formål for at udføre CPU-tunge eller tredjeparts-afhængige processer. Kommunikationen mellem *AZF* applikationerne og monolitten, foregår primært gennem *amqp*-protokollen som følge af *Azure Service Bus*.

Læringsmål

Jeg vidste ikke meget om firmaet på forhånd. Udover hvad jeg kunne informationssamle mig til gennem digitale medier og en praktiksamtale med CTO'en.
Dog vidste jeg at jeg ville blive inkorporeret som teammedlem i udviklingsafdelingen. Hvis primære arbejdsopgaver består af at udvikle og drifte bestyrelsesportalen og datarum.

Dette betød at jeg kunne tage udgangspunkt i de kompetencer mine kollegaer indehaver. Og bestræbe mig efter at opnå disse. Ydermere, var der nogle arbejdsopgaver jeg blev påduttet, som følge af min baggrund og interesse for *DevOps* og softwareinfrastruktur. Dermed opsatte jeg disse læringsmål:

- Udvide min forståelse for at udvikle og drifte en eksisterende *ASP.NET* monolit. Hertil overholde udviklingsafdelingens software retningslinjer og -regler.
- Udvide min forståelse for *Vue.js*, og anvende den til BetterBoard softwareprodukternes brugergrænseflade.
- Lære hvorledes *bicep* kan bruges som *infrastructure-as-code* og udrulle ressourcer på *Azure*.
- Blive bedre bekendt med *Azure* som platform til softwareinfrastruktur.
- Arbejde som teammedlem i et udviklingsteam, der benytter en kontinuerlig udviklings metodologi[2]. Samt deltage i den interne software kvalitetssikring, der forekommer gennem *pull requests*.

Disse læringsmål anså jeg som kompetencer jeg ikke tilstrækkeligt opfyldte, da jeg startede min praktik. Men havde ambitioner om at opnå som følge af afrundingen på min praktikperiode.

