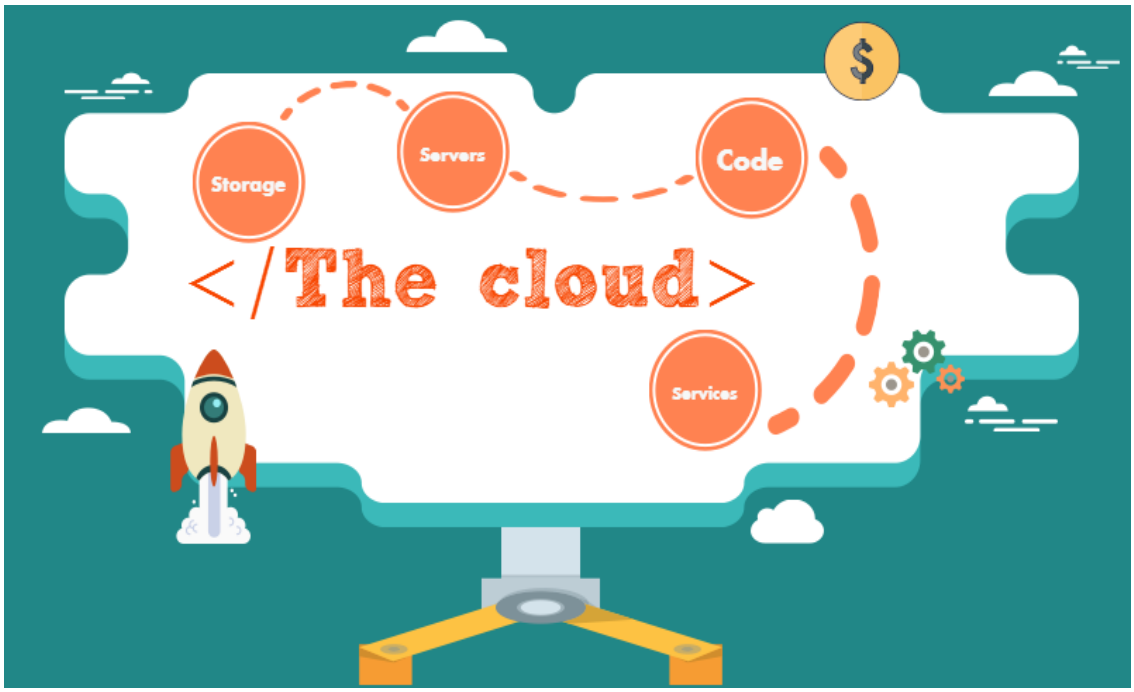
Migrering og analyse af fordele ved cloud services, herunder Amazon Web Services

Migration and analysis of advantages by using cloud services, especially Amazon Web Services



Bacheloratfhandling ved DTU Compute
Navn: Emil Hein Eriksen
Studienr: s120561

Udarbejdet i samarbejde med:
Virksomhed: FT Group
Vejleder: Stig Høgh

Afleveret: 11/01/2016

Forord:

Tak til min vejleder fra DTU Stig Høgh. Han har guidet mig til mit valg af emne og igennem processen med udarbejdelse af rapporten.

Der vil igennem hele opgaven for så vidt muligt blive benyttet open source software, for at sikre at forskellige tests og systemer kan genskabes uden større omkostninger. Der er i testen af AWS blevet brugt en konto i forbindelse med FT Group.

Opgaven er til dels udarbejdet i samarbejde med FT Group. Der skal hertil lyde et tak for brug af bl.a. ressourcer, der har hjulpet udarbejdelsen af denne opgave. Al kode i denne opgave tilhører FT Group eller undertegnede og må kun distribueres med speciel tilladelse.

Indholdsfortegnelse

Liste over figurer

Abstract

In this paper, the new paradigm of cloud computing is being investigated. The possibilities are virtually endless, and all companies should at least be looking into, what the cloud has to offer. Every company online, are using some kind of cloud computing. However, it is not everyone who knows, how to really be a part of it and take advantage of it.
A practical view of how migration to the cloud could look like, and pitfalls one could encounter in the process is being presented. Both the process of moving applications and databases are being displayed in an attempt to guide others, if the decision of moving partially or fully to the cloud has been made.
The investigations shows through tests, that there a many advantages to gain, if a company starts using the cloud. Cost efficiency and flexibility are some of the greatest advantages one can benefit from, if done right. Some amount of research from the individual companies needs to be done, in order to avoid obstacles later on in the process.

Problemformulering

English
Using “the cloud” is becoming more and more popular amongst many companies. How does one start using the cloud as important part of the IT infrastructure? Old routines can make it difficult for companies to get started if the resources is scarce. This is however a good reason to get started on using the cloud to help your IT infrastructure. If you decide to use the cloud what advantages can one expect, if not already using it? Moreover the following questions will also be examined.

- Who can use “Cloud Computing”?
- Which advantages can one expect, when moving to the cloud?
- Which areas a potential dangerous when moving to the cloud?
- How will the future look like when talking about hosting of servers and other it-infrastructure?

Dansk
At bruge “skyen” bliver mere og mere populært blandt mange firmaer. Men hvordan kommer man i gang med bruge de mange ressource som en del af ens it infrastruktur? Vedligeholdelse af gamle systemer, kan gøre det svært for firmaer, at finde ressourcer til at komme i gang med at bruge skyen. Dette er dog endnu en grund til udforske skyen til at hjælpe med forbedre it-infrastrukturen for mange firmaer.
Til at besvare spørgsmålet vil jeg gennemgå både teori og praksis forbundet med at migrere til skyen, samt fremhæve nogle af de vigtigste features der er forbundet med dette. Herunder vil følgende spørgsmål også blive undersøgt.

- Hvem kan bruge ”*Cloud Computing”*?
- Hvilke fordele kan et firma forvente at få ud af, at bruge skyen?
- Hvilke områder er særligt ”farlige” når det kommer til at flytte til skyen?
- Hvordan vil fremtiden se ud ift. hosting af serverer og andet it-infrastruktur

Motivation

Flere og flere firmaer sidder med gamle løsninger, som kører stabilt for nu, men i fremtiden ikke kan klare den stigende forespørgsel og trafik fra en globaliseret verden. Størrelsen på en virksomhed er ikke længere det eneste der gør den til førstevalg hos kunder. Det er i høj grad muligheden for at kunne reagere hurtigt og være så fleksibel som muligt. Alt dette skal helst ikke være en udvikling som ikke kan håndteres pga. virksomheders it-infrastruktur. Derfor er cloud computing blevet en mere og mere almindelig måde at håndtere it-infrastruktur på.

Miljøspørgsmål er vigtige for mange nationer og firmaer. Derfor er det vigtigt at der spares på energien og specielt CO^2 udledningen ved at spare på driften af it infrastrukturen. Store datacentre har muglighed for større effektivitet og energiforbrug end nogensinde før.

Derfor er den moderne udfordring at bruge de muligheder der tilgængelige for at optimere virksomheders arbejdsgange og processer. Et nyt paradigme man ikke kan komme uden om er cloud computing. Dette kan være med til at hjælpe en masse virksomheder, men også være med til at formindske udledningen af $CO^{2.}$

Det nye paradigme giver mange spændende muligheder. For at alle virksomheder kan begynde at udnytte dem, er det vigtigt at nogle af de nye processer er fuldt forstået. Mange firmaer er endnu ikke helt trygge ved at flytte deres it-infrastruktur. Derfor skal emnet belyses og forhåbentligt skal opgaven her, hjælpe til at give et indblik i de forskellige muligheder der tilbydes.

Denne opgave skal læses som en hjælp til, at komme i gang med at bruge ”cloud-løsninger” til at opfylde forskellige behov. Dette skal ikke ses som en udtømmelig guide til alle tjenester der udbydes, men skal dog give indsigt i nogle af de vigtigste features. Der vil som udgangspunkt bliver taget afsæt i Amazons Web Services (AWS). Selvom dette kun giver en afspejling af en enkelt udbyder, vil det dog være med til at tegne et generelt billede, af de forskellige services der tilbydes af mange forskellige udbydere.
Tests samt resultater vil bliver præsenteret som underbyggende argumenter for eventuelle fordele ved brugen af visse web services.
Når der i mange af afsnittene bliver snakket om ressourcer, skal det forstås i en bred forstand. Det skal forstås som værende en samlet betegnelse for både computerkraft, mandetimer, knowhow og mere.

Indledning og baggrund

Hvad er Cloud Computing?

"Cloud Computing" er stadig relativt nyt og kan som analogi sammenlignes med internettet. Det kan ikke beskrives som én ting, men derimod som en samlet beskrivelse for alle de nye muligheder det nye paradigme tilbyder.

At "flytte ting til skyen" eller "Cloud hosting" er vendinger som bliver brugt i mange sammenhænge og er på mange måder for abstrakt til at beskrive hvad det egentlig handler om. Det hele handler om IT infrastruktur. Selvom "Skyen" i sig selv er et abstrakt ord, er det vigtigt at holde for øje, at der stadig er tale om rigtige computere, i rigtige datacentre, der skal bruge strøm og køling ligesom alle andre computere. Når ting flyttes til skyen, overlader man egentlig bare noget eller al ansvaret til en web hosting udbyder.
Det er også værd at bemærke at "*Cloud computing"* er en frase der bliver brugt om mange forskellige ting.
Aspekter af "*Cloud Computing"* omhandler bl.a. forøgelse af ydelse, svartider, sikkerhed, osv. Store firmaer som Amazon, Google og Microsoft tilbyder alle forskellige former for "Cloud Computing" services. Samlet står disse tre udbydere for en stor andel af det samlede udbud. De tre udbydere tilbyder også mange af de samme services. Derfor bliver disse tre også tit sammenlignet.

"*Cloud Computing*" kan derfor ikke siges at være kun en ting, ligesom internettet ikke kan beskrives som værende en enkelt ting. Set fra et bredt synspunkt har det nye paradigme i sinde at gøre ting bedre, billigere og nemmere for enkeltpersoner eller virksomheder der vælger at bruge det.

"*Cloud Computing*" kan opdeles i både hardware og software services. Disse giver tilsammen kunder helt nye muligheder, når der skal udvikles nye systemer. Nogle skal udelukkende bruge hardware, mens det mest almindelige er at bruge en blanding af både hardware og software.
Hele industrien er vokset til at være en "multi-billion dollar industry" og er stadig i

fremgang [1]. For at industrien kan blive ved med at vokse, er der brug for at få styr på terminologien omkring de forskellige services.

Barrieren bliver stadig mindre og mindre og der er principielt ikke noget der forhindrer virksomhederne i at bruge de nye ressource, som kan agere muskler for IT-Infrastrukturen i virksomheden.

Før jeg dykker helt ned i hvad "*Cloud Computing*" specifikt kan tilbyde, er en smule kontekst og terminologi er nødvendig.

Skyen tilbyder både services inden for hardware og software. Som samlet definition vil jeg bruge en model udformet af NIST[1]. De definere cloud computing, som bestående af følgende tre modeller, i 2010:

Figur 1 Typer af cloud services

Infrastructure-as-a-service eller Iaas betyder at kunder køber en del af den infrastruktur store cloud udbydere som Amazon har opbygget. Amazon har datacentre i hele verden, hvilket tillader dem at udleje en dynamisk del af deres hardware til kunder. På denne måde kan kunder nemt leje en specifik mængde hardware til eget brug. Dette er ofte i form af cpu regnekraft, lagerplads eller memory. Dette giver mugligheden for at kunder kan leje hardware på samme måde som man lejer en bil. Man betaler kun for, hvad man bruger. En af de store fordele ved dette er at man kan købe præcis den mængde hardware man har brug for, uden at skulle vælge en bestemt pakke, som reelt set ikke giver den individuelle

1 National Institute of Standards and Technology

kunde den bedst mulige løsning.

Software-as-a-service også kaldt Saas gør det muligt for kunder at leje software, udviklet af udbyderen. Denne software kører på udbyderens hardware og er i teorien blot et ekstra abstraktionslag oven på deres hardware. Ved brug af denne service kan man benytte deres services helt uden at bekymre sig om, hvilken infrastruktur der holder servicen kørende. Disse slags services tilgås som regel igennem en webbrowser eller et API.

Platform-as-a-service eller Paas er tæt forbundet med Iaas og gør det muligt for kunder at benytte en hel platform til at udgive applikationer eller hjemmesider der bliver hostet på en underliggende infrastruktur. Kunden har derfor ikke 100 % kontrol over den underliggende hardware, selvom en hvis konfiguration ofte er mulig. Denne mulighed kræver at kunden selv har hele kildekoden der skal udgives og uploadet, samt alle referencer den pågældende kildekode skal bruge. Denne type service bliver ofte brugt af virksomheder, til at publicere egne applikationer og systemer.

Cloud computing bliver i mange tidsskrifter og magasiner brugt som et buzzword og som følge heraf glemmer folk ofte, at det blot er hardware der kommunikere på en måde der får det til at se lidt mystisk ud (virtualisering). Hvad enten du har din egen server, computer eller bruger skyen er det vigtigt at huske at enhver kode altid bliver afviklet på en fysisk computer et sted i verden. [2]

For at få en bedre forståelse for hvad ”Cloud Computing” kan være, se her følgende eksempel:

1. Microsoft online
2. Gmail
3. Dropbox

Disse services er alle eksempler på “*Cloud Computing”.* Fælles for dem er, at de udadtil viser en ”simpel” applikation, som i baggrundet bliver kørt på kæmpe store virtualiserede servere i store datacentre.

Amazon web services:

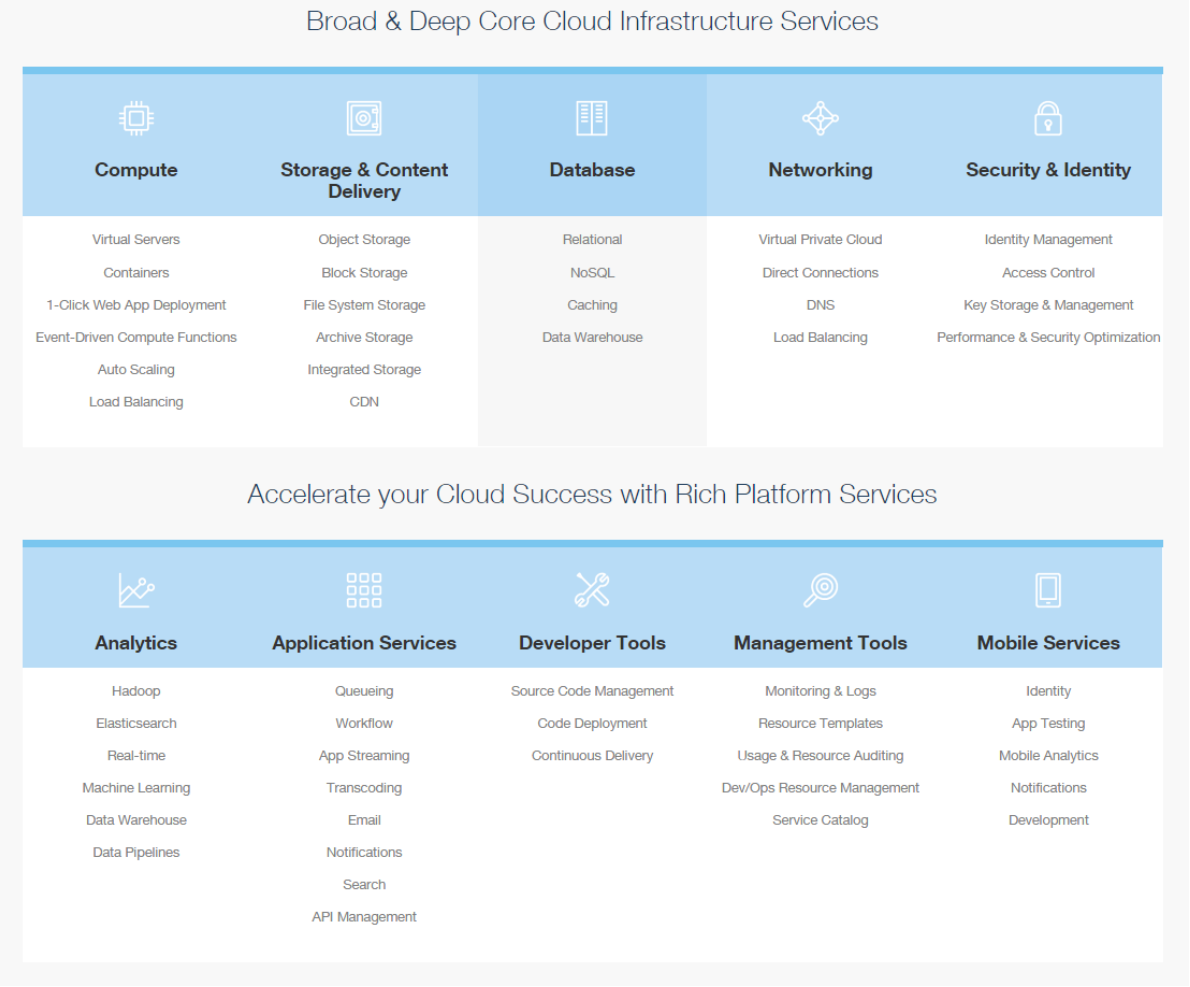
Amazon web services begyndte med at udbyde forskellige former for webservices i 2006. Det er specielt deres Infrastructure-as-a-service (IaaS) der har gjort dem til den absolut største udbyder af cloud services. [3]

Det er gældende for AWS, såvel som for mange andre udbydere, at størrelsen på disse firmaer betyder flere fordele for kunderne. Da AWS er så udbredt i hele verden, er mængden af informationer, guides og eksempler overvældende og en uudtømmelig kilde for viden og hjælp til selvhjælp. AWS tilbyder ligesom mange af de andre udbydere en gratisperiode, hvori man kan prøve mange af de services som man måske senere skal bruge i forretningsøjemed. Dette giver en unik mulighed for at teste hele platformen af, med ens egne systemer og applikationer, og på baggrund af dette, vurdere om det er noget man vil

udforske nærmere.

Foruden dokumentation og guides, tilbyder de større udbydere flere forskellige datacentre lokaliseret i hele verden. Dette betyder at man uden problemer kan ekspandere til andre dele af verden, uden at skulle foretage ændringer af ens infrastruktur. Hvis man er et globalt firma, kan der være mange fordele i at få hostet sine servere tæt på der hvor ens kunder er.

Antallet af webservices er stigende, og der kommer flere år efter år. Nogle af dem er til general brug, mens andre er mere specialiserede til specifikke formål. Herunder ses et overblik af de nuværende services der tilbydes af AWS.



Figur 2: overblik over AWS services

Features og muligheder

Selvom forskellige udbydere tilbyder forskellige services, er der dog nogle services som er så generelle at alle udbyder dem. Derfor vil jeg i følgende afsnit give en introduktion til disse services. Det vil være med udgangspunkt i AWS. Afsnittet skal give baggrund for nogle af de tests der vil blive præsenteret senere i opgaven, samt et indblik i nogle af kerneområderne af cloud computing.

I de fleste tilfælde vil mange af disse features skulle benyttes i forbindelse med hinanden for at give det bedste overblik og kontrol med ens samlede applikation. Senere vil jeg beskrive hvordan man kan migrere større systemer, så de benytter flere af de tilgængelige services.

Listen over features er langt fra fyldestgørende og der kommer hele tiden flere services til den samlede pakke. Derfor prøver jeg at holde mig til de vigtigste hovedpunkter. Jeg har valgt de områder som jeg mener der burde fokuseres på, hvis man første gang stifter bekendtskab med f.eks. AWS.

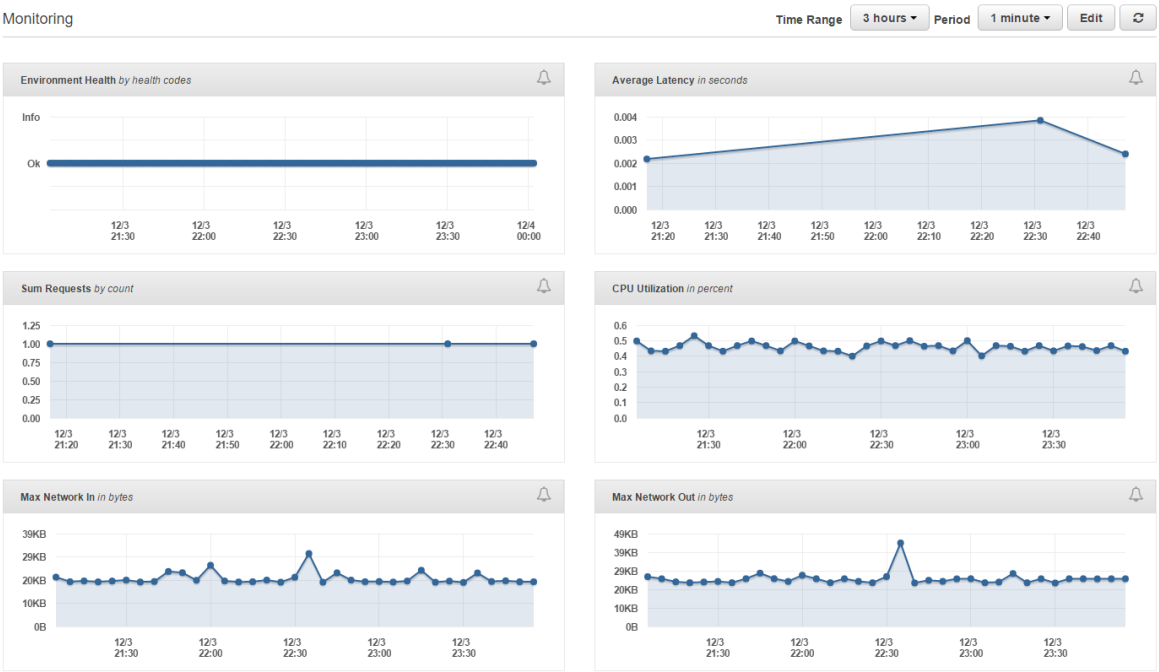
Monitorering af ressourcer

Med hosting i skyen er det blevet meget nemmere at monitorere ens services og systemer. Hvis et firma hoster servere selv, eller har dem igennem et andet firma, har overvågning typisk været en større opgave. Hvis ens services er spredt over mange servere eller de er af ældre dato, kan det være svært at få alle tænkelige overvågningsparametre, efter behov. Dette er blevet nemmere med cloud hosting, som har alle tænkelige parametre tilgængelige både til internt brug, men også til frit brug for kunden. Der er forskellige grunde til at cloud udbydere går så meget op i statistikker og overvågning af kundernes, men den vigtigste er penge.