[2] Også bedre kendt efter det engelske term: Continuous Development Methodology

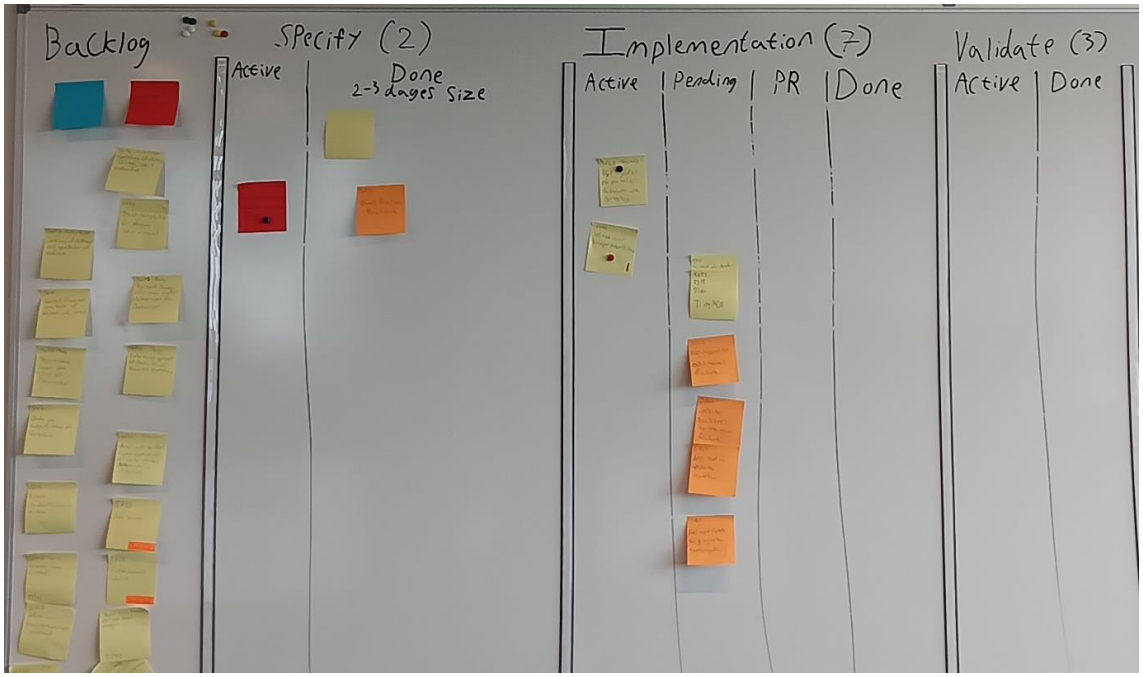
Arbejdsopgaver

Som fullstack-udvikler i et team. Har majoriteten af mine arbejdsopgaver bestået af at implementere *features* eller *bugfixes*, på BetterBoards hovedprodukt: Bestyrelsesportalen.

Nye *features* eller *bugfixes* på bestyrelsesportalen eller datarum, bliver primært løbende oprettet af enten salg- eller kundesupport afdelingerne. Herefter bliver de prioriteret, ved nogle ugentlige møder, af salg- og teknisk leder.

Dette resulterer i at man som udvikler, har en strømlinet arbejdsproces.
Man tager blot en *task* fra *backloggen*. Og implementerer kravene fra start til slut.

Forneden ses et billede af det *Kanban* board som fremhævede udviklingsafdelingens arbejdsgang:



Figur 2: Fysiske Kanban Board

En *task* der blev taget fra *backloggen*, skulle først overvejes om den bør eller kan dekomponeres yderligere. Såfremt at enhver *task* eller *subtask* estimeres til højst to eller tre dages tidsinvestering, fra den er påbegyndt, til den er klar til at blive sat i produktionsmiljøet.

Når en *task* er under *Active* kolonnen på *Kanban* boardet. Signalerer det at en udvikler i øjeblikket er i gang med at implementere eller teste.

Når den enkelte udvikler menes at en opgave er færdig: Sendes den til intern kode review, ved at åbne en *pull request* mod *main*-branchen i source control.

En opgave der står under *Done* kolonnen i *Implementation*, er klar til at blive sat i produktionsmiljøet. Som sender den til *Validate*, for at gennemse opgaven en sidste gang: Om alt agerer som det skal, også i produktionsmiljøet.

Arbejdsopgaver af denne natur indebar typisk at skulle arbejde med monolitten i *Vue.js* eller *ASP.NET*.

Virtualisering, Infrastruktur og DevOps

Jeg arbejdede også på at optimere eksekveringsplanen for de CI / CD pipelines der eksisterer for softwaren.
Som følge af at BetterBoard er et Microsoft-hus, benyttes *Azure Pipelines* på *Azure DevOps* til at definere de pipelines:

```yaml
trigger: main


pool:
  name: ContainerAgents
stages:
  - stage: Build
    displayName: 'Build for Release and Publish Build Artifact'
    jobs:
      - job: Build
        workspace:
          clean: all
        steps:
          - task: NuGetAuthenticate@1
            displayName: 'NuGet Authenticate'
          - task: UseDotNet@2
            displayName: 'Use .NET SDK version 7.0.100'
            inputs:
              version: 7.0.100
          - task: DotNetCoreCLI@2
            displayName: 'Restore Project'
            inputs:
              command: restore
              projects: '**/*.csproj'
              feedsToUse: config
              nugetConfigPath: Nuget.config
...
```