Monitorering af kunders data er tæt forbundet med den prismodel mange cloud udbydere benytter. Modellen er konceptuelt nem at forstå. Man betaler for hvad man bruger. Selve modellen er beskrevet ydereligere i afsnittet omkring pris.

For at udregne hvad kunderne bruger, og derfor hvad de skal betale, har udbydere som AWS implementeret en omfattende platform, som kan monitorere alle dele af alle systemer, ned til mindste detalje.

Herunder ses et udsnit af hvilke data der er tilgængelige for monitorering af en virtuel server hostet hos AWS:



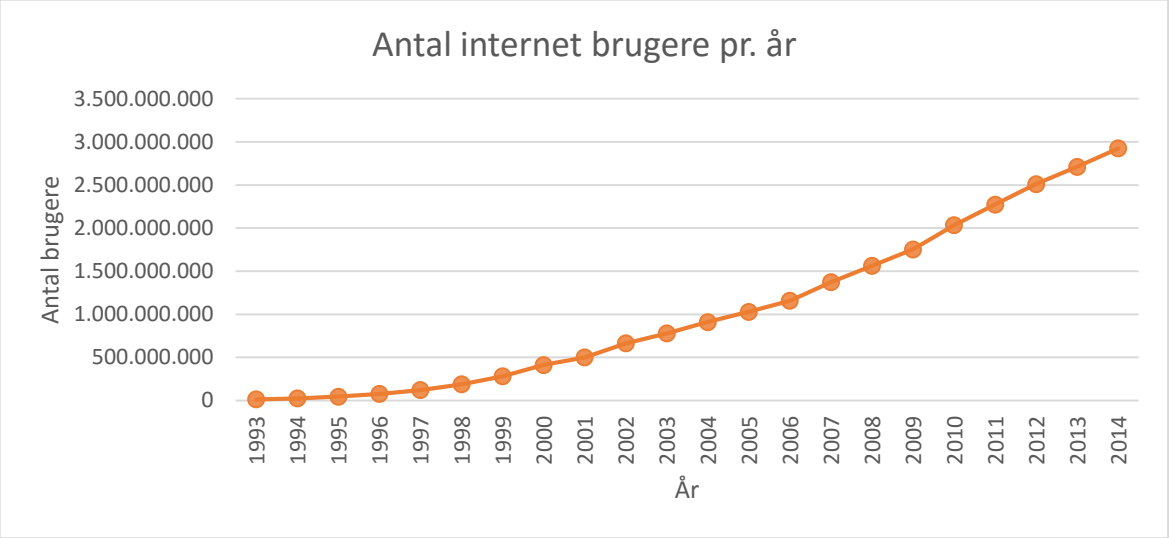
Figur 3: EC2 monitorering

Al den data og indsigt betyder også, at mere ansvar ligger hos kunden. I traditionelle hosting løsninger har en fast pris været aftalt for en fastlagt ressource. Med cloud hosting er ydelse ikke længere det eneste kunden skal holde øje med og være bekymret for. Da prisen i høj grad afhænger af de brugte ressourcer, er der nu et større incitament til at effektivisere ens løsninger og holde øje med at prisen ikke lige pludselig stiger pga. af en fejl. Man kan nemt forestiller sig et scenarie hvor en virksomhed har slået autoskallering

til, hvorefter efter en fejl udløser den mekanismer der starter flere ressourcer op. Dette vil resultere i en øget pris, selvom de ekstra ressourcer ikke er nødvendige.
Selvom cloud computing i høj grad har til formål at forsimple mange processer relateret til vedligeholdelse af infrastruktur og deployment af software, skal kunder samtidig være meget opmærksomme på hvilke ansvarsområder, der varetages af udbyderen. Udbydere som AWS gør meget ud af at give brugerne så meget selvstyrring som overhoved muligt, uden at gøre det for kompliceret. Derfor skal virksomheden være klar over, at et skifte til en cloud udbyder, ikke kun betyder mindre vedligeholdelse af infrastruktur og software, men samtidig også betyder mere vedligeholdelse af ting som f.eks. pris på ressourcer, effektivitet, monitorering, osv.

Skallering af ressourcer

En af de vigtigste elementer ved cloud computing er muligheden for fleksibel skallering. Skallering er er blot et udtryk for hvor mange flere eller færre ressourcer man skal bruge, for at udføre en stigende eller faldende arbejdsbyrde. Da der bliver flere og flere brugere, må visse typer af systemer forvente større udsving i belasting. Se nedenfor antallet af brugere.



Figur 4 antal brugere World Wide per år [4]

Som grafen ovenfor illustrere bliver der flere og flere internetbrugere på verdensplan. Dette betyder også at udsving i forespørgsler har potentiale til at blive større og større. Af denne årsag er autoskallering et uvurderligt værktøj.

Eksempel 1

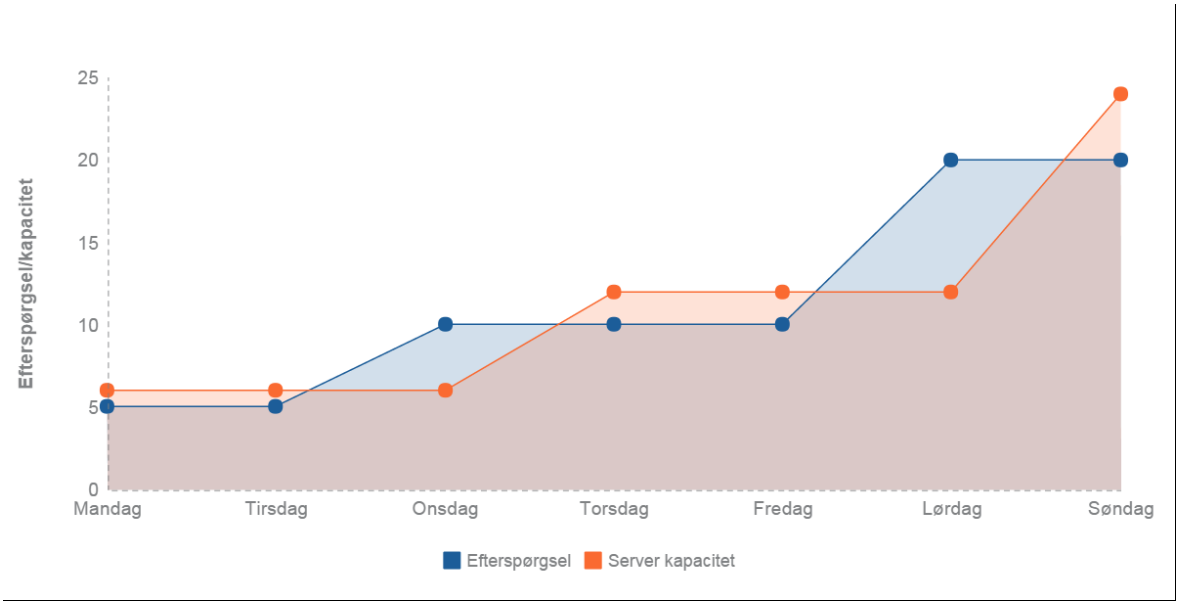
Som et klassisk eksempel betragt følgende:
Lad os antage, at et firma ejer to servere der til sammen kan svare på 100.000 forespørgsler i timen til deres hjemmeside. Grundet god reklame eller et uforudset opsving, får firmaet

pludselig 200.000 forespørgsler i timen over en periode på mange timer, og serverne lukker i værste fald helt ned.

Før ”Cloud Computing” var muligt, var firmaet enten nødt til at indkøbe flere servere til at klare den øgede trafik, eller ringe til deres serverudbyder og få opsat nogle flere. Dette er ikke blot dyrt, men også tidskrævende, hvilket fører til længere nedetid hos slutbrugeren. Hvis firmaet igen får flere forespørgsler er de nødt til at lave samme trick igen. Hvis de derimod får færre besøgene står de med en masse serverkraft som de ikke kan bruge til noget. Uanset omstændighederne er det en ufleksibel løsning, som alt afhængig af aftalen med serverudbyderne kan koste firmaet rigtig mange penge.
Med indtoget af Cloud computing, og i denne sammenhæng især AWS, er alle disse problemer blevet nemmere at takle for alle størrelser af virksomheder. Dette skyldes mekanismen autoskallering, som er blevet et af de mest populære og benyttede services.

Autoskallering

Autoskallering er et udtryk der bruges om muligheden for at opsætte regler for hvornår man ønsker at skalere ens løsninger. Selve reglerne er i sin natur statiske, men tillader dog den samlede ressource at være højst fleksible, så den passer til alle størrelser af trafik. Uanset hvilke parametre man bliver udfordret på, kan autoskallering hjælpe med at klare et dynamisk behov for ressourcer. Denne service er en af de vigtigste i hele samlingen af services. For at tydeliggøre problemstilling, betragt da nedenstående:

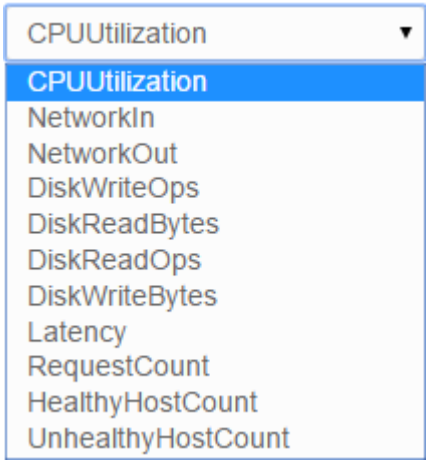


Figur 5 Eksempel på udbyd og efterspørgsel på ressourcer

Ovenfor er er et diagram som grafisk viser *Eksempel 1*. Her ses det tydeligt hvornår firmaet er i over og underskud på ressourcer. Arealet under linjen for serverkapacitet og over linjen for efterspørgsel svarer til uudnyttede ressourcer, som firmaet betaler for. Arealet over linjen for serverkapacitet og under linjen for efterspørgsel, svarer til at serveren(e) ikke kan

følge med efterspørgslen og der vil som resultat være nedetid eller lang ventetid for kunderne. I eksemplet vil der om lørdagen være et stort underskud af ressourcer.

Mulighederne for autoskalering er mange, selvom det overordnede mål er det samme. I et optimalt setup vil de to kurver i eksemplet ligge så tæt på hinanden som muligt.
Der er selvfølgelig flere forskellige måder hvorpå firmaer kan få de to kurver til ligge tæt op af hinanden. Selve systemet har selvfølgelig en del at sige, men også hvilken slags brugere, typer af forespørgsler, osv. har meget at sige i forhold til hvor nemt det er at få de to kurver til at ligger tæt på hinanden. Netop derfor tilbyder Amazons setup, at man kan opsætte regler der alt efter behov kan bestemme hvordan kurven for serverkapacitet skal se ud. For at opsætte disse regler skal man derfor være klar over hvordan ens system er belastet i forskellige tidsrum. Man kan som standard vælge følgende parametre til at udløse en op eller nedjustering af ressourcer.



Figur 6: Parametre for autoskalering

Disse parametre gør det muligt for firmaer, selv at justere sine systemer, så de passer til det ønskede behov. For at finde ud af hvilke parametre man er udfordret på, kan det være en god ide at se på monitoreringen af ens applikation. Man kan ud fra denne lave en analyse af hvordan man bedst muligt skalere ens løsning.

Ud over disse kan brugeren vælge at sætte tidsbestemte regler. Disse kunne f.eks. være at lukke alle testserverne ned om natten, og starte dem op igen hver morgen. På denne måde kører der ingen instanser som ikke bliver brugt, og derved sparer firmaet penge. Hvordan dette gøres vil blive beskrevet senere i rapporten.

Sikkerhed

Sikkerhed er ofte et af de punkter, kritikere påpeger som en af svaghederne ved cloud løsninger. Grænsefladen hvorpå man skal sikre sig mod angreb er større end ved traditionel hosting. Dette er udbyderene også klar over, derfor har mange gjort meget ud af, at sikkerheden er i top, så kunder har tillid til at deres systemer er sikre. Nogle af de

traditionelle sikkerhedshensyn er uden tvivl blevet mere sikre, mens andre nye trusler er opstået. Generelt kan sikkerhedsrisikoen ved cloud computing opdeles i tre kategorier.

1. Traditionelle trusler.
2. Trusler grundet skyens tilgængelighed.
3. Trusler grundet tredjeparts kontrol med systemer.

Hvad disse trusler nærmere betegnet er, og hvordan udbydere som AWS prøver at bekæmpe disse trusler bliver forklaret i følgende:

Som bruger af f.eks. AWS kan vigtigheden af sikkerhedsaspektet hurtigt blive overskygget af alle de andre muligheder, og det er derfor ikke svært at forestille sig, at nogle glemmer at tage sig tid til at følge de bedste standarder på området.
Med den nye frihed man har med cloud computing følger også et ansvar for selv at sikre dele af løsningen. Det er i mange tilfælde kunderne selv der står for adgangen til alle ressourcerne. Derfor skal man være ekstra forsigtig, hvis man ikke kender 100 % til de sikkerhedsrisici der er ved at hoste alle ens ressourcer hos f.eks. AWS.

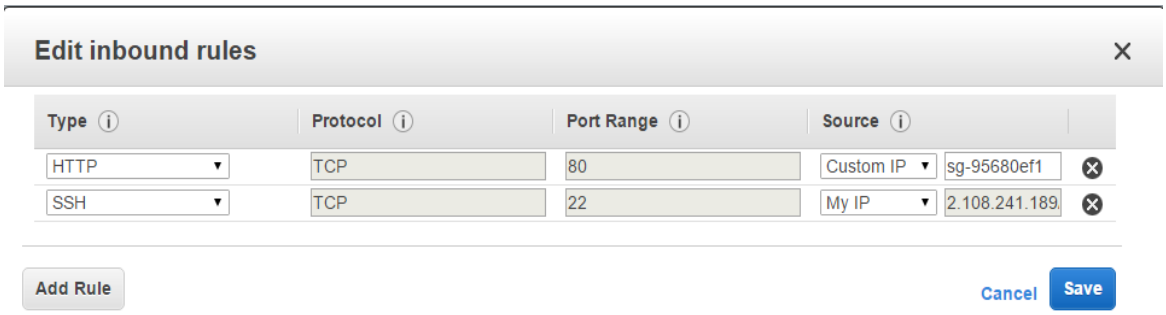
Adgang til alle ressourcer i AWS styres igennem modulet IAM (Identity & Access Management). IAM håndtere alt fra brugeres adgang til andre ressourcers adgang til hinanden. Med andre ord, så skal kunderne selv opsætte denne sikkerhed, så det stemmer overens med hvilke ønsker og krav man har til overordnede sikkerhed.

Den lidt sløret grænse for hvor ansvaret ligger i forhold til sikkerhed, kan være med at gøre det svært at gennemskue hvilke dele man selv skal holde styr på, og hvilke sikkerhedshensyn der allerede er håndteret for en. Lad os sige at data uheldigvis bliver slettet af udbyderen eller at udbyderen bliver hacket. Herefter kan det være svært for brugeren at bevise hvor fejlen er opstået.

I forhold til fysisk sikkerhed omkring hardware, er den langt højere hos firmer som Amazon, google og Microsoft end den typisk vil være med *in-house* hosting. Det er ekstremt vanskeligt at få fysisk adgang til de servere, udbyderne lejer ud. Dette betyder dog ikke at den samlede adgang og generelle trussel er mindre, blot at den fysiske sikkerhed er forhøjet. [5]

overordnet set er snitfladen en del større ved få alt infrastruktur hostet hos en cloud udbyder. Dette er både godt og skidt, da man også kan få nogle positive effekter ved at kunne kontrollere flere adgangspunkter, for ens brugere og ressourcer. Negativt set er muligheden for brist også større, hvis administrationen af adgangen ikke bliver udført ordenligt.

I AWS kan sikkerheden til ressourcer styres ved hjælp af bl.a. sikkerhedsgrupper. Disse grupper har et sæt indgående og udgående regler, som virker som en traditionel firewall:



Figur 7: Indgående regler for sikkerhedsgruppe

Ovenfor er et eksempel på indgående regler for en sikkerhedsgruppe. Dette betyder blot at en specifik ressource tilknyttet denne sikkerhedsgruppe, kun tager imod forespørgsler fra de angivne protokoller og ip adresser.

Lovgivning

En risiko ved at flytte sin hosting til skyen er, at lovgivningsmæssige aspekter omkring cloud computing endnu ikke er helt fastlagte i alle lande. Derfor kan der være visse udfordringer med lovgivning omkring f.eks. behandling af personfølsomme data på kryds af landegrænser. Visse lande har regler der forbyder firmaer at personlige data forlader landets grænser. Dette kan være svært at afgøre når tingene er hostet hos en udbyder der har datacentre i hele verden.

Safe harbour aftalen omhandler bl.a. at amerikanske firmaer skal forpligte sig til at beskytte EU borgeres persondata, med ligeså stor forsigtighed som EU landende. Dette er blot en af mange regler på området omkring beliggenhed og overførsel af persondata. For større firmaer, som AWS, google og Microsoft, er det ikke et stort skridt at stille en sådan garanti, da de typisk har ressourcer nok til at efterkomme alle tænkelige krav.
Visse brancher har specielle krav om adgang til data og den præcise håndtering af dem. Dette kan være en afgørende faktor, der gør at det ikke er muligt at vælge en cloud udbyder til at varetage ens infrastruktur.

Pris

Et vigtigt aspekt forbundet med det at flytte til skyen, er prisen. Og det er muligheden for at spare penge, som tiltrækker mange virksomheder. En af de nye modeller i forhold til klassisk hosting, er muligheden for at betale for de ressourcer man bruger. Dette er noget som i mange andre sammenhænge er blevet helt almindeligt. Abonnementer til telefoner bliver i højere grad afløst af samme model, hvor kunden kun betaler for sit forbrug. Elregninger, vandregninger, benzin, løn, osv., bliver alle beregnet ud fra hvor meget man bruger eller arbejder. Derfor er der ingen grund til at IT infrastruktur ikke også kan fungere

på samme måde.
Generelt tilbyder klassiske udbydere en række forskellige opsætninger til en fast pris om måneden. Mens dette kan have sine fordele, i og med man kender sin udgift på forhånd, er det oftest kun til ulempe for kunden. Ideen om at en række opsætninger skulle passe perfekt til en lang række applikationer og usecases er forkert. Typisk vil man ende op med at betale for noget man ikke kan udnytte fuldt ud, eller som ikke er tilstrækkeligt til ens behov.

Det er dog vigtigt at understrege at moderne cloud hosting ikke kan sammenlignes direkte med at leje en dedikeret server, da der kan være andre ting der påvirker prisen. Dette kunne bl.a. omfatte support, ansvar og andre ting, som kunne gøre det fordelagtigt at leje en dedikeret server. At sammenligne priser på dedikerede serverer, virtuelle servere og cloud serverer er en jungle der kan være svær at navigere i, men herunder vil nogle af forskellene på prismodellerne blive beskrevet.

Selvom udbydere generelt ønsker at gøre det så billigt som muligt, kan der opstå interessekonflikter og udbyderene skal til at tænke i spilteori og psykologi for at optimere deres overskud. Udbyderene konkurrere ikke blot med andre udbydere, men også med sig selv og de andre prismodeller de udbyder.

Når man i begyndelsesfasen skal se på hvilken webserviceudbyder man skal vælge kan det være rigtig svært at sammenligne. Specielt overvejelser omkring prisen, som er et afgørende argument i valget om at skifte til skyen. For at kunne lave en begrundet beslutning, skal der kunne foretages en form for sammenligning og analyse af om det kan betale sige. Da kombinationsmulighederne for opsætninger er enorme, vil en sammenligning på en simpel opsætning ikke bidrage til forståelsen af forskellen, på prismodellerne i klassisk hosting over for cloud hosting. Hvad den nye prismodel derimod tvinger virksomheder til, er at tænke over hvad de egentlig bruger og har brug for. I et klassisk setup vil en hostet server typisk have en fast mængde lager tilknyttet, til en fast pris. Med en sådan aftale vil en virksomhed betale for den mængde lager uanset om de bruger den hele eller ej. Derfor betyder det også mindre hvad man har liggende, da det jo ikke koster noget ekstra, så længe man ikke er over sin faste grænse. På denne måde behøver virksomheder ikke at tænke nøje over hvordan og hvor meget data der gemmes. Når firmaer betaler for hvad de bruger, bliver de i langt højere grad opmærksomme på, hvordan der kan optimeres for at spare ressourcer og derfor også penge.