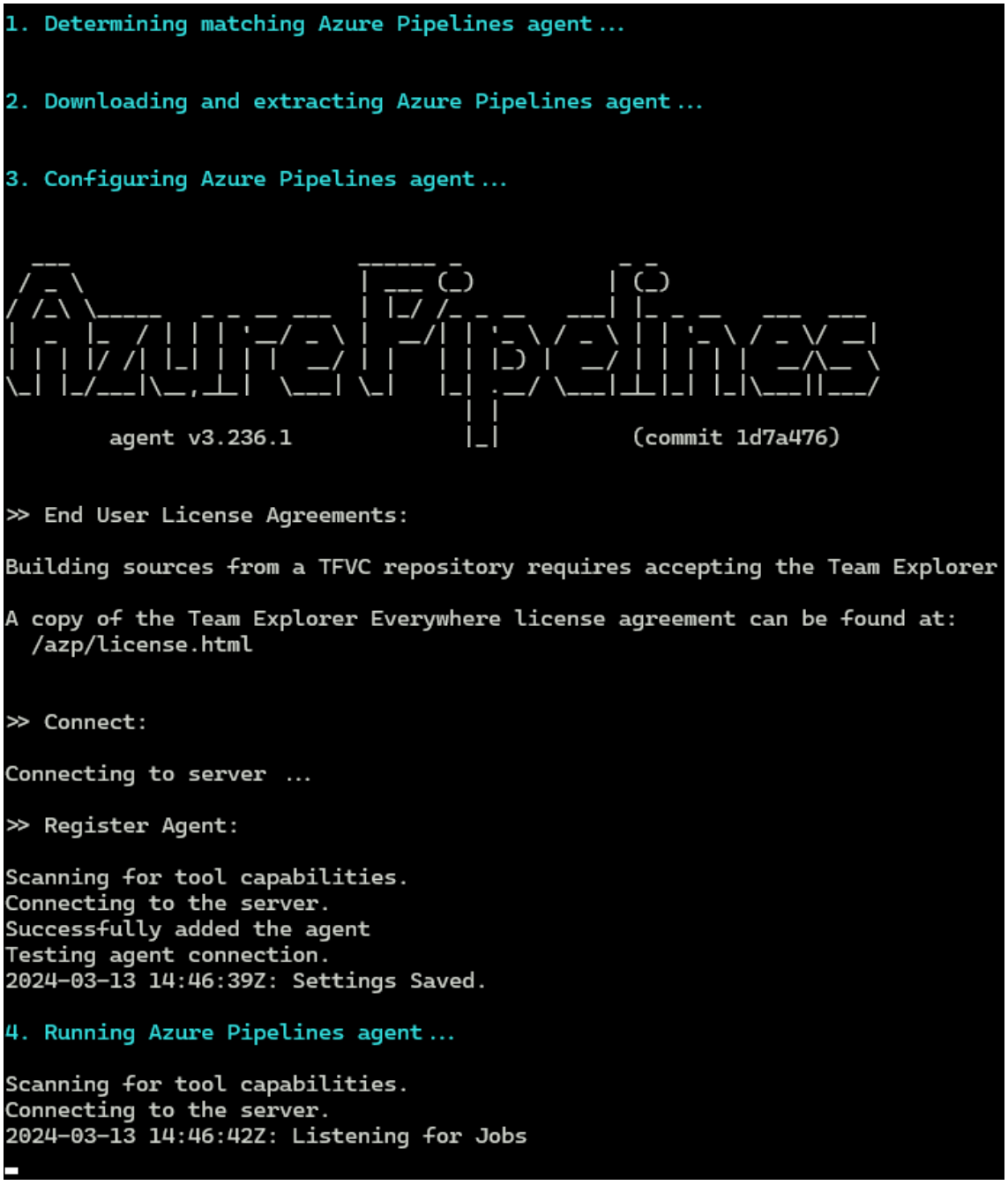
Figur 3: Snippet af Azure Pipeline

Jeg opsatte en *dockerfile* der beskrev et *image* vi kunne bruge til *containerized self-hosted agents*. Til formål for at kunne køre pipeline jobs, uden at have flaskehals i eksekveringsplanen. På baggrund af styrken ved dynamisk horisontalt skalerende containere.

```
FROM mcr.microsoft.com/azure-cli:2.56.0

RUN apk update
RUN apk upgrade
# General purpose linux packages
RUN apk add bash curl git icu-libs jq docker

# Processor architecture
ENV TARGETARCH="linux-musl-x64"

WORKDIR /azp/

COPY ./start.sh ./
RUN chmod +x ./start.sh

ENV AGENT_ALLOW_RUNASROOT="true"

ENTRYPOINT ./start.sh
```