Store udbydere som AWS udbyder lagerplads for 0,007 dollars pr. GB data. Dette er ekstremt billigt og kun muligt fordi stordriftsfordele gør det billigere for Amazon at have globale datacentre kørende og derfor kan udlåne ressourcer billigt. Om et skifte kan give besparelser kommer meget an på den specifikke virksomhed.
Som eksempel viser *David Greenwood* [6] at en migreringen til skyen kan give store

besparelser for virksomheder, forudsat at optimale strukturelle omstændigheder tillader det.

Pris efter udbud og efterspørgsel

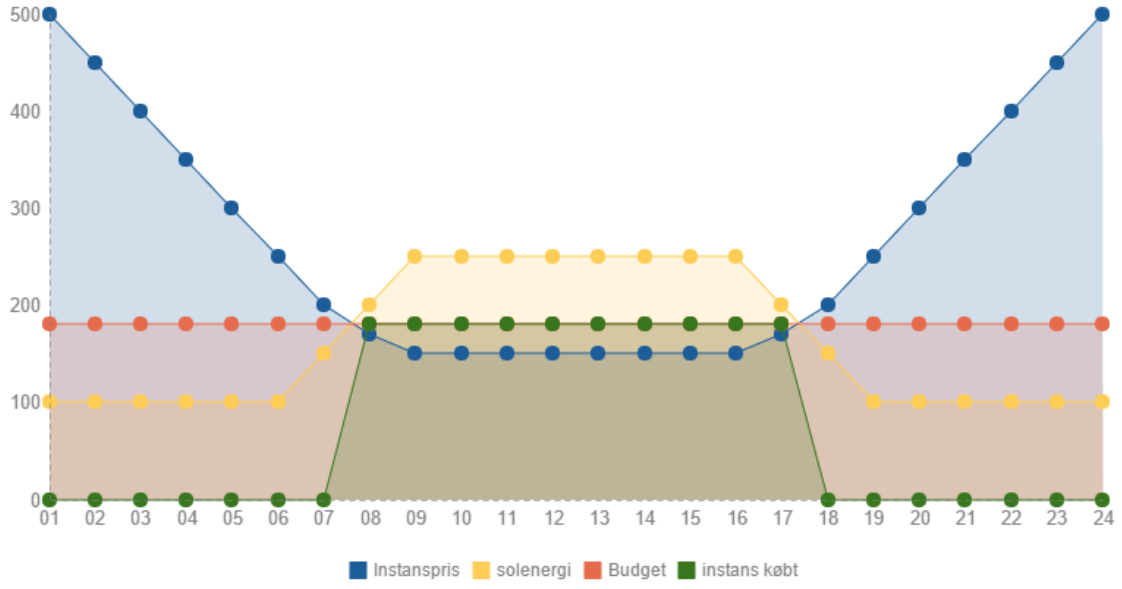
En anden prismodel der er opstået ud af det nye cloud paradigme er muligheden for at betale i forhold til en udbud og efterspørgselsmodel. Dette fungerer i praksis ved at kunden vælger en maksimal pris, der vil betales for en bestemt ressource. Så længe udbuddet er højt er prisen lav og man kan bruge sin ressource. Som udbuddet falder og prisen bliver højere, vil ens ressource blive frigjort så snart prisen overstiger den maksimalt tilladte pris.[7]

Dette er en fordel for både kunder og udbyder, da kunder kan få eksekveret noget af deres arbejde, på tidspunkter hvor ressourcer er billigt. Samtidig kan udbyderene maksimere deres Work load, så ressourcer bliver mere jævnligt fordelt. Dette kaldes hos AWS for ”spot requests”. Dette betyder blot at ens instans vil køre så længe ens bud overstiger den nuværende markedspris, sat af AWS.

Denne model hænger tæt sammen med ønsket om formindsket CO^2 udslip og generelle ønske om at være effektiv. Med kæmpe datacentre i hele verden, er strøm en af de største udgifter i vedligeholdelsen af dem. Derfor kan der spares mange penge ved at køre mest muligt arbejde, mens strømpriserne er lave. Dette afhænger selvfølgelig af, hvor i verden

man er. Hvis strømkilden er solceller kan det være en fordel at købe mest muligt strøm i perioder hvor strøm er billigt. [8]

Herunder vises en figur der illustrere en tænkt situation, hvor energiprisen har betydning for kunders besparelser ved at købe instanser hvor energi er billigt:



Figur 8: Instanspriser efter udbud og efterspørgsel

Linjen markeret ”budget” er, hvad kunden ønsker at betale for en given instans. Hvis prisen for en instans overstiger budgettet, bliver den ikke oprettet. I ovenstående model kan udbyderne have gavn af at flere instanser bliver købt når energien er billig. Samtidigt kan kunder få udført deres arbejde til en fastsat maxpris.

Andre eksempler, hvor f.eks. udbud og efterspørgsel alene bestemmer prisen på instanser er også mulige. Dette er forskelligt fra udbyder til udbyder og de vil sjældent opgive præcist hvad der bestemmer deres ”øjebliks-priser”.

Fastsat pris ud fra reserverede instanser

En tredje prismodel er muligheden for at reservere en bestemt instansopsætning i en fastsat periode. Herefter kommer ingen ekstra betalinger på denne/disse instanser. Denne model ligner meget den klassiske model for leje af servere. Modellen kan have sine fordele, da man på denne måde kan være sikker på hvad man betaler og derfor nemt kan regne med

forbrugsprisen for en lejede ressourcer. Denne prismodel er typisk meget billigere pr. ressource end de andre modeller, men kræver også man binder sig til et bestemt forbrug.

Aftalevilkår for services

En af de vigtigste områder som enhver virksomhed må tage stilling til når man skal vælge en cloud udbyder, er de service level agreements (SLA eller aftalevilkår), der omhandler hver enkelt service man bruger. En SLA er en aftale der laves mellem brugeren af servicen og udbyderen. Aftalen omfatter en specificeret beskrivelse af hvad udbyderen forpligter sig til at sikre ift. til oppetid, kvalitet, support, service og alt andet forbundet til den givne service.
Da abstraktionsniveauet generelt set er højere ved at bruge cloud computing og cloud services, er man også nødt til at være ekstra påpasselig med de indgående aftaler. SLA'er er typisk dokumenter der er nedskrevet i formel form som gør dem nemme at læse.

Selvom dokumenterne kan være nemmer at overskue enkeltvis, er det anderledes svært at skabe sig et samlet overbliv på kryds af services og udbydere. Der findes endnu ikke en officiel standard for hvordan disse kontrakter er udformet, samt hvordan de nødvendige parametre er udregnet og angivet. Dette betyder at forskellige udbydere kan love samme oppetider, men udregner definitionen forskelligt. Et andet element der gør disse aftaler indviklede for især mindre virksomheder er, at det i mange tilfælde (Heriblandt AWS) er op til kunden at bevise, at udbyderen ikke har opfyldt deres aftale. Hvad en eventuel konsekvens skal være af en brudt aftale, er som regel også angivet i en SLA. [9]

Nedenstående figur, er et udklip af AWS EC2 (Amazon Elastic Cloud Computing) SLA:

Monthly Uptime Percentage	Service Credit Percentage
Less than 99.95% but equal to or greater than 99.0%	10%
Less than 99.0%	30%

Figur 9: EC2 SLA vedrørende oppetid

Af ovenstående kan udledes, at AWS forpligter sig til en oppetid på minimum 99,95 %, hvilket er langt bedre end hvad de fleste andre udbydere kan tilbyde. Hvis aftalen ikke overholdes, har kunden krav på en refundering svarende til en procentdel af prisen.

Herunder ses et udklip af AWS Route53 (AWS Domain Name System service) SLA:

Duration Amazon Route 53 was not 100% Available	Service Credit
5 - 30 minutes	1 day Service Credit
31 minutes - 4 hours	7 days Service Credit
More than 4 hours	30 days Service Credit

Figur 10: Route53 SLA vedrørende oppetid

Herover ses at AWS Route 53 forpligter sig til en oppetid på 100%. Hvis denne forpligtelse ikke kan holdes, resulterer det i en refundering i form af kredit til samme service.

Herunder ses et udklip af AWS RDS (AWS database service) SLA:

Monthly Uptime Percentage	Service Credit Percentage
Less than 99.95% but equal to or greater than 99.0%	10%
Less than 99.0%	25%

Figur 11: RDS SLA vedrørende oppetid

I forhold til database hostingen, forpligter AWS sig til samme oppetid som for deres EC2 service, dog med en lidt mindre tilbagebetalingsprocent i tilfælde af brud på aftalen.

Eksempel 2:

Betragt et simpel system som består af følgende services som er hostet ved en cloud udbyder.

1. En routing service til at dirigere trafik (AWS: Route 53)
2. Et website der kører på en virtuel server (AWS: EC2)
3. En tilhørende database med data til websitet (AWS: RDS)

Hvis dette system indeholder kritiske data, eller er dybt afhængig af oppetider og stabilitet, bliver det hurtigt vigtigt at holde styr på aftalerne med udbyderen. I et sådan setup hos AWS, med brug af kun tre services, kan oppetider blive svære at sammenholde med de indgåede aftaler.

Da de tre forskellige services har forskellige aftalevilkår, men i høj grad er afhængige af hinanden, er det vigtigt at være klar over hvordan den ”samlede” SLA ser ud. Hvis et firma driver en kritisk service som er afhængig af, at alle tre services kører på samme tid, så må et nedbrud af bare én af dem, resultere i at hele system er nede. Dette er dog kun set fra virksomhedens side, da de tre services ikke har en samlet officiel aftale for vilkår, men tre separate aftaler. Dette betyder at cloud udbyderen kan have holdt alle tre aftaler, selvom det samlet system har været nede i længere tid end nogle af aftalevilkårene lover, for hver service. Dermed har udbyderen ikke brudt nogle aftaler, mens kunden oplever et nedbrud større end der er aftalt i nogle af de enkelte aftaler. I værste konsekvens kan det blive yderst besværligt, at finde og dokumentere alle måleparametre for at bevise at et samlet system har været nede i længere tid end hvad aftalerne beskriver.

For at alle virksomheder skal føle sig trygge ved dette områder mangler der stadig noget standardisering af disse aftaler, på tværs af udbydere. På denne måde kan de forskellige SLA’er nemmere sammenlignes og lægge grundlag for valget af udbyder.

Automation af processer

En af de store forskelle på hosting af egne servere og databaser kontra cloud hosting, er muligheden for at automatisere en masse processer der er forbundet med IT infrastrukturen. Som tidligere nævnt kan man med cloud hosting nemt opsætte regler for autoskallering, som gør det smertefrit at skalere ens løsninger uden at skulle foretage sig noget manuelt arbejde. Samtidigt gør udbyderene det muligt at bruge deres værktøjer til deployment og monitorering. Disse værktøjer er ofte gratis at bruge, og gør det muligt at automatisere mange processer som før har været manuelle.

En anden del af automatiseringen, er den del som sker selvom det ikke er aktivt valgt af brugeren. Dette gælder bl.a. monitorering, automatisk backup, load balancing og meget mere.

Eftersom Amazon, Google og Microsoft er ressourcestærke aktører, giver de også

kunderne en utrolig mulighed for at interagere med de mange services igennem deres højtudviklede API ‘er. Dette er med til at gøre automatiseringsprocesserne endnu mere effektive.
Som eksempel vil følgende kode uploade en fil til en ”AWS S3 bucket” (et samlet drev for en konto):

Kode stykke 1:

```
var AWS = require('aws-sdk');

var s3 = new AWS.S3();

 s3.createBucket({Bucket: 'myBucket'}, function() {

  var params = {Bucket: 'myBucket', Key: 'myKey', Body: 'Hello!'};

  s3.putObject(params, function(err, data) {

      if (err)

          console.log(err)

      else       console.log("Successfully uploaded data to
myBucket/myKey");

   });

});

Kilde: http://aws.amazon.com/sdk-for-node-js/
```

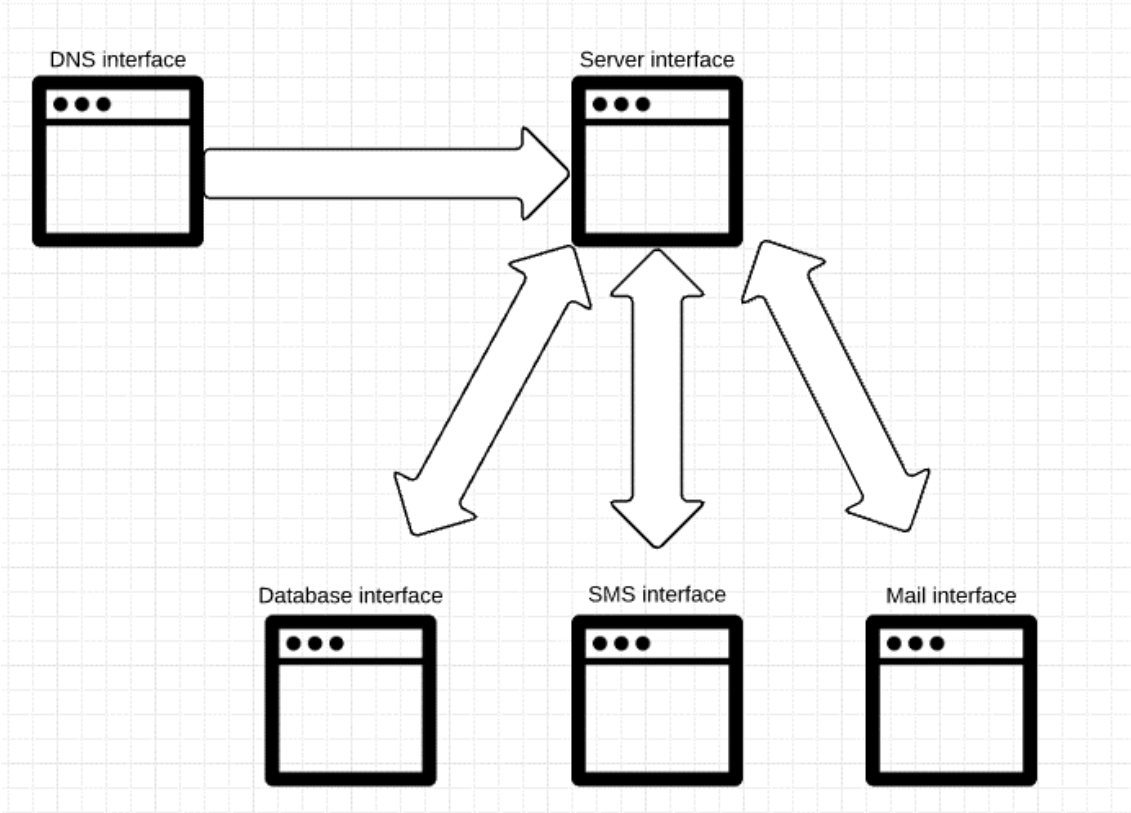
Koden ovenfor er blot et af mange eksempler, der er udgivet af AWS. Dette gør det utrolig nemt at implementere cloud services, som S3, ind i en allerede eksisterende kodebase.

Ovenstående er fremragende eksempel på hvordan man kan bruge det på forhånd angivne API, til at automatisere en proces der tager backup af et ønsket datasæt.

Samling af systemer:

Mange virksomheder har enten deres systermer fuldstændig opdelt eller slet ikke. Der kan være fordele ved begge arkitekturer, men som i eksemplerne nedenfor vil vise, kan der også være fordele ved at samle ens services så de i det mindste har samme interface.

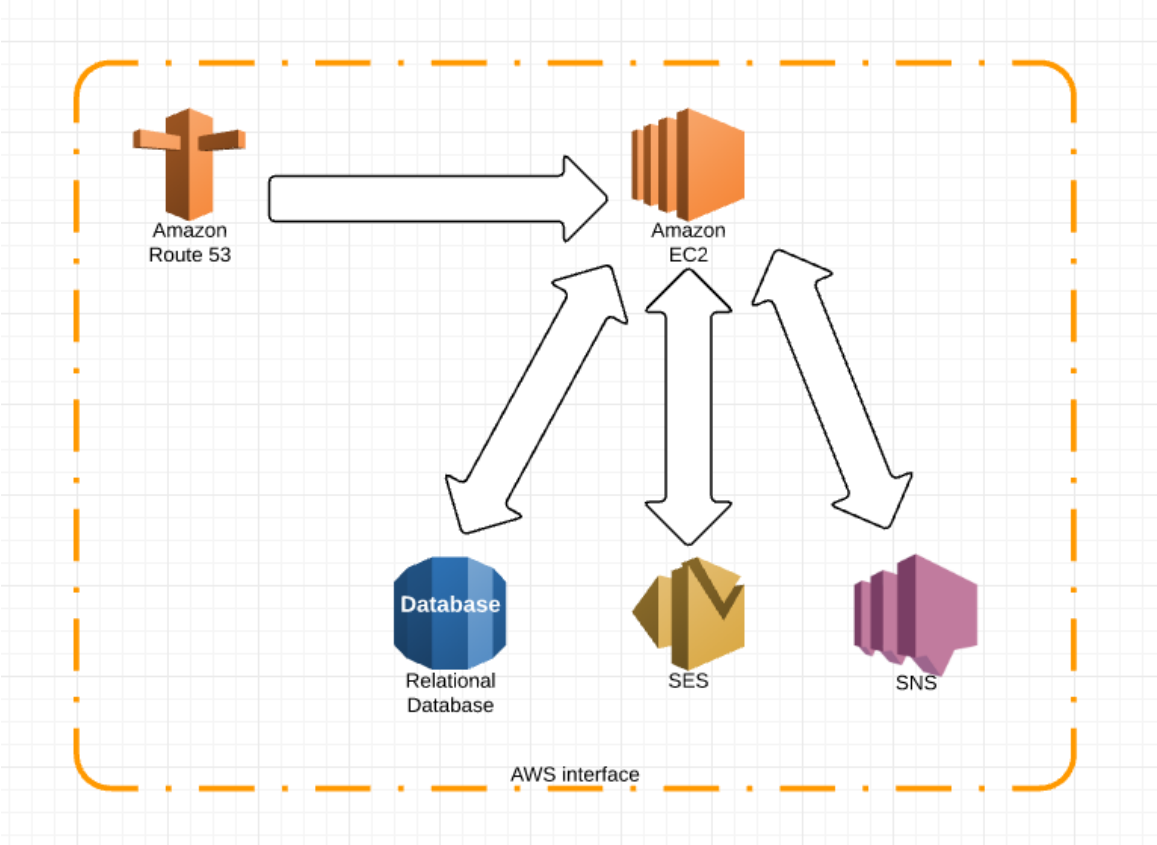
Se først system 1:



Figur 12: eksempel for forskellige interfaces

Her gør systemet brug af mange forskellige services, som hver især er en service, som har deres eget interface og regler for hvordan de skal bruges og administreres. Dette betyder at kunder selv skal administrere de forskellige interfaces, som kan være en stort opgave.

Med cloud udbydernes stigende antal services er det efterhånden muligt, at samle komplicerede systemer i samme interface.



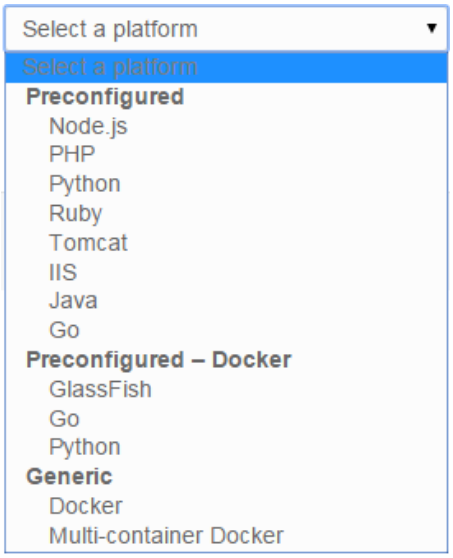
Figur 13: Eksempel på samlet interface

I dette system er alle elementer samlet i samme interface, hvilket giver en ensartethed i måden tingene administreres på. Her er både DNS opsætningen og sms udsendelsen en del af udbyderens interface, hvilke forsimpler administrationen voldsomt.

Hurtig udvikling

En af de største fordele ved at vælge cloud services er muligheden for udvikle og teste nye applikationer langt hurtigere end tidligere. Her kan specielt små virksomheder gøre gavn af nogle af de værktøjer cloud udbydere tilbyder. Da små virksomheder typisk har få ressourcer, vil det være kritisk at ny udvikling kan gøres uden at bruge for meget tid, på opsætning af infrastruktur. Udbydere som AWS tilbyder at tage sig af alt fra server opsætning, software opdateringer, autoskallering, monitorering, deployment og meget mere, uden kunder skal gøre andet end at uploade deres kode. Dette betyder at virksomhederne kan fokusere på selve koden, som vil gøre det langt nemmere for små virksomheder at afprøve nye ting, uden at skulle afsætte enorme ressource til det.