Figur 4: Self-hosted Azure pipeline agent Dockerfile

Imaget afhang af et *shell* script til at udføre sin *agent* logik. Således at man fik dette output i logs fra en kørende *container* baseret på *imaget*:



Figur 5: Self-hosted Azure pipeline agent container logs

Virtualisering til Lokalt Udviklingsmiljø

Som benævnt under overskriften ”Teknologi”, er der en række *AZF* applikationer som systemet består af.

For at gøre det nemmere at arbejde med systemet i et udviklingsmiljø, blev jeg påduttet at virtualisere de hyppigst brugte applikationer, til containere.

For at opnå ovenstående, udrullede jeg et privat *image registry*. Skrevet i *bicep* og udrullet som *Azure*'s domæne-specifikke ressource: *Azure Container Registry*:

```
param location string

param acrName string


resource acr 'Microsoft.ContainerRegistry/registries@2023-07-01' = {
  name: acrName
  location: location
  sku: {
    name: 'Basic'
  }
}
```

Figur 6: Azure Container Registry Bicep File

Registry ressourcen skulle indeholde de forskellige *images* og deres *versioner*. Hertil de vigtigste aspekter værende dynamiske *version tags* som '*stable*' eller '*latest*'.

Når en ny version af en given *AZF* applikation er blevet kreeret, blev den kompileret, i en *Azure Pipeline*, til et artefakt. Som da kunne bruges både til udrulning samt bygning af et nyt *image*. Dette *image* ville da få et unikt *version tag*, samt tildelt *tagget* '*latest*':

```
  - stage: Dockerize

    displayName: 'Dockerize Published Artifact to Image and Push to
ACR'
    jobs:
      - job: Dockerize
        steps:
          - task: DownloadBuildArtifacts@1
            displayName: 'Download Build Artifact: drop'
            inputs:
              buildType: 'current'
              downloadType: 'single'
              downloadPath: $(Build.Repository.LocalPath)
          - task: ExtractFiles@1
            displayName: 'Extract artifact'
            inputs:
              archiveFilePatterns: '**/*.zip'
              destinationFolder: '$(Build.Repository.LocalPath)/pub-
lish_output'
              cleanDestinationFolder: true
          - task: Docker@2
            displayName: 'Login to ACR'
            inputs:
              command: login
              containerRegistry: <registry>
          - task: Docker@2
            displayName: 'Build and Push'
            inputs:
              repository: <repository>
              command: buildAndPush
              Dockerfile: 'release.Dockerfile'
              buildContext: $(Build.Repository.LocalPath)
              tags: |
                latest
                $(Build.BuildId)
```

Figur 7: Azure Pipeline containerizer et publiceret artefakt

Dette betød at en *docker compose* kunne udnyttes, hvori ethvert *AZF* applikation *image* havde *versionstagget* '*latest*', for at opnå konsistens på tværs af de lokale miljøer.

Opsummeret set, arbejdede jeg også med en hel del *Azure*, for softwareinfrastruktur. Både gennem den domæne-specifikke *Azure CLI*, brugergrænsefladen på platformen, og *infrastructure-as-code* gennem *bicep*.

Refleksioner

Konjekturen mellem den strømlinet arbejdsproces der kommer fra *Kanban*, og de spontane beslutninger der kom fra dagligdagens *stand-up*. Resulterede at jeg aldrig havde to uger der så ens ud. Det er ikke nødvendigvis en hverdagsrytme der passer til alle individer. Men det var en rytme som jeg satte stor pris på.