I AWS hedder en service der kan gøre det nemt for nye kunder at komme i gang med et hurtigt arbejdsflow, for "Amazon Elastic Beanstalk". Denne service tillader kunder at uploade kode til en virtuel server, af kundens ønskede størrelse. Herefter vælger kunden den ønskede opsætning af serveren og herefter klarer AWS resten. Herunder ses en liste over de pre-konfigurationer AWS i øjeblikket tilbyder igennem "Elastic Beanstalk".



Figur 14: Mulige server konfigurationer hos AWS

Dette betyder at hvis man programmerer i en af de ovenstående programmeringssprog, så har AWS allerede opsat servere der er klar til at hoste ens kode. Alt kunden skal gøre er at uploade koden og herefter vender AWS tilbage med en adresse, hvorpå man kan tilgå sin applikation. Hvis man kigger på ens konto, er der blevet oprettet en instans med sikkerhedsgrupper, autoskallering samt al opsætningen af netværk og andet infrastruktur nødvendigt. I firmaer er der til denne proces ikke behov for knowhow omkring

deployment, routing, DNS opsætning eller andre administrative aspekter, af at idriftsætte en applikation.

På samme made kan en database også hurtigt oprettes og man har herefter et helt lukket miljø, som ikke "forurener" ens eksisterende ressourcer. Med den korte tid mellem kodning og deployment bliver eventuelle fejl opfanget hurtigere og er derfor meget billigere. Derfor kan man hurtigt teste nye ideer og fejle, uden en masse ressourcer bliver spildt.

Praktisk gennemgang af hurtig udvikling i skyen

I det følgende vil jeg beskrive hvordan man kan opsætte en simpel Node.js applikation som er hostet hos Amazons Elastic Beanstalk service.

1. Det første det kræver er at oprette en konto hos Amazon Web Services.

En gratis konto er nok til at afprøve processen.

2. Lav en simpel applikation i et passende IDE

Her kan en eksisterende eller en template applikation nemt bruges.

3. Upload til AWS beanstalk

Dette kan enten gøres via AWS API eller via interfacet på deres hjemmeside

4. Brug evt. Route 53 til at benytte eget domæne.

Dette er ikke nødvendigt for at applikationen kører, men hvis man allerede har et domæne kan dette passende bruges sammen med Route53 for at kunne tilgå sin applikation fra ens eget domæne.

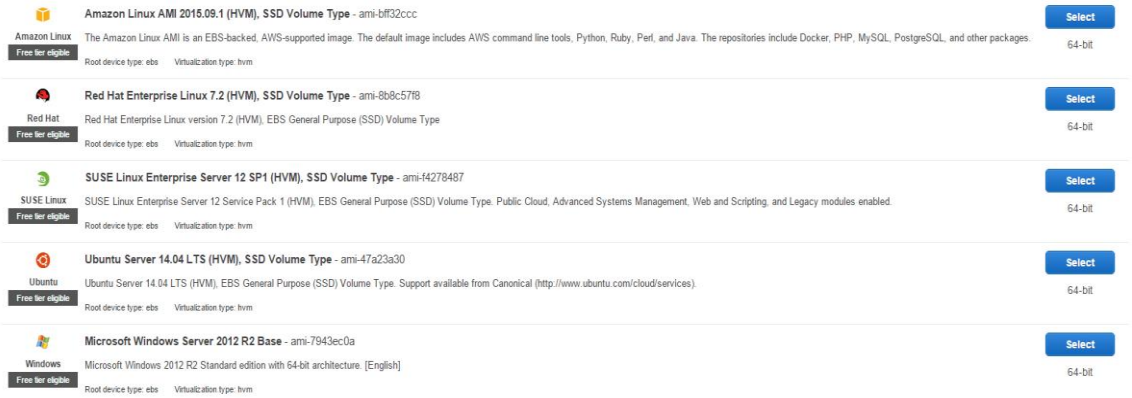
Som beskrevet ovenfor, er det med cloud computing nemt og hurtigt, at oprette nye projekter og applikationer, som skal fungere som prototyper. Alt afhængigt at typen af applikation man vil prøve, er det muligt at få en standard applikation op at køre på 30-60 minutter.

Forskellige platforme.

Tidligere har det været svært at prøve forskellige platforme eller programmeringssprog, hvis man i forvejen er bundet af en udbyder eller selv hoster sine ressourcer, og derfor kun har adgang til de licenser man har købt. Dette bliver lettere at gøre når man har en

dynamisk platform, som ikke nødvendigvis er bundet til nogle bestemte software eller hardware producenter. Hvis udbyderne er tilpas åbne, bliver det muligt at bruge open-source software, som kan gavne specielt mindre virksomheder, hvis de vil afprøve nye muligheder.

For de større firmaer, hvis infrastruktur kan gøre det svært for dem at afprøve nye teknologier, kan skyen være en god mulighed for, nemt at implementere nye eller gamle løsninger på en ny platform.
Dette kan bl.a. gøres ved hjælp af virtuelle servere (EC2), hvor en stor vifte af images er tilgængelige. Disse images, som er en specifik konfiguration af styresystemer, kan frit vælge fra en lang liste. Herunder ses en liste over nogle af de AMI 'er (Amazon Machine Image) der er tilgængelige for en gratis konto:



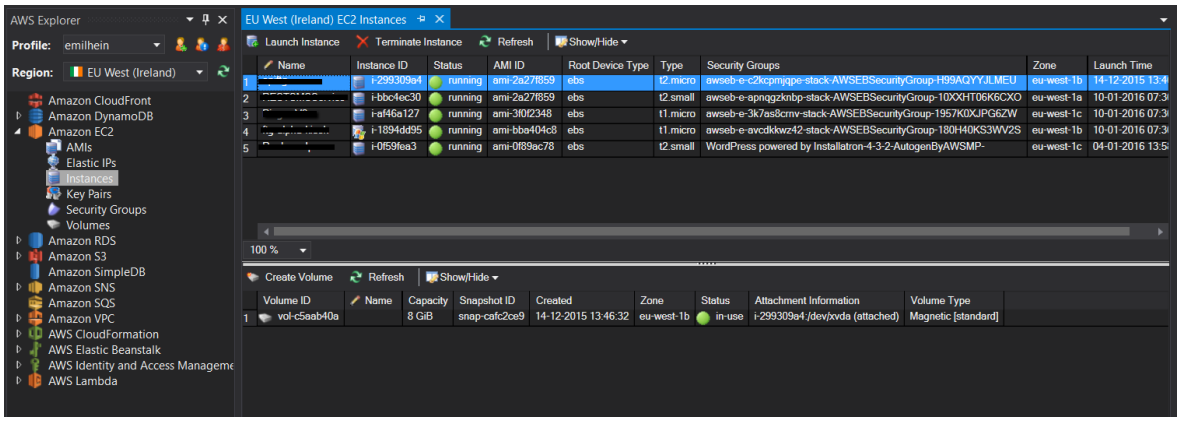
Figur 15: Gratis AMI's hos AWS

Derfor er styresystemer ikke længere en barriere for udviklerer. Hvis man ønsker en server med et bestemt OS, er det blot at vælge det ønskede AMI g oprette en instans.

Grundet den store kundebase store cloud udbydere har, og det fællesskab der er opbygget omkring dem, findes der en lang liste af skræddersyede AMI 'er. Disse kan frit bruges, dog på eget ansvar.

AWS har som en ekstra service udviklet en masse SDK's (Software Development Kit) til forskellige programmeringssprog og udviklingsprogrammer (bl.a. Visual studio og Eclipse). Dette gør det utrolig nemt at komme hurtigt i gang med det ønskede kodningssprog man ønsker. Jeg har gennem tests med SDK til Visual Studio, fundet det

værdigfuldt, at have alle informationer omkring ens AWS opsætning direkte ved hånden i ens IDE[2].



Figur 16: AWS SDK til Visual Studio

Som ovenstående viser kan det være det være utrolig tidsbesparende, at have alle ens ressourcer tilgængelige igennem et IDE, så man nemt kan få et overblik over hvilke ressourcer man har at arbejde med. Ved hjælp af dette SDK er muligt at lave deployments til AWS direkte igennem Visual Studio.

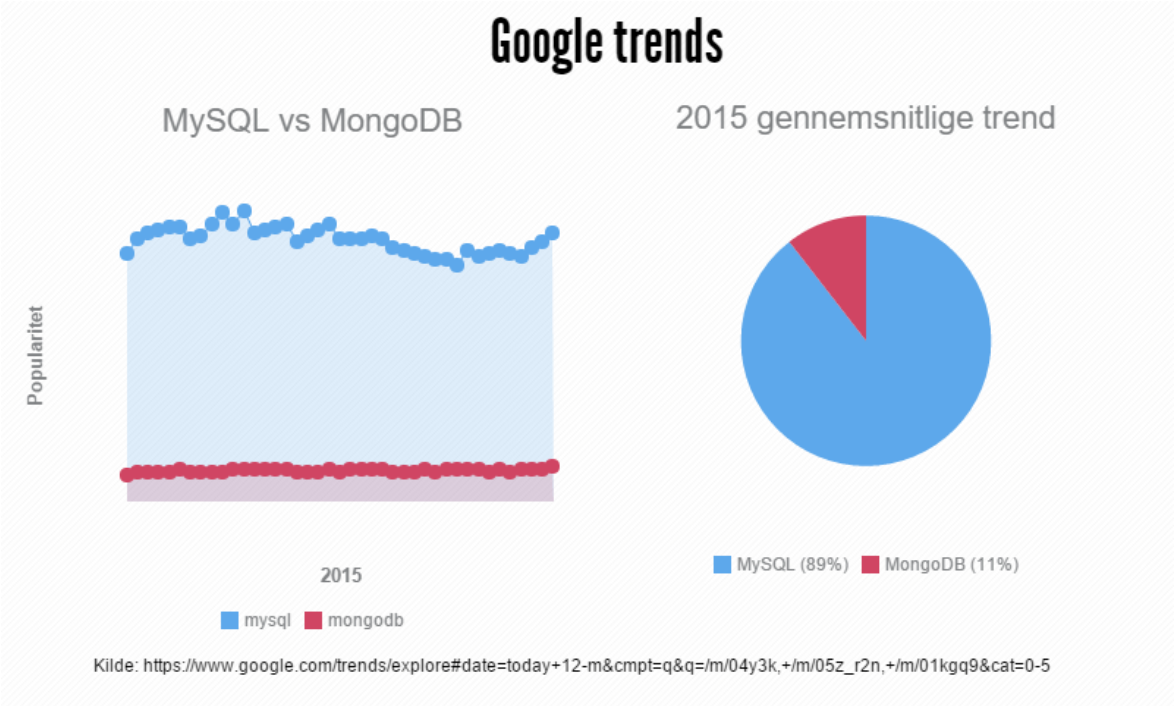
[2] Integrated development environment

Praktisk migrering til skyen

Migrering af database til skyen:

At migrere til skyen er en samlet betegnelse for processen der flytter ens databaser, applikationer eller andre ressourcer til en af web hosting udbyderne. Denne proces er dybt afhængig af typen og størrelsen af de applikationer man skal have flyttet. Hvis et firma skal have migreret en database til skyen skal man tænke på flere forskellige ting. Først skal der undersøges hvilke databaser de forskellige udbydere understøtter. Efterhånden understøtter de fleste, mange af de mest almindelige typer af databaser, så som: MySQL, MSSQL, mongodb, postgreSQL og flere. Hvis man har specialiserede krav til sin database, kan det være meget specifikke grunde til at vælge den ene udbyder frem for den anden.
Selve migreringen kommer an på mange ting, men nogle af de følgende elementer er nogle af de mest kritiske og tidskrævende.

1. **Størrelsen af databasen betyder noget.** Størrelsen på en database skal teoretisk set kun forøge tiden det vil tage at uploade al data, og ikke tilføje kompleksitet eller på andre måder gøre det svære at migrere. Dette er dog sjældent tilfældet. Større databaser har tendens til at have mere avancerede schemas. Senere vil jeg forklare hvorfor dette bliver et problem for migrering af større databaser.
2. **Den specifikke type af database betyder noget.** Dette element er nok den mest fejlbehæftede proces når man skal migrere en database til skyen. Generelt set bruges der af de fleste firmaer to typer af databaser. Disse bliver i daglig tale refereret til som være enten en SQL eller NoSQL (relationelle og ikke-relationelle). Open source databaserne MySQL, SQLite og Mongodb er blandt de mest populære, hvor mongodb er den eneste ikke-relationelle. På verdensplan er de relationelle database langt mere brugt, og på næste side ses en graf over en google trend søgning som viser tendensen af google søgninger på de forskellige typer af databaser i de 2015.



Figur 17 MySQL vs MongoDB trend

Den specifikke forskel mellem de forskellige typer af databaser er uden for afgræsningen af denne rapport, og senere eksempler vil derfor primært benytte sig af MSSQL og postgreSQL databaser (begge relationelle).

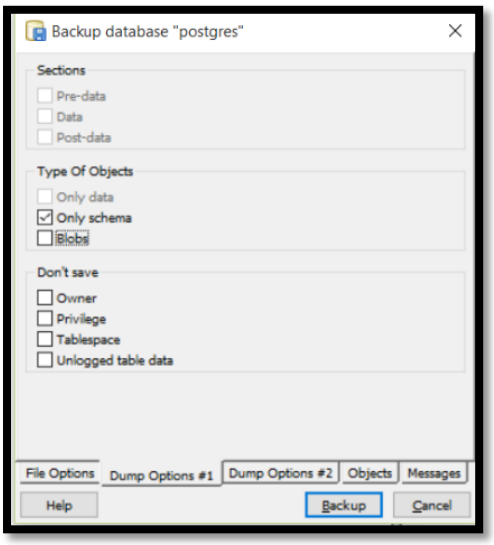
Lad os antage, at et firma vil migrere en MySQL database på 100 Megabyte til skyen. Da dette er relativt lille database, burde migreringen kunne ske forholdsvist let. For argumentets skyld lad os sige at udbyderen ikke understøtter MySQL (langt de fleste gør). Hvad skal firmaet så stille op. Skal man skifte database type eller skifte udbyder? Umiddelbart løser skiftet til en anden udbyder det nuværende problem, men det løser ikke det mere generelle problem med kompatibilitet på tværs af platforme. Problem er at forskellige typer af databaser understøtter forskellige schemas og operationer. Den mest almindelig form for migrering af databaser er derfor fra og til databaser af samme type. Dette kunne være fra en lokal MySQL instans til en MySQL instans hostet hos udbyderen.

Migreringen mellem to databasser af same type kan ske på mange forskellige måder. Jeg vil i følgende afsnit beskrive to forskellige metoder til at migrere en PostgreSQL. Databasen der skal migreres er allerede hostet hos AWS. Destinationen for migreringen er en anden databaseinstans også hostet på AWS. Grunden til at start databasen er hostet hos AWS, og ikke er en lokal instans, er for at sikre den nemmeste migrering til eksemplet. Om databasen er hostet hos AWS eller lokalt er for testen ligegyldigt og fremgangsmåden vil være præcis den samme.

Den første og mest simple metode, gør brug af de indbyggede backup og genskabelses

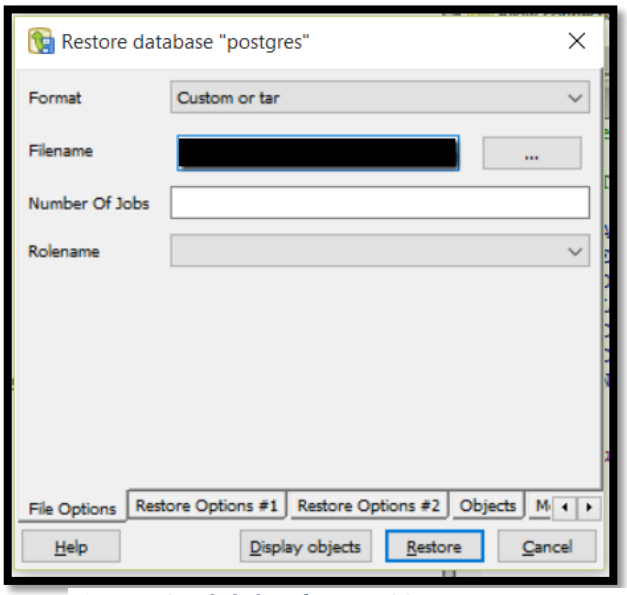
funktioner, der medfølger i alle slags databaser eller administrationsværktøjer. Til en Postgres database, kan værktøjet pgAdmin bl.a. bruges til dette.

Databasen der testes er en Postgres 9.4.4, og er blot en lille testdatabase til at illustrere metoden.



Figur 18: pgAdmin backup værktøj

PgAdmin lader brugeren bestemme hvilket format der bruges til backup, samt hvilke dele af database man ønsker at gemme. For mindre databaser er der ingen problemer i, at tage backup af både schema og data i databasen. Når backuppen er foretaget med succes, kan PgAdmin bruges til at forbinde til den ønskede destinations server. Herefter kan en ny database oprettes, som en gendannelse af den tidligere backup.



Figur 19 Genskabelse af PostgreSQL DB

Hvis alt går godt, er der nu oprettet en kopi af den gamle database på destinations serveren hostet i skyen. Den/de applikationer der bruger databasen, kan herefter blot ændre

forbindelsesstrengen, så den nye database bliver brugt. Dette beskriver dog kun selve migreringsprocessen, og der er mange andre aspekter der skal tages i betragtning hvis man vil lave en fuld migration til skyen. Der skal bl.a. tænkes på sikkerhed og skalerbarhed.

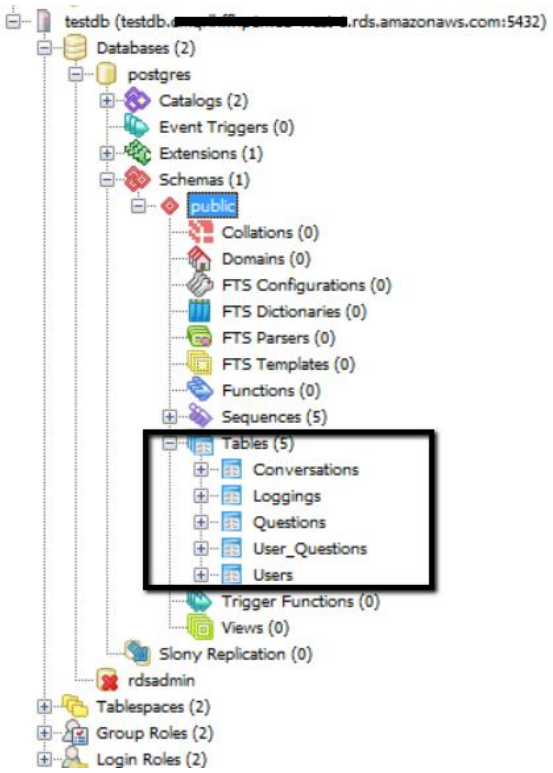
Den anden metode til migrering af SQL databaser udføres ved at lave en scripting af databasen. Da alle tabeller i en database er blevet konstrueret ud fra en række kommandoer, er det muligt at genskabe disse kommandoer (at scripte) og herved bruge disse kommandoer til at skabe en ny database i skyen. Denne metode bruges af flere forskellige værktøjer til at eksportere databaser.
Ved ovenstående eksempel kommer scriptingen af en enkelt tabel til at se ud som følgende:

Kode stykke 2:

```
CREATE TABLE "Conversations"
(
  id serial NOT NULL,
  "ConversationDescription" character varying(255),
  "createdAt" timestamp with time zone NOT NULL,
  "updatedAt" timestamp with time zone NOT NULL,
  CONSTRAINT "Conversations_pkey" PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE "Conversations"
  OWNER TO emilhein;
```

Hvis man samler alle tabellerne i databasen til en samlet fil, ender man med koden til at oprette databasen ethvert andet sted. Som tidligere nævnt kan der godt opstå problemer hvis man scripter fra én version af en database til en anden. Derfor er denne slags af migrering heller ikke altid mulig. Denne metode giver dog langt større frihed idet man kan tilrette den samlede script så det understøttes af destinationsdatabasen. Ved ovenstående metode, lykkedes det uden problemer at migrere en mindre PostgreSQL database, der

bruges til et sms system. Databasen, vil senere blive brugt i testafsnittet. Efter den succesfulde migrering kan databasen ses i pgAdmin.



Figur 20: migrerede PostgreSQL database

Når en database er overført til skyen er det vigtigt at være opmærksom på sikkerheden omkring dette. Hvis en database tidligere har været hostet lokal eller på dedikerede servere har adgangen til databasen højst sandsynligt været beskyttet af en firewall. Det er derfor vigtigt at genoprette samme sikkerhed på den virtuelle server, som der tidligere har været.

Dette er kun selve migrationen. Derudover er der mange punkter man også er nødt til at være opmærksom på. For at teste en om en migrering er lykkedes og ens data er ikke er blevet ændret i processen er der flere ting man gøre. Tests, som dem jeg foretager senere, er en af flere muligheder for at teste om ens database er migreret korrekt.

Migrering af applikationer til skyen:

General migrering eller flytning af applikationer og systemer kan være et komplekst emne, alene grundet de mange forskellige platforme og setups. Derfor vil jeg med udgangspunkt i Amazon Web Services påvise hvordan en migreringen af et specifikt system kunne foregå. Som udgangspunkt er AWS EC2 (Elastic Cloud Computing) et godt sted at starte, for hosting af applikationer og systemer af forskellig art. EC2 tilbyder traditionel hosting af en virtuel server man selv har direkte adgang til. Dette betyder at man selv kan bestemme hvilke programmer der skal installeres, hvilket operativ system man vil køre på sin server, og i sidste ende hvilke applikationer man vil hoste. Der findes også andre specialiserede hosting services som tilbyder nogle lidt andre fordele og priser. Disse bruges typisk til mere specialiserede hosting formål og det er derfor også Elastic Cloud Computing services jeg vil foretage migrering og tests på.

EC2 instanserne hos AWS tilbyder en stor vifte af mulig setups. Disse dækker alt fra helt små instanser, der kan bruges til små projekter til store servere der er optimale til store og tunge udregninger. Der findes i skrivende stund 39 forskellige opsætninger af hardware, som man frit kan vælge imellem. Til alle formål i denne opgave har jeg valgt de mindre versioner, som er tilgængelige for en gratis konto, så ingen af resultaterne afhænger af store og dyre instanser.

I de fleste tilfælde vil EC2 instanser være den bedste mulighed for leje af virtuelle servere. Hvis serveren sættes op som ens eksisterende server, kan migreringen af mange applikationer ske forholdsvist smertefrit, hvis der ikke skal laves ændringer før en migrering overhoved er mulig.

Før, under og efter en migrering

Følgende giver et indblik i nogle af elementer der skal overvejes i forhold til, om migrering til skyen er den rigtige ide for den enkelte virksomhed og applikation. Nogle nichevirksomheder vil have krav som kræver meget specifikt hardware og/eller software. Da cloud udbydere i sagens natur er nødt til at spænde bredt, for at opfylde så mange behov som muligt, kan det være svært at få opfyldt meget specifikke behov. Hvis en analyse viser, at den type service man gerne vil have flyttet til skyen, er fuldt kompatibel med hvad udbyderen understøtter, kan næste fase begyndes.

Alt afhængig af hvilken type service man ønsker at flytte, skal man være klar over hvor sensitive ens data er, samt hvor længe man er villig til at være offline hvis en fejl skulle ske.
Selvom forberedelse kan gøre nedetid under en migration minimal, kan det i større systemer være svært helt at undgå. Derfor skal en endnu en analyse foretages for at sikre at nedetiden for f.eks. et website er minimal ved skiftet til hosting i skyen.

Det kan næsten ikke understreges nok hvor vigtigt det er at teste. En tilgang til opgaven er følgende:

1. Nedskriv en liste med de vigtigste funktioner i den applikation/service/database, etc.
2. Implementer denne liste som enten load test, Selenium test eller andre former for formelle automatiserede test.
3. Test først implementerede tests mod det originale system (ikke i skyen).
4. Efter migrering, kør da de samme tests imod systemet der er hostet i skyen.

En langsigtet overvejelse der kan foretages er muligheden for cloud-til-cloud migration. Dette kan virke underligt at tænke over, når man lige har valgt udbyder, men i realiteten er det et vigtigt punkt at tænke på. I henhold til afsnittet omkring lovgivning på området omkring persondata, kan der opstå incitament til at migrere til en anden udbyder. Dette er dog meget afhængig af virksomheden og typer af data der arbejdes med. Det er sjældent en god ide at gøre sig selv eller sit firma for afhængig af en bestemt udbyder og dets services. Som hovedregel kan det derfor anbefales ikke at skræddersy sin applikation alt for meget, til en bestemt udbyder. Hvis et skifte senere bliver en realitet, er det fordelagtigt at kunne

migrere til en anden cloud udbyder, uden større besvær.

Jeg har fundet følgende aspekter vigtige i hele processen med at migrere et fungerende system til skyen.

Før migreringen
At sikre muligheden for cloud-to-cloud migration bliver ofte glemt, men alligevel et vigtigt aspekt. Det kan være farligt at låse sig fast på en bestemt udbyder, udelukkende på grund af specifikke funktioner. Dette kan gøre det svært at skifte udbydere, hvis der senere skulle opstå komplikationer.

Under analysen af migreringen skal man holde sig for øje hvor svær migreringen bliver. Dette kan have store omkostninger for visse applikationer. *Muhammad Ali Babar* [10] beskriver bl.a. hvordan deres applikation skulle ændres drastisk før den kunne migreres til AWS. Dette krævede flere skridt og kunne måske få andre og lignende projekter til at genoverveje migrering til skyen.

Under migreringen
Hvis man planlægger det ordenligt, burde selve migreringsskridtet ikke være kritisk. Det er ofte ikke anbefalet at man laver en migrering direkte på en applikation i produktion. Man vil typisk migrere først og herefter ”udskifte” applikationen. På det tidspunkt hvor den endelige flytning sker, er det yderst vigtigt at man monitorere systemet tæt, så man kan skifte tilbage, hvis større fejl skulle opstå. Ved migrering af f.eks. kundedatabaser, skal man være klar over muligheden for, at man mister de data der er blevet oprettet mellem selve migreringen og det tidspunkt man vælger at skifte database på. For at sikre at de to databaser (originale og cloud hostet) er synkroniseret før skiftet findes der avancerede metoder i f.eks. AWS, der kan hjælpe. En anden og mere primitiv måde at løse problemet på, er at lukke siden midlertidigt, så man sikre sig at ingen nye data vil blive gemt i databasen. Det er ikke alle systemer der har denne mulighed, og der skal derfor tænkes grundigt over hvordan man holder de to databaser synkroniserede.

Efter migreringen

Eftersom migrering af applikationer og systemer generelt er afhængige af mange ting, er det vigtigt at have nogle generelle områder hvorpå man kan teste om ens specifikke applikation er migreret korrekt. Til tests af hjemmesider er der flere forskellige værktøjer der kan bruges til, at teste om migreringen er sket uden fejl.

For at teste om en hjemmeside er migreret korrekt, kan værktøjet JMeter bruges til at stressteste, for at se om den ønskede ydeevne er til stede. Denne slags load tests, som bliver nævnt i detaljer i testafsnittet, vil typisk kun giver resultater om svar og load tider. Disse tests er ikke optimale til at teste om der er sket fejl ved selve migreringen.
Til at teste at alt virker som det skal, er avancerede queries mod databasen, samt udførlige Selenium tests mere anvendelige. Database queries skal tjekke om alt internt i databasen virker. Jeg har erfaret at udførlige Selenium tests er utrolig brugbare til at teste både en

hjemmesides UI, men samtidig også den tilhørende database.
Med udførlige tests, kan alle afkroge testes for at se om alle elementer på en given side er synlige. Samtidigt tester en Selenium test også databasen, ved at agere en normal bruger og på den måde tvinger systemet til at bruge databasen, som den vil gøre under normale omstændigheder.

Hvis man har et tilstrækkelig kompliceret system, kan mere formelle og omfattende metoder bruges. [11] nævner bl.a. en 5-skridts model som består af følgende 5 faser:

1. Undersøg om en migrering overhoved er økonomisk og teknisk mulig.
2. Planlægning og analyse af hvad migreringen kræver.
3. Selve migreringen.
4. Test og deployment af systemet.
5. Overvågning og vedligeholdelse af systemet.

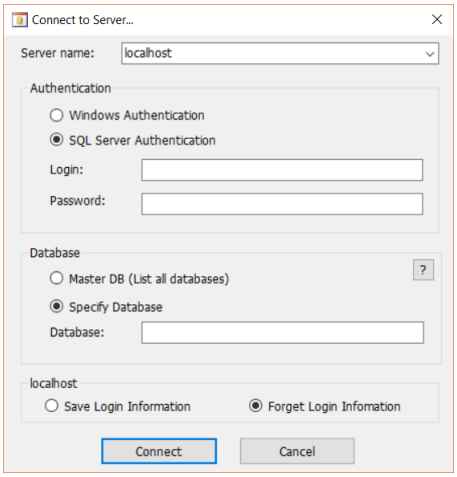
Test

Værktøjer

Til test af forskellene mellem forskellige setups i både AWS og dedikerede servere, skal der bruges et sæt værktøjer der gør det muligt at teste både databaser og applikationer. Derfor kan det rigtige sæt værktøjer sikre at man får testet alle de nødvendige elementer involveret i migrering til skyen. Herunder beskrives en række værktøjer jeg har brugt til mine egne tests. Specielt JMeter har været et uundværligt værktøj.

SQL Database Migration Wizard

Dette værktøj er udviklet af Microsoft, og er udviklet til brug med deres cloud hosting (Azure), for at hjælpe kunder med at migrere deres databaser fra én lokation til en anden. Værktøjet gør brug af metoden tidligere nævnt, ved at generere et script fra en source database, og herefter køre det mod en tom database eller eksisterende database efter eget valg. Jeg vil bl.a. bruge dette værktøj til at migrere en database til AWS.



Figur 21: Migration Wizard from Microsoft

Dette værktøj virker kun til Windows og SQL database. Det er med værktøjet lykkedes at migrere hele Freetrailer' danske testdatabase (DK_TEST) til AWS. Databasen er dog så stor at det kræver flere tests, for 100% at kunne verificere det som en succes.

PgAdmin

Et administrationsværktøj til f.eks. PostgreSQL databaser. Dette værktøj minder om SQL management studio til en MSSQL database. Værktøjet kan bruges til at lave manuelle operationer mod databasen, samt at se og redigere i basale opsætninger. Det er et open-

source værktøj, hvilket er en stor fordel for især mindre virksomheder, da der ikke skal bruges penge på dyre licenser til specialiserede værktøjer.

JMeter

JMeter er et open source program udviklet af apache. Programmet er skrevet i Java og vil danne rammen for de fleste tests. JMeter tillader brugeren at opsætte en lang række forskellige tests og analyser. Programmet er opbygget modulært, hvilket vil sige at mulighederne for udvidelser er mange. Man kan selv kode en udvidelse, eller man kan bruge nogle allerede testede moduler. Programmet tillader både at foretage http forespørgsler, samt kald til forskellige typer af database. Det er bl.a. disse features der senere vil blive brugt til load test og svartids-test på både databaser samt hele applikationer. Et andet vigtigt element er muligheden for at oprette forskellige tråde, der udfører de angivne instrukser. Dette skal simulere et mere realistisk mønster i antallet af forespørgsler oprettet af mange forskellige brugere.

Alle tests kan læses i bilag A1:

Selenium

Selenium er et open source værktøj som oprindeligt er udviklet af ”ThoughtWorks”, men er senere blevet opdateret af mange forskellige udviklere. Selenium er udviklet som et værktøj til at teste web applikationer. Mere specifikt gør Selenium det muligt at oprette funktionelle tests som kan køres automatisk. I mange virksomheder har funktionelle tests været udført manuelt, men med Selenium er det nemt at opsætte disse test til at køre automatisk. [12]
Det effektive resultat af Selenium tests er en simulering af hvad en normal bruger ville gøre. Testen består typisk af en sekvens af handling og eventuelle inputs en bruger ville udføre. En test kunne se ud som følgende.

Eksempel 3 - Selenium sekvens

1. Hent webadressen www.google.dk
2. Klik på søgefelt med id=”lst-ib”
3. Skriv ”cloud computing” i søgefeltet
4. Tryk på knap med id=”lsb”
5. Vent på siden loader
6. Tryk på øverste resultat
7. Tjek om sidens title=” Cloud computing - Wikipedia, the free encyclopedia”

Denne test vil foretage en google søgning af sætningen “cloud computing” og herefter trykke på øverste resultat. Der tjekkes så om første resultat er en wikipedia side. Hvis Ja, kan denne test siges at være bestået.

Med denne slags test, kan man optimalt set skrive ens tests før man udvikler en

webapplikation. Dette kaldes testdriven-Development, og er noget som kan mindske udviklingstiden og gøre applikationens specifikationer tydeligere for udvikleren(e).

For at kunne køre mange af disse test og måle på load hastighederne vil jeg bruge et Selenium plugin til JMeter, så alle tests kører igennem JMeter.

Opsætningen af Selenium i JMeter, kan læses i detaljer her:
https://blazemeter.com/blog/jmeter-webdriver-sampler

Hvis man går i dybden med disse tests kan de i sammenhæng med værktøjer som JMeter være utrolig nyttige. Der findes et hav af udvidelser som gør det muligt at lave yderst avancerede tests. Da al kildekoden er open-source har firmaer også mulighed for at skrive plugins til programmet selv. Firmaer som *Digital Focus* [12] har selv skrevet plugins og fundet det utrolig nemt at eksekvere testdriven Development, hvor man hurtigt kan skrive en test og derefter gå i gang med at kode ud fra denne test. Da Selenium tests skrives på en enkelt måde kan testere eller analytikere hurtigt skrive disse tests, uden nødvendigvis at kende til implementationen.

Test af autoskalering af applikationer

Som tidligere beskrevet er migrering af en database blot et udtryk for at flytte en database med data fra et sted til et andet. Fordelene ved at flytte sine databaser til skyen kan være mange, men et af de helt store er de dynamiske muligheder skyen tilbyder.

Først vil jeg se på muligheden for auto skallering. Dette er en funktion der gør det muligt at definere regler, som kontrollerer databasens, applikationen eller serverens ydeevne. Disse regler kan inkludere mange forskellige ting, men vil typisk sættes op omkring load på databasen, antal forbindelse eller andre parametre.
To af de vigtigste aspekter ved auto skallering er prisbesparelse og ydeevne i forhold til tidligere statiske serverstørrelser.
Autoskallering kan i perioder uden meget trafik, lukke ned for instanser og herved spare firmaet for penge, som alligevel ikke bliver brugt til at forbedre kundernes oplevelse.
Den anden del af det er, at under perioder med meget trafik kan autoskallering oprette flere instanser og dermed forøge kapaciteten. Dette hjælper med at holde siden eller applikationen tilgængelig under stort pres. Tidligere har disse muligheder været umulige og man har været nødt til at købe en fast kapacitet. Som mange andre ting er efterspørgslen på de fleste sites og applikationer langt fra stabil, og det må forventes at se forskellige forbrugsmønstre og besøgstal inden for de fleste sektorer.

Test af database

Tests af databaser er et stort emne, og man kan afdække mange forskellige aspekter. I følgende vil jeg forsøge at beskrive og analysere nogle af fordelene ved at migrere sine databaser til skyen og benytte sig af nogle af fordelene.

Databasen der vil blive testet er lille og består af 5 tabeller (Se figur 21).
Databasen er en del af et sms system, som kan udsende sms spørgeskemaer baseret på folks svar. På sigt kan man forestille sig, at serveren der modtager og behandler alle de indkomne sms'er, kunne blive hårdt belastet. Derfor kan ydelsen fra databasen være kritisk for den samlede ydelse af systemet.

Testen af selve databasen vil blive udført ved at udføre nogle relative simple operationer, INSERT/READ/DELETE, mange gange. Disse operationer vil være typiske operationer i systemet og er derfor også repræsentative som en test af ydelsen fra databasen.

Til at udføre selve testen kan flere forskellige teknikker overvejes. Først kan selve applikationen bruges til at teste databasen, ved at genere forespørgsler til databasen. Fordelen ved denne type af test, er at forespørgslerne til databasen vil være identiske med dem produktionsmiljøet vil lave. Derfor vil testen give et naturligt billede af hvordan databasen vil regere på de faktiske forespørgsler i systemet.

Ulempen ved metoden er, at svartider fra databasen ofte er meget kortere end dem man har fra selve applikationen (såfremt det er simple databaseforespørgsler). Derfor vil testen højst sandsynligt kun teste ydelsen af applikationen og vil i høj grad afhænge af hvordan applikationen har implementeret selve database kaldene.
En test fra den ”rigtige” applikation vil derfor ikke kunne lave en troværdig test af ydelsen på en database, med mindre man i applikationen har enten meget tunge kald eller rigtig mange kald til databasen.

Test 1 - database performance og monitorering.

Denne test skal give et praktisk indblik i hvordan backup af databaser håndteres i AWS. Som standard gemmer AWS en fuld backup hver dag i 30 dage, kaldet et snapshot. Inkrementelle backup bliver også gemt, så man kan gendanne data ned til en præcision på en transaktion. Dette betyder at man kan gendanne data fra et meget præcist tidspunkt, hvor man ved en fejl er opstået eller man ønsker at gendanne til.

Der begyndes med en PostgreSQL database der allerede er migreret til AWS (se figur 21).

Testen består i at indsætte data over to perioder (se figur 23). I hver periode foretages indsættelse af brugere (se kodestykke 3).

Hver periode med indsættelse sker over en tidsperiode på 55 sekunder med 65 sekunders mellemrum.

Herunder vil jeg kort forklare og beskrive en simpel test af en migreret database til AWS og teste nogle af de features der følger med når databasen er hostet hos AWS.
Testen sker på en database, der kører med PostgreSQL version 9.4 og er hostet på Amazons RDS service.

Hele testresultatet kan ses i Bilag A 3, men nedenfor vises nogle nøglefigurer.

Testen er udført i JMeter med følgende parametre:

Antal brugere: max 6 på ethvert givent tidspunkt.

Ramp-up-time: 10 sekunder.

Hver bruger udfører en forespørgsler af typen:
Kode stykke 3:

```
insert into public."Users"
values
(DEFAULT,floor(random()*(99999999-10000000)+10000000),now(),now());
```

Dette betyder altså at JMeter simulere 6 samtidige brugere, der hver især opretter 1 bruger i databasen.

Figur 22: Forventede antal parallelle brugere.

Ovenfor ses et estimat af, hvordan den relativt simple test kommer til at foregå. Ideen er herefter at lave en manuel aflæsning af antallet af rækker indsat i tabellen. Herefter vil en backup blive taget og endnu en manuel aflæsning vil vise, hvordan backup af en database til et bestemt tidspunkt hostet hos AWS fungerer.

Databasen indeholder til at starte med 2855 rækker med data, hvilket bliver vigtigt senere i testen, da dette er den eneste tabel vi i denne test berører. I midten af testen (den røde pil

på figur 24) noteres det midlertidige resultat.

Figur 23: Resultat af testen. Viser antallet af parallelle brugere

Ovenfor vises det faktiske resultat af testen. Grafen viser antallet af aktive tråde over tid.

Nedenfor ses hvor mange rækker der er blevet indsat i den første periode af testen. Billedet er derfor taget ca. 1 minut og 55 sekunder efter testens start (den røde pil).

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
insert user	646	80	45	420	39,32	0,00%	12,9/sec	0,11	9,0
Select all	644	242	111	854	114,87	0,00%	12,9/sec	2795,25	221840,1
TOTAL	1290	161	45	854	117,92	0,00%	25,8/sec	2789,22	110752,6

Figur 24: halvvejs i testen, med rapport over indsatte rækker

Da antallet af rækker i tabellen var 2855 til at starte med, er der altså nu 3501 rækker (2855+646), før anden del af testen går i gang. Efter anden del af testen ser resultatet således ud.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
insert user	1233	81	45	420	32,95	0,00%	5,5/sec	0,05	9,0
Select all	1230	258	111	1297	137,58	0,00%	5,5/sec	1294,18	242351,1
TOTAL	2463	169	45	1297	133,61	0,00%	10,9/sec	1293,63	121032,5

Figur 25: det endelige resultat af testen.

Der er nu indsat 1233 rækker og det totale tal er derfor 4088.

Nedenfor ses et udsnit af den sidste indsatte række i første del af testen, samt den første indsatte række i 2. del. Der er 65 sekunder imellem disse, og det er i dette tidsrum, der skal genskabes førend det kan kaldes en succes.

3501	140423	92036958	2015-[redacted] 21:13:40.037485+00	2015[redacted]21:13:40.037485+00
3502	140424	41434426	2015-[redacted] 21:15:45.106971+00	2015[redacted]21:15:45.106971+00

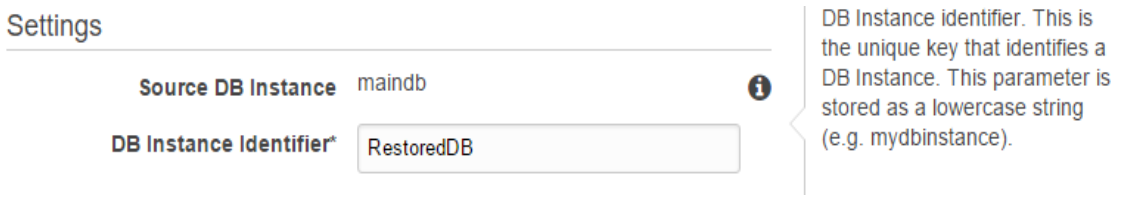
Figur 26: skillelinjen mellem den sidst indsatte linje i første del, samt den første i 2. del

For at genskabe databasen så tabellen indeholder 3501 (den røde pil) rækker skal der laves en genskabelse af databasen som den var i tidsrummet mellem 21:13:40 og 21:15:45. Dette kan lade sig gøre, da AWS laver en fuld backup hver dag, samt Inkrementelle backups hver gang en transaktion bliver lavet. Derfor kan der ved en gendannelse vælges et præcist tidspunkt hvortil man vil gendanne. Se nedenfor.