Faktummet at det er et regelmæssigt lille firma mh.t. medarbejderantal. Betyder at man som individ, bliver nødt til at kunne kapere en bred vifte af arbejdsopgaver. Som netop spiller ind i mine kollegaers stilling: Fullstack-udvikler.

De læringsmål jeg satte for mig selv, har været ambitiøse. Men jeg kan med glæde sige at jeg har rykket mig markant længere end jeg havde regnet med. Der har været rummelighed til at kunne fordybe sig individuelt, men også mulighed for at få hjælp hurtigt – Uden den helt store bureaukratiske kommunikationslogistik der kan være i store virksomheder.

Monolit

- Fik jeg udvidet min forståelse for at udvikle og drifte en eksisterende *ASP.NET* monolit. Hertil overholde udviklingsafdelingens software retningslinjer og -regler?

Ja. Selvom jeg havde en baggrund inden for *.NET*, havde jeg minimal erfaring med at videreudvikle på et eksisterende stort projekt. Det er et helt andet aspekt at skulle fortolke eksisterende kode, frem for at være med fra starten af projektet. Ydermere bygge videre på projektet og forholde sig til arkitektur- og kodeprincipper.

Min første personlige strategi for at videreudvikle, gik ud på at læse en betragtelig mængde kode. Men jeg blev hurtig sat på bedre tanker af min kollega:

> *”Spring ud i at implementere noget i stedet for at læse det hele. Så kvalitetssikrer vi det i en pull request!”*

Hvilket fik mig til at indse umuligheden jeg var i gang med: At læse og forstå over hundrede tusinde linjer kode, jeg ikke selv har været med til at skrive. Før jeg nogensinde skulle gå i gang med at implementere noget.

Vue.js

- Fik jeg udvidet min forståelse for *Vue.js*, og anvendt det til BetterBoard softwareprodukternes brugergrænseflade?

Ja. På trods af at dette læringsmål kan ses som trivielt, af et individ med en anden baggrund end jeg, var dette – og er stadig – et vanskeligt læringsmål. Da jeg sjældent arbejder godt sammen med visuelle aspekter, artikuleret gennem kode.

Med samme strategi som at arbejde med *backend* monolitten. Sprang jeg ud i at videreudvikle *frontend* projektet. Med spædt forhåndsforståelse for komponent-baseret *javascript* frameworks som *Vue.js*, betragtede og indlærte jeg alle nye aspekter jeg så i den eksisterende kodebase. Og forsøgte mig bedst muligt at videreudvikle alt efter den gense softwarearkitektur.

Baseret på min kollegas kvote foroven. Så kunne det ikke gå *helt* galt. Da alle ændringer på kodebasen i sidste ende skulle godkendes af en anden erfaren fullstack-udvikler.

Jeg indså at jeg skulle undertrykke mit imposter syndrom. Og have et simpelt mål i baghovedet: Få skrevet noget kode.

Infrastructure-as-Code

- Fik jeg lært hvorledes *bicep* kan bruges som *infrastructure-as-code* og udrulle ressourcer på *Azure*?

Til en vis grad. Konceptet bag *IaC* var ikke nyt for mig. Men anvendelsen af det var.
Jeg fandt hurtigt ud af at man bør kvalitetssikre sin *IaC* i høj grad, før man vælger at udrulle det. Ét aspekt er når det gælder kritisk infrastruktur i et produktionsmiljø. Et andet, er det faktum at det er en tidskrævende proces. Fra at *bicep* filer er udrullet som *Azure* ressourcer.

Som benævnt tidligere kompileres *bicep* til *ARM templates*.
Ydermere er en syntaks fejlfri *ARM template* ikke nødvendigvis en valid *Azure* ressource. Derfor er der mange skridt i udrulningsprocessen hvor det kan slå fejl.

Et værktøj i ærmet af kvalitetssikringsprocessen var dog at man kunne validere om ens *bicep* fil bl.a. kunne kompilere korrekt nok til en *ARM template*. Som man da kunne gennemse for eventuelle fejl, før man skød den hvide pil efter *cloud provideren*:

```
az deployment sub validate --file <bicep_file> --
location <location>
```

Figur 8: Azure CLI eksempel

Azure

- Blev jeg bedre bekendt med *Azure* som platform til softwareinfrastruktur?

Til en vis grad. *Azure* som *cloud provider* og platform er enorm. Jeg var godt klar over at dette var et ambitiøst læringsmål. Baseret på at jeg på forhånd var klar over at der eksisterede certifikater, blot for at have ekspertviden omkring netop dette emne.

Jeg havde minimal forhåndsviden for *cloud providere* som *Azure*. Den erfaring jeg primært havde med udrullede miljøer, var af små *on-premise* karakteristika. Det gav modstrid i forhold til en af mine kerneprincipper: Jeg vil vide hvordan *alt* fungerer.

Dog fik jeg med succes, konfigureret og udrullet adskillige *Azure* ressourcer, både gennem *IaC* og *CLI*. F.eks. *Azure Container Instances, -Apps* og *-Registries*, for at nævne en håndfuld.

Jeg føler mig mere sikker i at arbejde med *cloud providere*, i stedet for at altid have hardwaren i nærheden. Men jeg føler stadig at jeg er langt fra at kunne tilegne mig et certifikat på emnet.

Kontinuerlig Udviklings Metodologi

- Fik jeg arbejdet som teammedlem i et udviklingsteam, der benytter en kontinuerlig udviklings metodologi. Samt deltage i den interne software kvalitetssikring, der forekommer gennem *pull requests*?

Ja. Det var et nyt koncept for mig, ikke at have tidsbaseret *sprints* som er kendetegnende for en *scrum* baseret projektstyringsmetode.

En ting der nagede mig, var dog at man ikke havde nogle milepæle eller *burndown charts*, som er kendetegnende for *scrum*. Man kunne hurtigt komme under den forvaltning at projektet aldrig kom fremad.

Selve *Kanban* processen, var nyligt tilegnet i udviklingsafdeling da jeg kom til. Og det blev implementeret ud fra den basis at selve hovedproduktet var essentielt ”færdigt”.

eSom nævnt tidligere, begik majoriteten af ens arbejdstid som udvikler sig på at videreudvikle dette hovedprodukt, i form af små individuelle *userstories* eller *bugfixes*. Dette betød at der sjældent var store implementerings milepæle at stræbe efter.

Med hensyn til den interne kvalitetssikring, følte jeg til en start en form for *imposter syndrom*. Da jeg essentielt set kom temmelig ”grøn” og dikterede kvalitetskriterier.
Men som jeg senere fandt ud af: Har vi, som udviklere, forskellige baggrunde og interesser. Og kan derfor ofte byde ind med substantielt input, uanset erfaring – Til en vis grad.

Konklusion

Opsummeret fortalt, har det været en enormt berigende praktik.

Som nævnt tidligere, er det et enormt positivt aspekt for mig, at få lov til at have sådan en bred vifte af arbejdsopgaver. Essensen ved at være en generalist eller fullstack-udvikler, er noget som jeg altid har og vil bestræbe mig efter.

Personligt, ville jeg foreslå andre kommende softwareudviklere at erfare sig med *cloud providere* som *Azure* eller *AWS* tidligt i deres karriere. Især fordi majoriteten af softwarefirmaer har infrastruktur distribueret gennem førnævnte platforme.

Jeg vil foreslå at udnytte digitale stipendier, som forskellige medie platforme tilbyder, til netop at få lidt *cloud provider credits* at lege med.
Personligt gjorde jeg brug af *GitHub Student Developer Pack* på daværende tidspunkt.

Bilag

Logbog

Link til min logbog kan findes her.

Jeg skrev logbog løbende, gennem min praktikperiode, i min favorit *markdown* editor *Joplin*. Som jeg da har uploadet som en selvstående *markdown* fil.