Figur 27: gendannelse til specifikt tidspunkt

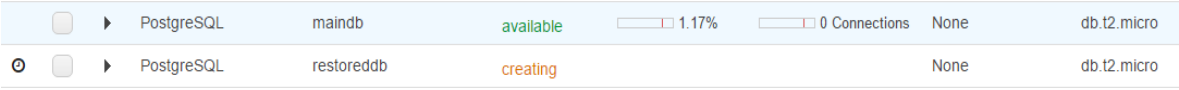
For at gøre det mere sikkert at lave en backup, bliver den nuværende database ikke overskrevet med en tidligere version, men en helt ny instans af den tidligere version oprettes. Dette betyder at man kan tjekke om gendannelsen er gået succesfuldt, før man skifter over til den gendannede database.

Nedenfor ses hvordan man navngiver sin backup, hvorefter der oprettes en ny instans af den gamle database.



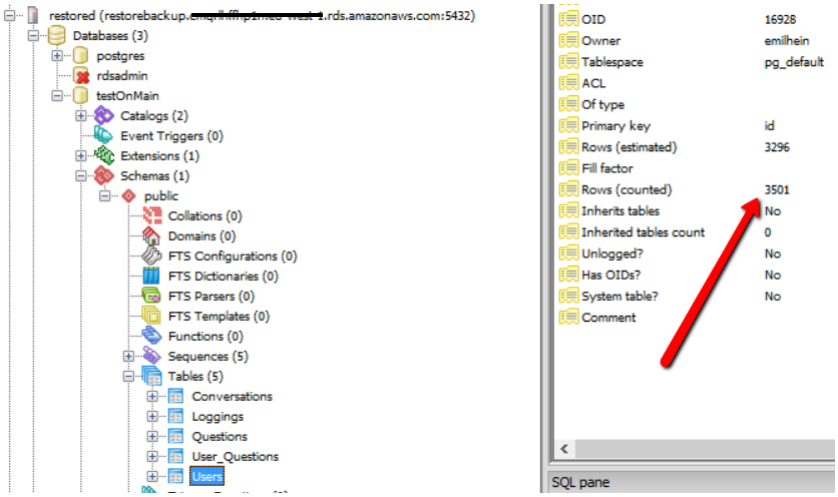
Figur 28: navngivning af den nye database

Efter et navn til den nye database er valgt, vil AWS begynde at gendanne en kopi af den gamle database, i en ny instans.



Figur 29: Oprettelse af den nye database

Efter oprettelsen af den nye database, kan resultatet tjekkes i pgAdmin:



Figur 30: genskabelse af data ud fra specifik transaktion

På figuren ovenfor kan ses at den genskabte database indeholder de ønskede 3501 rækker og databasen kan nu siges at være genskabt til det ønskede tidspunkt.

Test 1 - Analyse

Hvad enten man vil gøre brug af AWS' mulighed for at gendanne en database til et snapshot taget tidligere, eller bruge muligheden for at gendanne til et specifikt tidspunkt, er det nemt. Som nævnt i afsnittet omkring prismodellen er der flere hensyn at tage når denne proces sker. Da der oprettes en ny database ved gendannelse, er det vigtigt at man sletter den gamle, da den ellers optager ressourcer og koster penge. Derefter skal det system der bruger databasen selvfølgelig sættes til at pege på den nye.
Selvom alt naturligvis kan opsættes automatisk, fandt jeg dog en ting som kan skabe problemer, hvis det bliver vigtigt at genskabe til et meget specifikt tidspunkt. Da en AWS konto kan bruges i alle tidszoner er det vigtigt at man har øje for hvilken tidszone man vil gendanne til. I dette tilfælde var det rimelig overskueligt, da det kun var en database, jeg skulle holde styr på.
For at vise vigtigheden, kig da på figur 27. Her ses det at de to række er indsat i tidszonen UTC + 0, mens gendannelsen skal ske i UTC + 1 (figur 28). Derfor skal man være meget opmærksom, hvis man har mange databaser fordelt over mange tidszoner.

Test 2 - EC2 autoskallering og ydelse

Denne test har til formål at teste autoskalleringsmuligheden og derved få en praktisk forståelse af hvordan denne kan bruges.
Testen består i at opsætte parametre der bestemmer om et website skal autoskalere og derved oprette en ny instans af serveren. Herefter bruges JMeter til at afsende et givent antal forespørgsler til websitet. Slutningsvis skal der tjekkes om autoskalerings-funktionen har givet den ønskede effekt.

Testen bliver udført på en applikation, der til testen kun skal kunne besvare tre http GET kald og returnere noget HTML kode. Først opsættes reglen for hvornår en skallering skal finde sted. Mange virksomheder vil typisk sætte disse til f.eks. cpu for eller memory forbrug på serveren. Dette er som regel for at undgå lange svartider hos de besøgende på et website. Alt dette er selvfølgelig meget forskelligt fra virksomhed til virksomhed, og det er svært at opsætte generelle regler for hvornår det er en god ide at skalere op eller ned. For at

forsimple testen og gøre den 100 % kontrollerbar, har jeg valgt at sætte en skalleringsregel der igangsættes efter følgende parametre:

Opskalér med en instans hvis: systemet modtager over 100 forespørgsler over en tidsperiode på 1 minut.

Nedskaler med en instans hvis: systemet modtager under 50 forespørgsler over en tidsperiode på 1 minutter.

Systemet er fastlåst til max at have to instanser.

JMeter bruges til at skabe 600 forespørgsler fra 10 forskellige brugere, og stiger derfor over grænsen der medføre en opskalering på en instans.

1. Testens udgangspunkt.

I AWS konsollen ser den service der skal testes, ud som følgende med en instans:

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Launch Time
RESTSMSService	i-0604bb8b	t2.small	eu-west-1b	running	2/2 checks...	November 25, 2015 at 7:58:04 PM UTC+1

Figur 31 Start af test med en instans

JMeter testen kan nu påbegyndes og der startes i denne forbindelse, med opsættelse af selve testen i JMeter. Den fulde opsætning kan læses i bilag ”Test 1 - opsætning”.

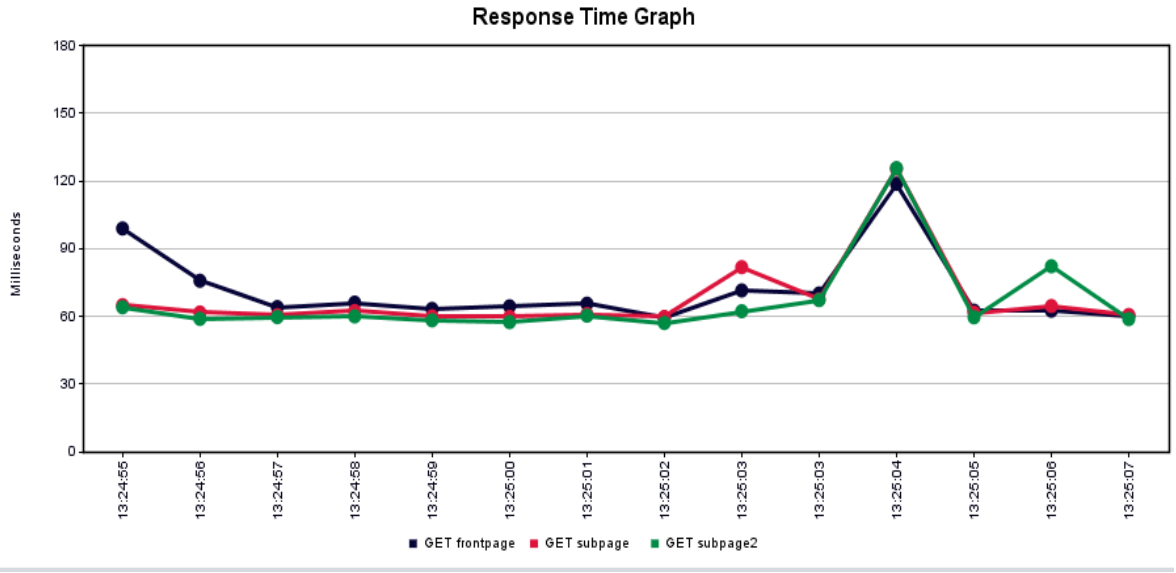
2. Testen startes.

Herunder ses et overordnet resultat for testen:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
GET front...	200	68	56	348	25,89	0,00%	15,4/sec	30,79	2053,6
GET subp...	200	66	57	358	30,93	0,00%	15,5/sec	30,13	1996,1
GET subp...	200	64	55	358	35,82	0,00%	15,5/sec	26,51	1754,6
TOTAL	600	66	55	358	31,17	0,00%	45,6/sec	86,21	1934,8

Figur 32: Test med HTTP GET mod 1 instans

Grafen nedenfor viser svartiderne for hver forespørgsel.



Figur 33: Graf over responstid fra 1 instans

En opskalering sker nu grundet de 600 forespørgsler.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Launch Time
	i-0ae27d87	t2.small	eu-west-1b	pending	Initializing	December 11, 2015 at 1:27:14 PM UTC+1
RESTSMSService	i-de5e2355	t2.small	eu-west-1a	running	2/2 checks...	December 11, 2015 at 12:52:13 PM UTC+1

Figur 34: Oprettelse af ny instans, grundet tærskelværdi overskredet

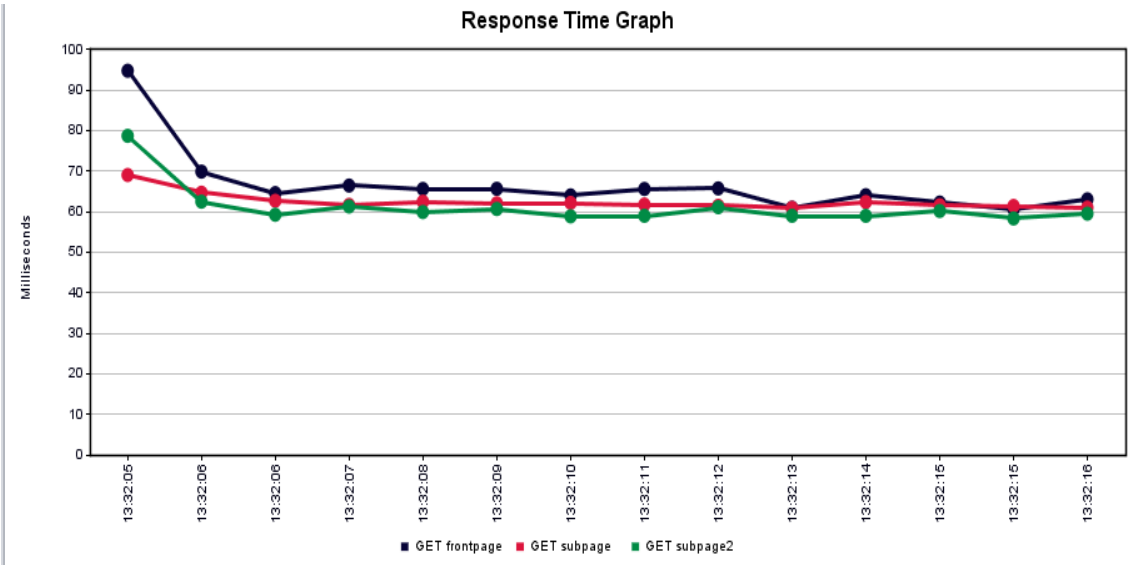
3. Testen køres igen

600 forespørgsler laves igen mod de nu 2 instanser. Dette medføre ikke endnu en opskalering, da der er sat grænsen på max 2.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
GET front...	200	65	57	165	13,71	0,00%	15,7/sec	31,58	2053,7
GET subp...	200	62	57	79	3,70	0,00%	15,9/sec	30,93	1995,5
GET subp...	200	60	55	95	4,64	0,00%	15,9/sec	27,22	1754,6
TOTAL	600	62	55	165	8,87	0,00%	46,8/sec	88,37	1934,6

Figur 35: Test med HTTP GET mod 2 instanser

Herunder ses svartiderne for forespørgslerne mod to instanser. Denne kan direkte sammenlignes med figur 33.



Figur 36: Graf over responstid fra 2 instanser

5. Testen afsluttes

Efter den opsatte tidsperiode vil systemet uden flere forespørgsler selv nedskalere, så der kun er en instans

Herunder ses hvordan der påbegyndes en nedlukning af en instans.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Launch Time
RESTSMSService	i-0ae27d87	t2.small	eu-west-1b	running	2/2 checks...	December 11, 2015 at 1:27:14 PM UTC+1
RESTSMSService	i-de5e2355	t2.small	eu-west-1a	shutting-do...		December 11, 2015 at 12:52:13 PM UTC+1

Figur 37: nedskallering af instans, grundet nedre tærskelværdi ramt

Her er instans ID: i-de5e2355 lukket.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Launch Time
RESTSMSService	i-0ae27d87	t2.small	eu-west-1b	running	2/2 checks...	December 11, 2015 at 1:27:14 PM UTC+1
RESTSMSService	i-de5e2355	t2.small	eu-west-1a	terminated		December 11, 2015 at 12:52:13 PM UTC+1

Figur 38: instans er lukket ned

Test 2 - analyse

Det er værd at bemærk at, hvorvidt ”*http Cache Manager”* (et modul i JMeter) er aktiveret eller ej, ingen forskel gør i denne test. En forespørgsel vil tælle som et kald til serveren, uanset om indholdet er cachet eller ej. Hvis autoskalering skal have til formål, at en hjemmeside kan besvare kunders forespørgsler inden for et accepteret tidsrum, vil andre

parametre end blot antal forespørgsler nok være mere brugbare. Som der kan udledes af figur 34 og 37 er der kun en lille ydelsesforbedring i figur 37, som har en instans mere. Dette kan skyldes flere ting, men er højst sandsynligt et udtryk for at denne test ikke belaster en instans mere end den kan levere, og derfor gives der ikke bedre ydelse ved en ekstra instans i dette tilfælde. Om en ekstra instans kan betale sig i forhold til en cost-benefit analyse af ydeevne over for pris er meget individuelt, men flere studier, herunder *Vladimir Stantchev* [13] viser, at der i hvert fald i visse tilfælde er meget ydeevne at hente i denne skalleringsmulighed.

I forhold til den praktiske læring i dette, er det nemt at opsætte regler der kan hjælpe med at kunne levere den ønskede ydelse i alle tilfælde. Denne funktion er utrolig brugbar og er især vigtig når man skal tænke på prismodellen der bliver brugt hos cloud udbydere. Et andet eksempel, jeg har fundet brugbart har været tidsbestemt skallering:

Name	Limits	Next occurrence (UTC)
nightClose	Min: 0, Max 0	2015-12-11 16:45:00 UTC+0000
morningOpen	Min: 1, Max 4, Desired: 1	2015-12-12 06:30:00 UTC+0000

Figur 39: tidsskallering af system

På ovenstående måde kan man have applikationer der ikke nødvendigvis er åbent for offentligheden, men måske er i test. Her er systemet sat til at lukke ned om eftermiddagen, og starte op igen om morgenen. På den måde betaler man ikke for ressourcer om natten, som man alligevel ikke bruger.

På figur 36 ses at standard afvigelsen er bemærkelsesværdigt lavere for testen på 2 instanser end for den med en instans (figur 33). Dette kan bruges som en kvalitetssikring eller som mål for autoskallering. I nogle tilfælde kan stabilitet være af højeste betydning. Omfanget af denne test gør dog, at det ikke endegyldigt kan bestemmes hvad afvigelsen skyldes. For mere præcise testresultater, skal der bruges dedikerede testmaskiner, samt flere testdata.

Test 3 - funktionel Selenium test

Den følgene test vil give indblik i hvordan man kan bruge Selenium tests, til at teste funktionelle elementer på en hjemmeside. Metoden har jeg brugt til at teste flere forskellige sider, og de kan med fordel også bruges før deployment, af en ny version af en hjemmeside.

Testen bliver foretaget på Freetrailer' nuværende hovedside www.freetrailer.dk. Der vil blive testet bestemte elementer på forskellige underside, men hovedformålet med testen er at tjekke, at et normalt flow kan gennemkøres. Testen her ikke udført på en instans hostet hos AWS, men præcis samme fremgangsmåde kan udføres på en migreret instans.

Det overordnede formål med denne test er ikke at teste en specifik funktion relateret til AWS eller cloud services, men derimod vise hvordan en rigtig test kunne se ud, som kan bruges til at påvise en succesfyldt migrering.

Testen er som de andre tests udført i JMeter og et udklip af testen kan ses herunder. Resten kan læses i Bilag A3 sammen med alle resultaterne af testen:

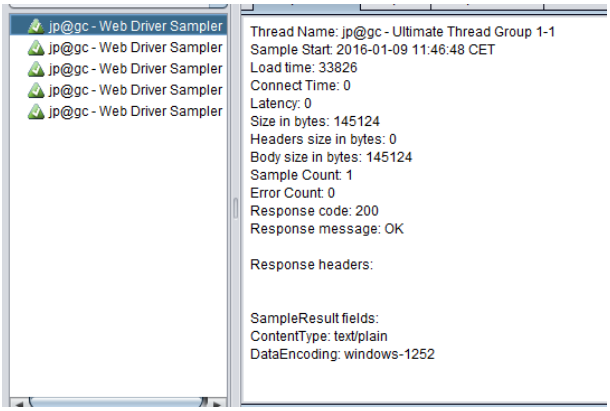
Kode stykke 4:

```
WDS.browser.get('http://www.freetrailer.dk')
//find search field
var searchField = driver.findElement(pkg.By.id('PostCode'))
searchField.click()
searchField.sendKeys(['Gentofte'])

//click search button
driver.findElement(pkg.By.xpath("//button[@type='submit']")).click();
var rentButton =
wait.until(ui.ExpectedConditions.presenceOfElementLocated(pkg.By.cssSelec
tor("div.trailer-CTA > span"))) //wait
rentButton.click();

var alert = wait.until(ui.ExpectedConditions.alertIsPresent()) //wait
alert.accept();

var waitforform =
wait.until(ui.ExpectedConditions.presenceOfElementLocated(pkg.By.id('firs
tName')))
var firstname = driver.findElement(pkg.By.id('firstName'))
firstname.click()
firstname.sendKeys(['Emil Hein'])

var phone =  driver.findElement(pkg.By.id("telephoneNumber"));
phone.click()
```

```
phone.sendKeys(["61796822"])
```

Ovenstående kode bruge bestemte elementer på siden til at interagere med siden og indsætte data, som var det en rigtig bruger. I praksis køres testen i JMeter, hvorefter et Firefox vindue åbner og Selenium begynder at bevæge sig rundt på siden.

Da Selenium i denne tests er sat til rent faktisk at udfører handlingerne i Firefox, vil testen kun indeholde 5 brugere, grundet kravet om ressourcer ved disse tests. Resultatet af testen kan ses her under:



Figur 40: Selenium test resultat

Test 3 - analyse

Det smarte med Selenium tests er at de kan køres i mange forskellige miljøer. Der findes plugins til forskellige browsere, som kan lave disse tests direkte. Hvis man ønsker specifikke data og analyser er værktøjet JMeter dog påkrævet.

Da en Selenium tests i dets natur, svarer til en bruger der udfører bestemte handlinger, er det ikke anbefalet at bruge Selenium som en load test, da det tager for mange ressourcer fra testcomputeren. Der findes dog services der kan gøre dette på en decentraliseret måde så man kan opsætte netværk af computere der kører disse test.
Det er smart at kunne opsætte en test på sit nuværende system og herefter kunne bruge samme test på en migrerede applikation. Hvis migreringen er er sket uden fejl, burde samme test bestå.

Ofte har tests af nye deployments været en manuel gennemgang af applikationen. Dette bliver overflødigt, hvis man har opsat fyldestgørende Selenium tests. Disse kan sættes op til at kører automatisk og kun give besked hvis en fejl findes.

Diskussion

At migrere til skyen er en stor omvæltning for alle firmaer, men der er penge at spare hvis man gør det ordenligt, og derfor er det et område alle virksomheder burde undersøge. Som så mange andre ting, er fordelene og ulemperne utrolig afhængige af unikke behov og ressourcer. Derfor kan der ikke gives en klar formel til at bestemme, om det er smart for en virksomhed, at få sin IT infrastruktur hostet i skyen.

Hvert firma er nødt til at opveje fordele mod ulemper, i forhold til lige præcis deres situation. På bundlinjen er penge selvfølgelig et vigtigt punkt, men også alle de nye muligheder er værd at bide mærke i.
Det er langt mere dynamisk at opsætte sin infrastruktur hos en ”*Cloud udbyder”*, da disse ofte har mange forskellige værktøjer til rådighed, som gør det nemt at tilpasse ens infrastruktur.

Det meste af denne opgave har fokuseret på AWS, men mange af de andre store udbydere har andre fordele, som vil tale til bestemte kunder. Traditionel hosting har typisk foregået hos mindre og lokale firmaer, som modsat de store ”*Cloud udbydere”* ikke har nær så mange ressourcer. Dette betyder bl.a. at dokumentation og hjælp til alle de forskellige problemer og funktioner er veldokumenteret, og nem at finde. Dokumentationen findes i flere forskellige former, men det jeg i forløbet har fundet specielt brugbart er online fora og den originale dokumentation udgivet af AWS. Dokumentationen er fyldestgørende, hvilket gør at mange problemer kan undgås hvis man læser ordenligt op, før man går i gang. Det store fællesskab der naturligt opstår omkring disse udbydere betyder også kunder der oplever problemer, kan dele deres løsninger og berette, om hvordan de kan undgås.

Jeg har erfaret at det er utrolig vigtigt, at man kan prøve tingene i praksis på en lille skala, før man migrerer et produktionsmiljø. Derfor er prøveperioder, med begrænsede ressourcer, en perfekt måde at blive bekendt med systemerne på. Ved brug af specielt AWS har jeg fundet, at stordriftsfordele har gjort mange ting nemmere.

Prismæssigt er der ingen tvivl om, at de billigste priser fås hos de store udbydere som Amazon, Google og Microsoft. Men det er netop også her man skal være utrolig forsigtig. Hvis man benytter den mest almindelige prismodel (Pay-per-use), skal man være klar over præcis hvordan det virker. Ved traditionel hosting, hvor man betaler en fast pris for en server af en hvis størrelse, kommer der ingen ekstraregninger. Dette kan sagtens ske hvis man ikke passer på med Pay-per-use prismodellen. Hvis man udforsker de mere eksotiske prismekanismer, som f.eks. spotpriser hos AWS er der rigtig mange penge at spare.
Jeg har modsat også erfaret, at den nye prismodel tvinger en til at tænke smartere. Tidligere skulle der ikke betales for forbrug, og derfor kunne man uden større betænkning blot bruge løs at harddiskplads, read/writes i databaser, internet throughput og mere. Med den mere fleksible model, hvor man betaler for ressourcer, bliver kunden tvunget til at

optimere koden, så den er så effektiv som muligt. Derfor bliver man ”tvunget” til at optimere sin kode og mange af processerne.

Hvem kan bruge skyen?

En del af besvarelse på problemformuleringen er en analyse af, hvem der kan bruge skyen. Svaret hertil kan uden større analyse beskrevet med, at det kan de fleste virksomheder. Både store og små virksomheder har noget infrastruktur som ville kunne drage fordel af at være hostet i skyen. Der findes så mange forskellige web services at det bliver svært at forestille sig sektorer, som ikke ville kunne drage nytte af det kæmpe potentiale, skyen tilbyder.

To aspekter er vigtige at bringe frem her:
1. For nystartede virksomheder kan det være en fordel at starte med at bruge skyen til hosting, så man fra start af ligger hele sin infrastruktur i skyen. Derfor ser man også mange nystartende virksomheder der bruger skyen.
Her kan bl.a. nævnes: Airbnb, Expedia, Netflix og Foursquare. Disse firmaer har i takt med deres succes, blot opskaleret deres it-infrastruktur. Dette har betydet at de har kunnet fokusere på deres kerneområder, uden at være begrænset at en global infrastruktur.

2. For allerede etablerede virksomheder kommer størrelsen og typen af infrastruktur til at spille en stor rolle i forhold til, om det er god ide at skifte noget, eller alt hosting til skyen. For større virksomheder kan selve mængden af infrastruktur gøre det svært at flytte det hele til skyen. Hvis et firma har investeret store mængder af penge og ressource i at få infrastruktur der passer deres behov, kan det være for stor en investering til at man kan skifte.
En anden vigtig grund til at det kan være svært at flytte til skyen, er hvis firmaet har meget specialiseret behov til enten performance, tilgængelighed eller andet.
Skyen tilbyder nemlig kun et begrænset udvalg af opsætninger. Dette vil for langt de fleste være tilfredsstillende, men der findes eksempler på firmaer der ikke kan flytte deres infrastruktur grundet specialiserede behov.

Ved gennemgang af de forskellige services i AWS har jeg fundet fordele ved at systemer i langt højere grad kan samles, da der udbydes alle de vitale dele for mange systemer. Dette gør administrationen af større systemer mange gange nemmere, da alt kan ordnes fra et enkelt interface i stedet for mange forskellige udbydere med mange forskellige slags interfaces.

Selvom der umiddelbart er mange fordele at hente ved at migrere til skyen er der dog også nogle meget reelle udfordringer, som især mindre virksomheder bliver ramt af. Hvis man er en mindre virksomhed, med få ressourcer og in-house viden, er support utrolig vigtigt i tilfælde af kritiske problemer. Hvis man oplever nedbrud, kan hurtig support være kritisk. Hos AWS er det muligt at købe support med en times svartid, 24 timer i døgnet. Dette er mere end de fleste traditionelle hosting udbydere tilbyder. Til gengæld er det ikke garanteret at man får en supportperson der kan dansk, hvilket kan være en ulempe.

En erfaring, som måske ikke begrænser sig til store cloud udbydere, men som stadigt kan være problematisk for især mindre virksomheder, er den generelle svartid på support. Under nogle af forsøgene i forbindelse med denne rapport, blev der faktureret et uregelmæssigt højt beløb for en instans i en anden tidszone. Da dette ikke går ind under kritiske henvendelse, var jeg nødt til at oprette en ticket i deres supportsystem. Som nævnt i afsnittet omkring prismodellen, skulle der argumenteres for forbrug på enkelte instanser og databaser, hvilket førte til en utrolig lang afklaringstid. Dette kan være problematisk for mindre virksomheder, som ikke har midlerne eller ressourcerne til at gennemgå disse diskussioner over længere perioder. Bevisbyrden er utrolig svær og der skal afsættes en hvis mængde tid, hvis man skal undersøge disse sager i dybden.

En anden, men tilsvarende problemstilling opstår, når virksomheder skal tage stilling til de forskellige "Service level agreements". Her kan det være meget svært at få et samlet overblik over f.eks. kravet om oppetid, hvis mange forskellige services bruges. For virksomheder hvor det er kritisk at et system ikke går ned, skal man være klar over konsekvenserne ved de forskellige "Service level agreements" og hvilken kompensationer man er berettiget til. Denne bevisbyrde kan også være svær at løfte og derfor skal man være fuldstændig klar over de risici der er forbundet med at indgå aftaler med store cloud udbydere som AWS, Google og Microsoft.

Med en opadgående trend i antallet af internet brugere på verdensplan, må kravet til de enkelte services siges at blive mere dynamisk. Man skal derfor hurtigt kunne skalere op og ned, hvilket i disse år giver cloud udbydere stort momentum. Der er ingen tvivl om at sektoren bliver ved med at se en stigning i antallet af virksomheder der bruger skyen. Derfor spår mange også at det vil være en ulempe for virksomheder der ikke vælger at bruge skyen og dens fordele.
Hele paradigmet med cloud computing er stadig relativt nyt og er først i de seneste år begyndt at tage fart, hvilket betyder at der stadig er visse områder der mangler modning førend det er klart til den helt store kommercielle udrulning. Nogle af disse områder omhandler bl.a. standarder for basal udveksling af data, mens andre omhandler lovgivning. Specielt lovgivningen gør at mange virksomheder ikke tør tage chancen forend bestemte lovgivningsmæssige aspekter er på plads. En undersøgelse i de skandinaviske lande peger også på at mange firmaer savner rådgivning omkring brugen af cloud computing. Dette skyldes til dels at mange processer og "bedste praksisser" er fuldstændig ukendt for dem. Hvis man ikke har in-house viden om det, kan det derfor virker afskrækkende, da en fuld migrering til skyen kræver en hvis mængde knowhow. [14] [15]

Processen og selve muligheden for autoskalering giver mange nye muligheder og er nem at opsætte. Autoskallering er en af de elementer der især adskiller sig fra traditionel hosting, og derfor også tiltrækker sig en del opmærksomhed. Virtualiseringens værktøjer gør muligt for kunder at oprette og nedlægge ressourcer på en så dynamisk måde at det kan ændre måden hvor på ressourcer i traditionel forstand tænkes.

Konklusion

I en verden hvor det handler om at være dynamisk og være i stand til at kunne handle hurtigt, er ”Cloud Computing” en mulighed for at kunne eksekvere planer og ideer hurtigere end konkurrenterne.
Jeg har gennem mine test af AWS fundet det utrolig hurtigt at komme i gang med og eksekvere nye projekter. Når man først har styr på hvordan tingene fungere i AWS, er det eneste man skal gøre, blot at kode. Herefter kan man blot spinde en instans op og teste sin kode på den. Det har aldrig været så nemt at komme i gang med at prøve kode på sin egen offentligt tilgængelige server.

Undervejs har der også været problemer, nemlig at bevisbyrden er utrolig svær at løfte, i tilfælde af fejl. Dette er bl.a. en af grundene til at virksomheder skal holde for øje, helt præcist hvor ansvaret ligger. Dette gælder både i forhold til prismæssige og sikkerhedsmæssige aspekter, men også de ”service level agreements” der indgås med udbyderne.

Sløret grænse for ansvarsområder kan være med til at få hele processen til at virke nemmere end den i virkeligheden er. Det meste af sikkerheden forbundet med at hoste sin infrastruktur i skyen, skal varetages af kunden selv.

De nye prismodeller, kan helt klart være en fordel hvis de bliver brugt rigtigt, men kan samtidigt også blive et dyrt bekendtskab, hvis man laver fejl. Derfor er det en både it og økonomiafdelinger der skal gennemse, og blive enige om en prismodel der passer til den enkelte virksomhed.
Hvis det udelukkende er ydelse man er ude efter, så er der ingen tvivl om at skyen er et godt valg. Der er næsten ubegrænsede ressourcer tilgængelige for alle der vil betale for dem.
Slutningsvis er alle overvejelser forskellige fra virksomhed til virksomhed og fra system til system. Derfor burde en intern analyse afgøre om det er fordelagtigt at skifte hosting.
I fremtiden vil flere og flere komme til at bruge store cloud udbydere, og derfor vil der i større grad være fokus på, hvilke fordele hver enkelt virksomhed kan drage af et skifte.

Referenceliste

[1] "Cloud Computing Dominates Deloitte's 2015 Global Venture Capital Confidence Survey - Forbes." [Online]. Available: http://www.forbes.com/sites/louiscolumbus/2015/09/19/cloud-computing-dominates-deloittes-2015-global-venture-capital-confidence-survey/. [Accessed: 26-Nov-2015].

[2] I. Bojanova, J. Zhang, and J. Voas, "Cloud Computing," *IT Prof.*, vol. 15, no. 2, pp. 12–14, Mar. 2013.

[3] D. C. Marinescu, *Cloud Computing: Theory and Practice*. Newnes, 2013.

[4] "Number of Internet Users (2015) - Internet Live Stats." [Online]. Available: http://www.internetlivestats.com/internet-users/. [Accessed: 21-Nov-2015].

[5] D. C. Marinescu, *Cloud Computing: Theory and Practice*. Newnes, 2013.

[6] A. Khajeh-Hosseini, D. Greenwood, and I. Sommerville, "Cloud migration: A case study of migrating an enterprise IT system to IaaS," *Proc. - 2010 IEEE 3rd Int. Conf. Cloud Comput. CLOUD 2010*, pp. 450–457, 2010.

[7] S. Kansal, G. Singh, H. Kumar, and S. Kaushal, "Pricing Models in Cloud Computing," in *Proceedings of the 2014 International Conference on Information and Communication Technology for Competitive Strategies - ICTCS '14*, 2014, pp. 1–5.

[8] J. Weinman, "Cloud Pricing and Markets," *IEEE Cloud Comput.*, vol. 2, no. 1, pp. 10–13, Jan. 2015.

[9] P. Patel, A. Ranabahu, and A. Sheth, "Service Level Agreement in Cloud Computing," *Cloud Work.*, 2009.

[10] M. A. Babar and M. A. Chauhan, "A tale of migration to cloud computing for sharing experiences and observations,"

Proceeding 2nd Int. Work. Softw. Eng. cloud Comput. - SECLOUD '11, p. 50, 2011.

[11] "A five-phased approach for the cloud migration." [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.413.6310&rep=rep1&type=pdf. [Accessed: 20-Nov-2015].

[12] A. Holmes and M. Kellogg, "Automating Functional Tests Using Selenium," in *AGILE 2006 (AGILE'06)*, 2006, vol. 2006, pp. 270–275.

[13] V. Stantchev, "Performance Evaluation of Cloud Computing Offerings," in *2009 Third International Conference on Advanced Engineering Computing and Applications in Sciences*, 2009, pp. 187–192.

[14] "Benefits of cloud maturity." [Online]. Available: http://pages.tieto.com/rs/517-ITT-285/images/Benefits_of_cloud_maturity.pdf?mkt_tok=3RkMMJWWfF9wsRoiuKzBZKXonjHpfsX76%2B0uXaK1lMI%2F0ER3fOvrPUfGjI4FS8RnI%2BSLDwEYGJlv6SgFT7HCMal0zrgPUxQ%3D. [Accessed: 26-Nov-2015].

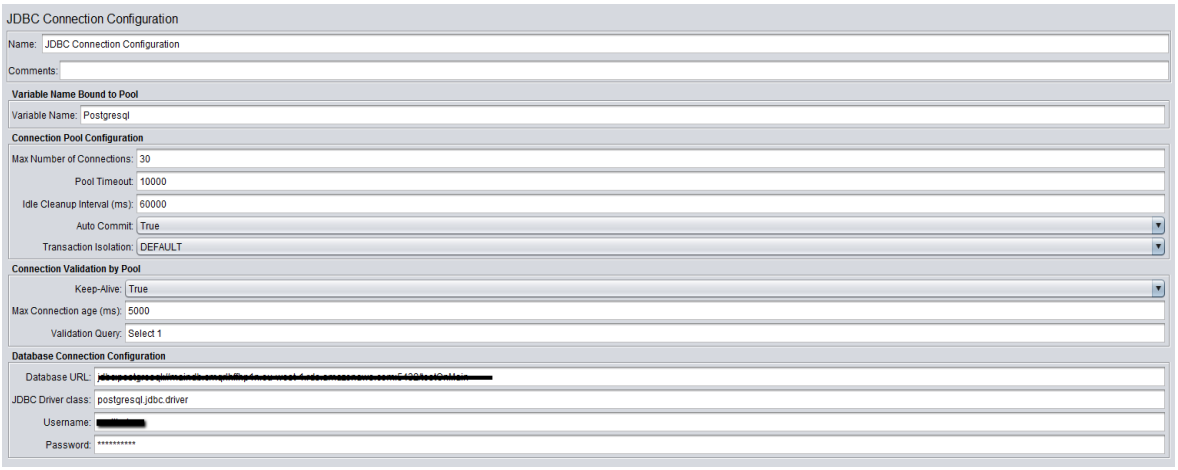
[15] W. Bumpus, "NIST Cloud Computing Standards Roadmap," *NIST Cloud Comput. Stand.*, pp. 1–3, 2013.

Appendix index

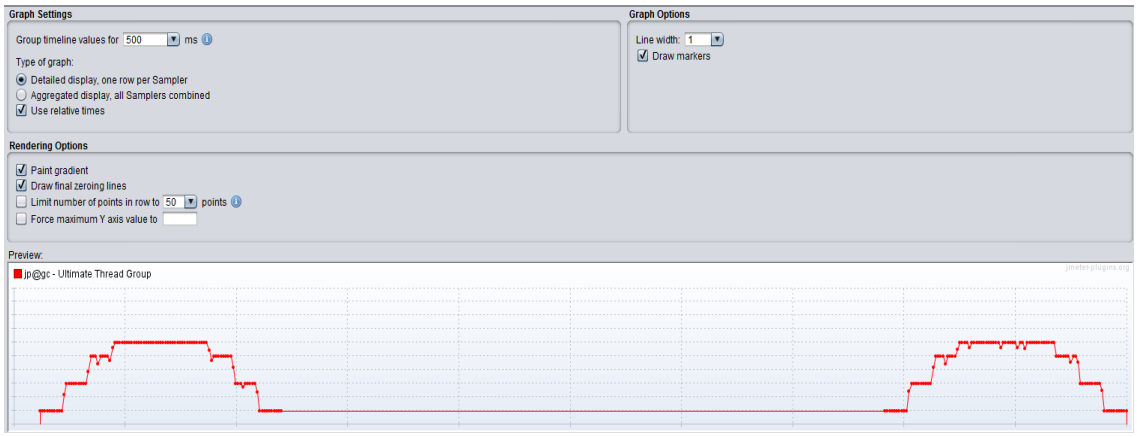
A1: JMeter opsætning

Til test er brugt JMeter 2.13. Denne version kan hentes her:
http://jmeter.apache.org/download_jmeter.cgi

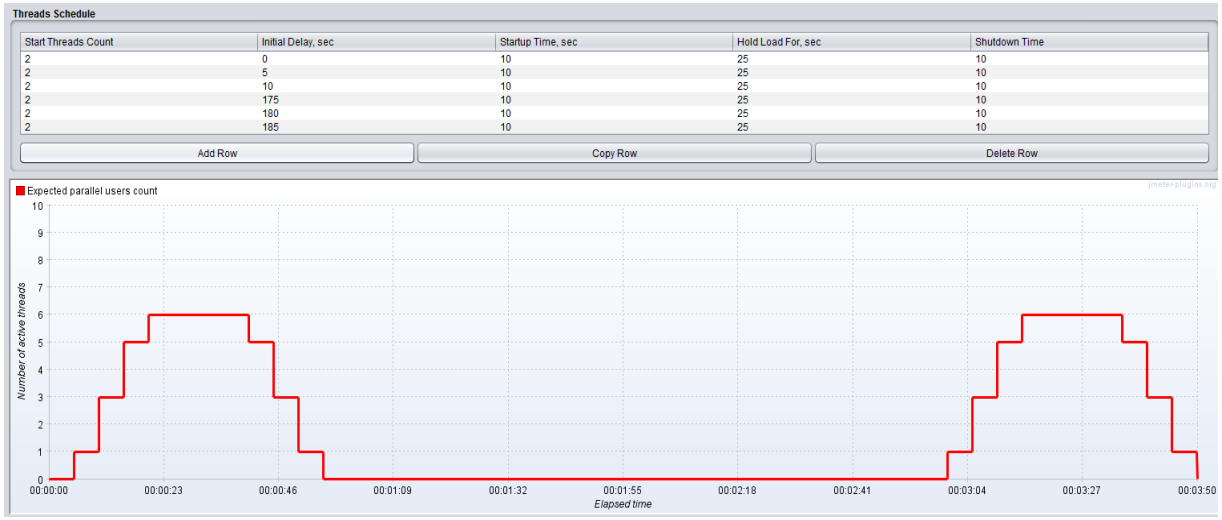
Test 1 - test opsætning



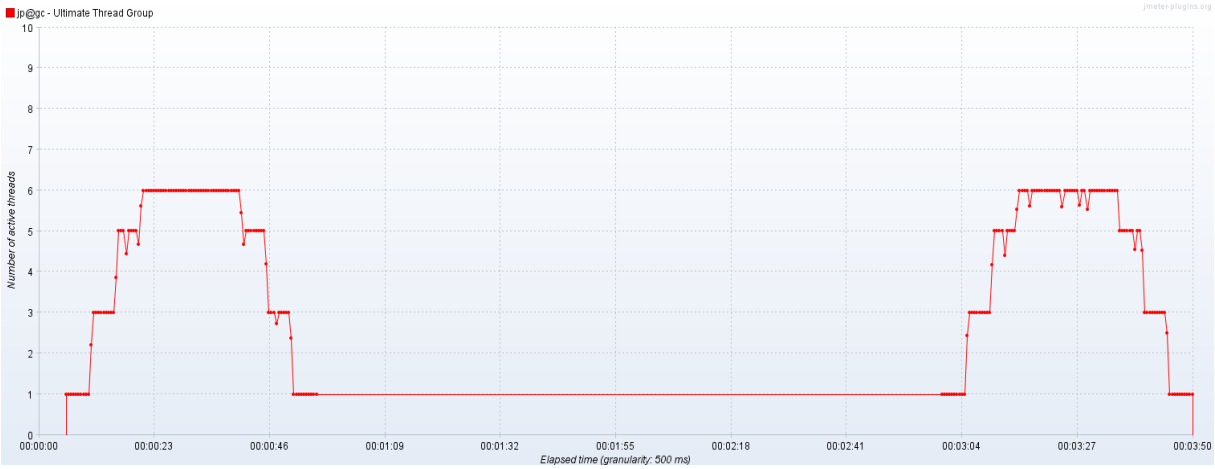
Test 1 - Ultimate Thread group



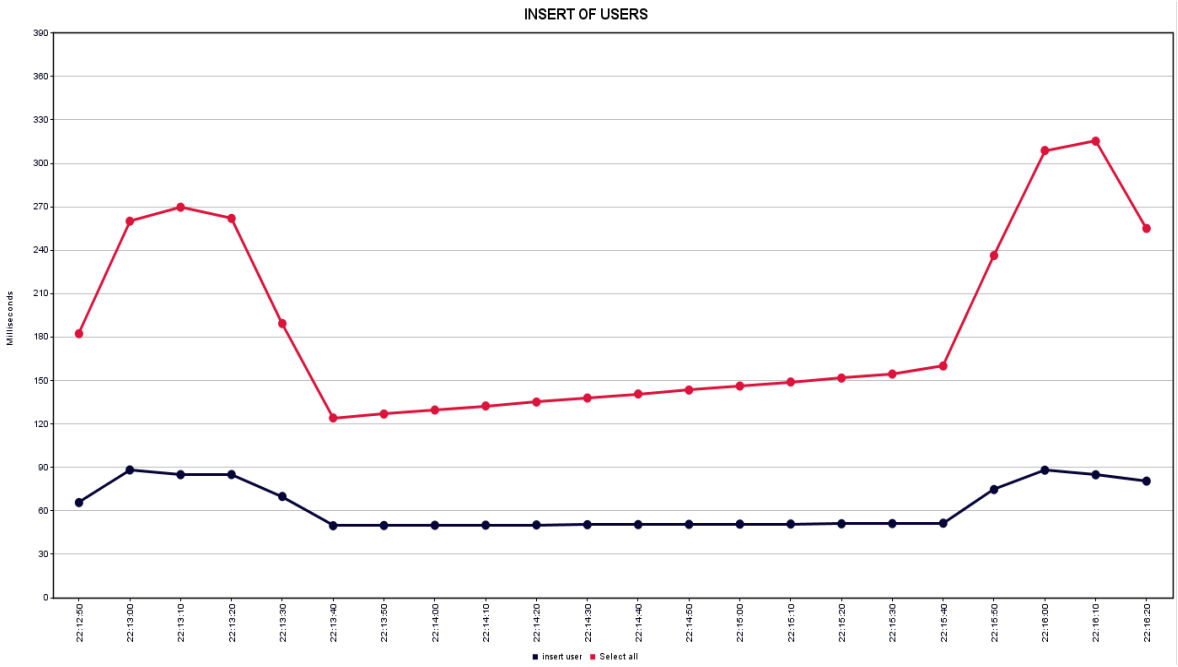
Test 1 - Ultimate thread group - teoretisk



Test 1 - Ultimate thread group - faktisk.



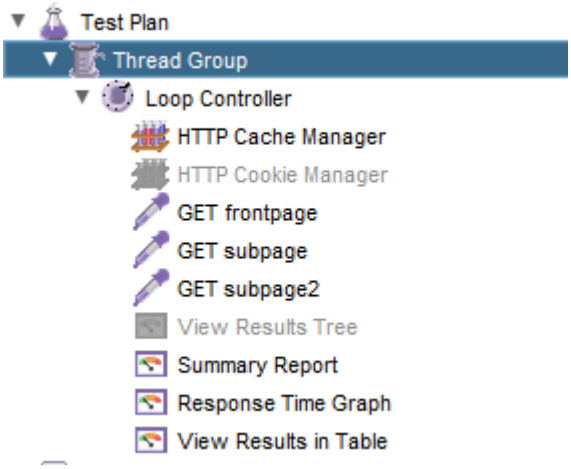
Test 1 - Svartid graf



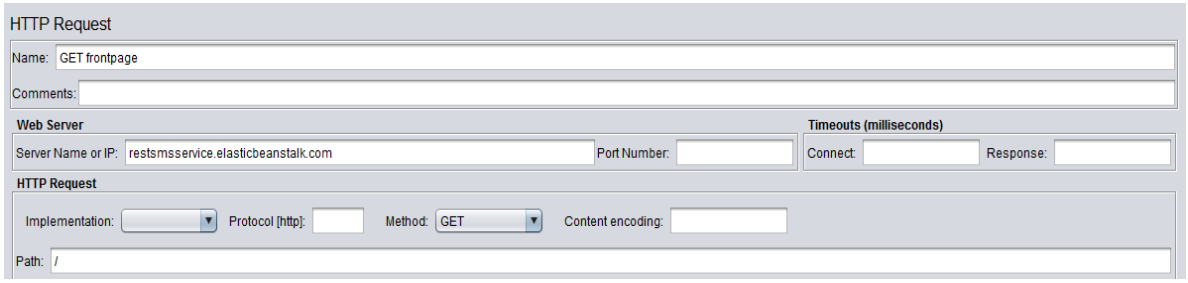
Test 1 - resultat

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
insert user	1233	81	45	420	32,95	0,00%	5,5/sec	0,05	9,0
Select all	1230	258	111	1297	137,58	0,00%	5,5/sec	1294,18	242351,1
TOTAL	2463	169	45	1297	133,61	0,00%	10,9/sec	1293,63	121032,5

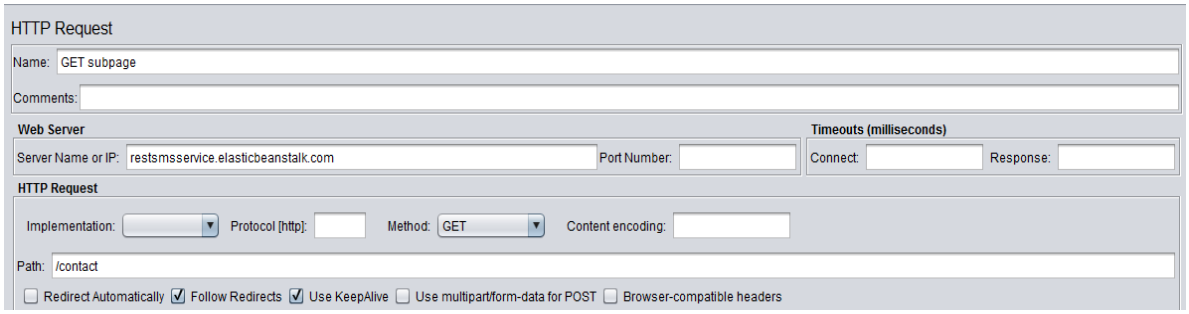
Test 2 - Test opsætning



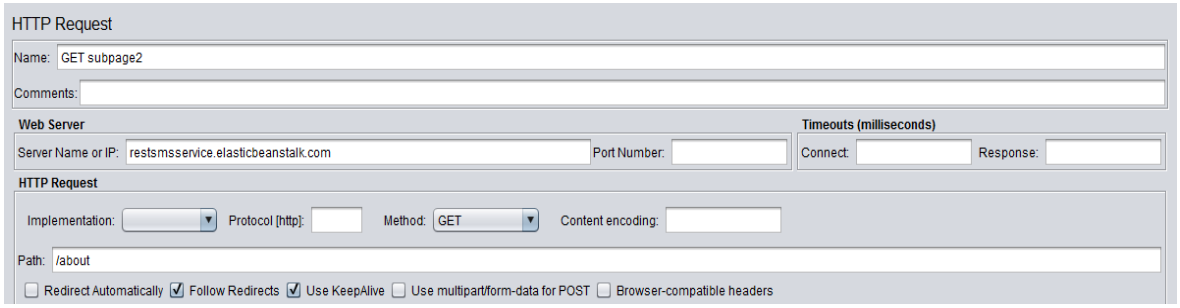
Test 2 - HTTP Request



Test 2 - HTTP Request2



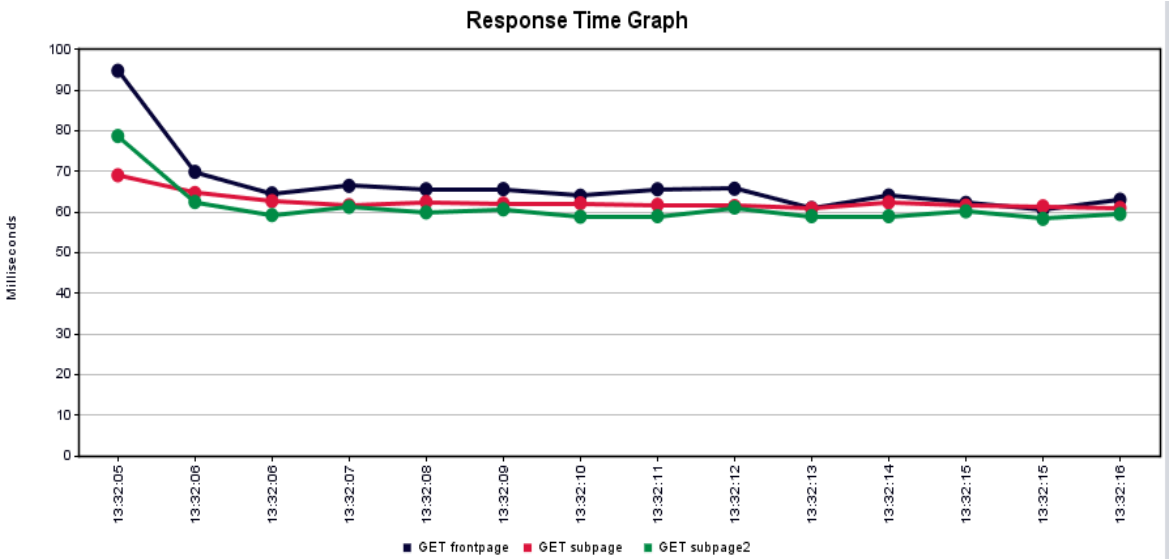
Test 2 - HTTP Request3



Test 2 - Summary Rapport

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
GET frontpage	200	65	57	165	13,71	0,00%	15,7/sec	31,58	2053,7
GET subpage	200	62	57	79	3,70	0,00%	15,9/sec	30,93	1995,5
GET subpage2	200	60	55	95	4,64	0,00%	15,9/sec	27,22	1754,6
TOTAL	600	62	55	165	8,87	0,00%	46,8/sec	88,37	1934,6

Test 2 - Svartidsgraf



Test 2 - Resultat I tabel

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Latency	Connect Time(ms)
1	13:32:05.200	Thread Group 1-1	GET frontpage	165		2051	165	74
2	13:32:05.369	Thread Group 1-1	GET subpage	64		1995	64	0
3	13:32:05.436	Thread Group 1-1	GET subpage2	95		1756	95	0
4	13:32:05.533	Thread Group 1-1	GET frontpage	76		2059	76	0
5	13:32:05.612	Thread Group 1-1	GET subpage	73		1995	73	0
6	13:32:05.688	Thread Group 1-1	GET subpage2	73		1754	73	0
7	13:32:05.763	Thread Group 1-1	GET frontpage	69		2057	69	0
8	13:32:05.835	Thread Group 1-1	GET subpage	70		1995	70	0
9	13:32:05.907	Thread Group 1-1	GET subpage2	68		1754	68	0
10	13:32:05.977	Thread Group 1-1	GET frontpage	69		2053	69	0
11	13:32:06.049	Thread Group 1-1	GET subpage	71		1995	71	0
12	13:32:06.123	Thread Group 1-1	GET subpage2	63		1752	63	0
13	13:32:06.190	Thread Group 1-1	GET frontpage	70		2053	70	0
14	13:32:06.190	Thread Group 1-2	GET frontpage	120		2057	120	47
15	13:32:06.262	Thread Group 1-1	GET subpage	68		1993	68	0
16	13:32:06.312	Thread Group 1-2	GET subpage	68		1995	68	0
17	13:32:06.332	Thread Group 1-1	GET subpage2	70		1754	70	0
18	13:32:06.382	Thread Group 1-2	GET subpage2	71		1754	71	0
19	13:32:06.405	Thread Group 1-1	GET frontpage	63		2051	63	0
20	13:32:06.456	Thread Group 1-2	GET frontpage	61		2051	61	0
21	13:32:06.471	Thread Group 1-1	GET subpage	61		1995	61	0
22	13:32:06.520	Thread Group 1-2	GET subpage	62		1993	62	0
23	13:32:06.534	Thread Group 1-1	GET subpage2	58		1756	58	0
24	13:32:06.584	Thread Group 1-2	GET subpage2	58		1758	58	0
25	13:32:06.594	Thread Group 1-1	GET frontpage	60		2053	60	0
26	13:32:06.645	Thread Group 1-2	GET frontpage	63		2051	63	0
27	13:32:06.656	Thread Group 1-1	GET subpage	62		1995	62	0
28	13:32:06.711	Thread Group 1-2	GET subpage	60		1999	60	0
29	13:32:06.720	Thread Group 1-1	GET subpage2	59		1752	59	0
30	13:32:06.773	Thread Group 1-2	GET subpage2	58		1752	58	0
31	13:32:06.782	Thread Group 1-1	GET frontpage	60		2057	60	0

Scroll automatically? Child samples? No of Samples 600 Latest Sample 57 Average 62 Deviation 8

A2: Selenium opsætning

Selenium samt guide hertil kan læses og downloades på følgende links:

Selenium IDE: http://www.seleniumhq.org/download/

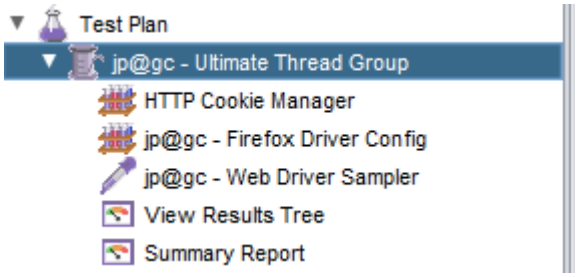
Guide: https://blazemeter.com/blog/jmeter-webdriver-sampler

selenium docs:
http://selenium.googlecode.com/svn/trunk/docs/api/java/org/openqa/selenium/package-summary.html

Udvidelser kan hentes her: http://jmeter-plugins.org/

A3: Selenium test i JMeter:

Opsætningen ser ud som nedenfor:



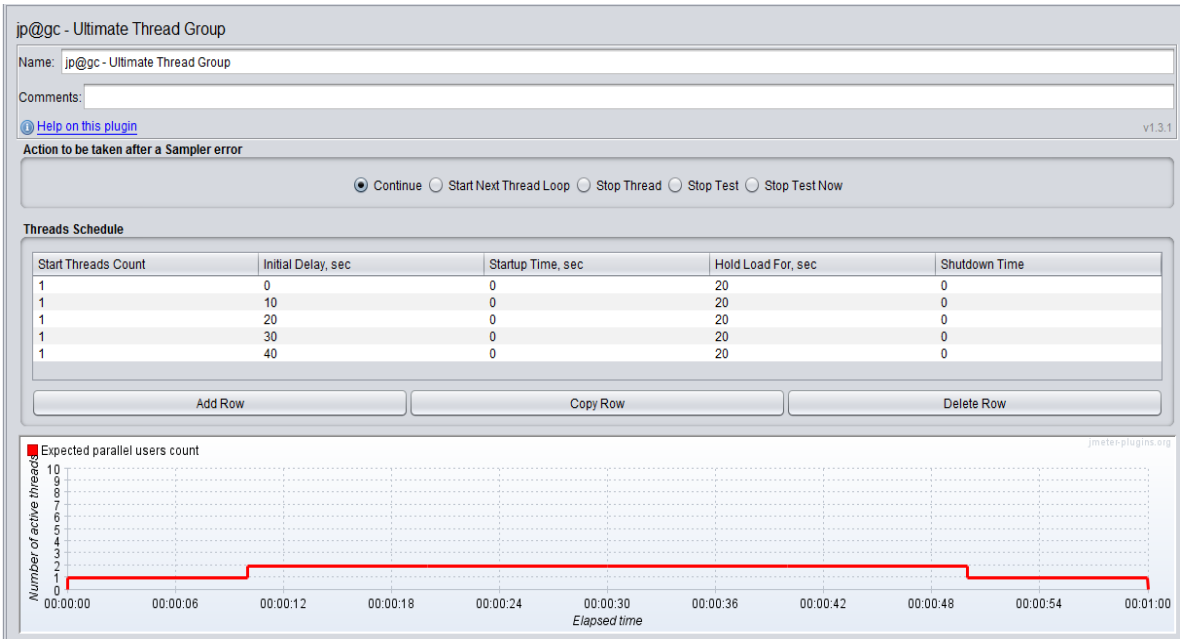
Selve scriptet der køres kan ses nedenfor:

```
var pkg = JavaImporter(org.openqa.selenium)
var support_ui =
JavaImporter(org.openqa.selenium.support.ui.WebDriverWait)
var ui=JavaImporter(org.openqa.selenium.support.ui) //import Selenium
Support UI package
var wait = new support_ui.WebDriverWait(WDS.browser, 5000)
var driver = WDS.browser;

WDS.sampleResult.sampleStart()
WDS.browser.get('http://www.freetrailer.dk')

//find search field
var searchField = driver.findElement(pkg.By.id('PostCode'))
searchField.click()
searchField.sendKeys(['Gentofte'])

//click search button
driver.findElement(pkg.By.xpath("//button[@type='submit']")).click();

var rentButton =
wait.until(ui.ExpectedConditions.presenceOfElementLocated(pkg.By.cssSelec
tor("div.trailer-CTA > span"))) //wait
rentButton.click()

var alert = wait.until(ui.ExpectedConditions.alertIsPresent()) //wait
alert.accept();

var waitforform =
wait.until(ui.ExpectedConditions.presenceOfElementLocated(pkg.By.id('firs
tName')))
var firstname = driver.findElement(pkg.By.id('firstName'))
firstname.click()
firstname.sendKeys(['Emil Hein'])
```

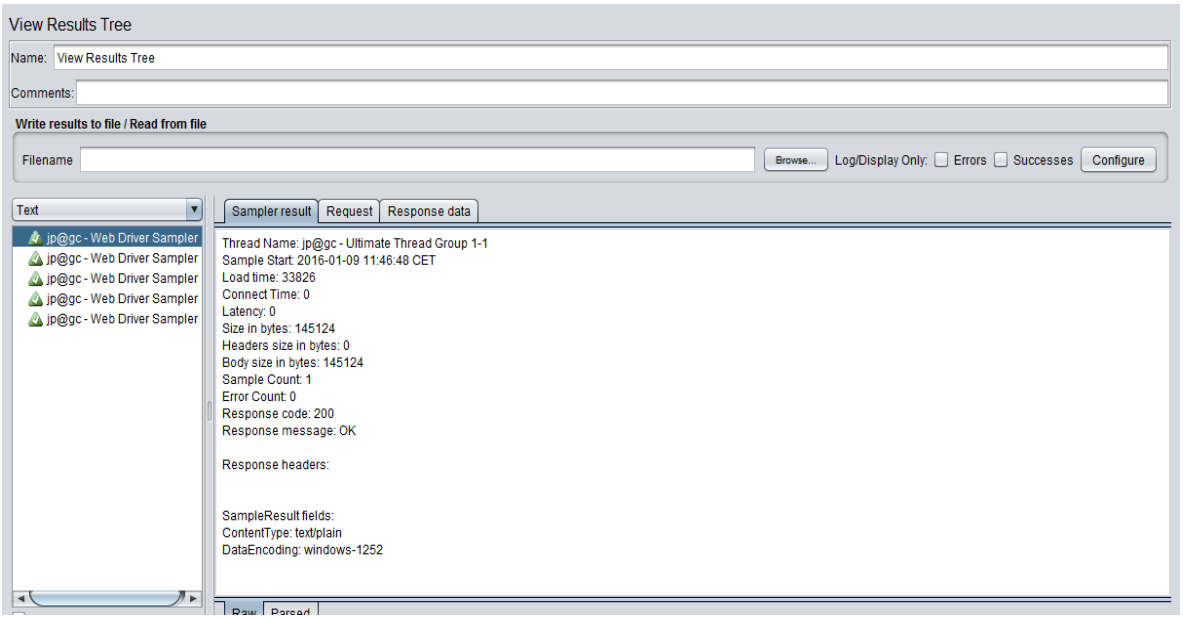
```
 var phone =  driver.findElement(pkg.By.id("telephoneNumber"));
 phone.click()
 phone.sendKeys(["61796822"])

  var postCode =  driver.findElement(pkg.By.id("postCode"));
 postCode.click()
 postCode.sendKeys(["2400"])

   var email =  driver.findElement(pkg.By.id("inputEmail"));
 email.click()
 email.sendKeys(["eh@freetrailer.dk"])

  var street =  driver.findElement(pkg.By.id("streetName"));
 street.click()
 street.sendKeys(["mesterstien 1"])

   var city =  driver.findElement(pkg.By.id("city"));
 city.click()
 city.sendKeys(["København nv"])
var agreement = driver.findElement(pkg.By.cssSelector("span.custom-
checkbox-ico.pull-left"))
agreement.click()

var correctData = driver.findElement(pkg.By.xpath("//div[@id='page-
content']/div/div/div/div/form/div[2]/div[7]/div/div/label/span"))
correctData.click()
driver.findElement(pkg.By.xpath("//button[@type='submit']")).click();
WDS.log.info(WDS.name + ' has logged an entry');
WDS.sampleResult.sampleEnd()
```

Resultat test 3:

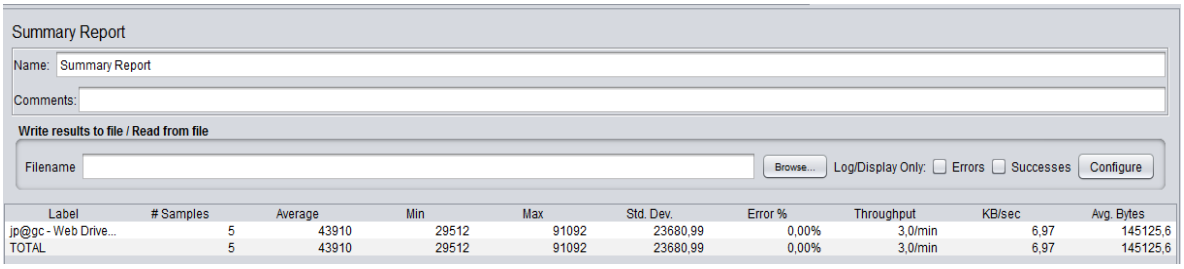
Selenium ultimate thread group



Test 3 Result Tree



Test 3 - Summary Report



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
jp@gc - Web Drive...	5	43910	29512	91092	23680,99	0,00%	3,0/min	6,97	145125,6
TOTAL	5	43910	29512	91092	23680,99	0,00%	3,0/min	6,97	145125,6

Test 3 Aktive trader over tid