CHALMERS

Migrera till molnet

Att välja den bästa lösningen

Examensarbete inom högskoleprogrammet Datateknik

ARIAN ALIKASHANI

ERMIN FAZLIC

INSTITUTIONEN FÖR DATA-OCH INFORMATIONSTEKNIK

CHALMERS TEKNISKA HÖGSKOLA

Göteborg 2024

www.chalmers.se

Examensarbete 2024

Migrera till molnet

Att välja den bästa lösningen

ARIAN ALIKASHANI
ERMIN FAZLIC

CHALMERS

Institutionen för Datateknik
Chalmers Tekniska Högskola
Göteborg 2024

Migrera till molnet
Att välja den bästa lösningen
ARIAN ALIKASHANI
ERMIN FAZLIC

Handledare: Arne Linde, Institutionen för data- och informationsteknik
Examinator: Lars Svensson, Institutionen för data- och informationsteknik

Examensarbete 2024
Institutionen för Datateknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: Himmel med vita moln.

Skriven i LaTeX
Göteborg 2024

Migrera till molnet
Att välja den bästa lösningen
ARIAN ALIKASHANI
ERMIN FAZLIC
Institutionen för Datateknik
Chalmers Tekniska Högskola

Sammanfattning

Att migrera till cloud har blivit ett populärt alternativ för företag som ville ha en flexibel, skalbar och kostnadseffektiv lösning. Denna forskning fördjupar sig i processen att migrera Xpektor från dess nuvarande infrastruktur till en molnbaserad infrastruktur. Genom intervjuer med nyckel personer, benchmarking av de 3 ledande Function-as-a-Service-leverantörerna och en kostnadsanalys, visar studien komplexiteten av molnmigrering. Resultatet från benchmarken mot AWS, Azure och Google Cloud, kombinerat med de associerade kostnaderna och befintlig utvecklarexpertis, ger Azure som rekomendation när det gäller val av molnleverantör.

Nyckelord: Cloud, Migrering, cloud benchmarking, Function-as-a-Service

Förord

Detta examensarbete påbörjades i samband med vår anställning på Wecoorp och utvecklingen av Xpektor. Det var ett guldläge för oss att lära oss om hur olika cloud leverantörer fungerar. Det har varit en rolig resa att se Xpektor och Wecoorp att växa och vi är stolta över att vi har kunnat bidra under denna resa. Vi ville tacka alla på Wecoorp som har hjälpt oss under skrivningen av detta arbete.

Vi ville dessutom tacka alla våra lärare och examinatorer som vi har haft under åren på Chalmers och ett extra speciellt tack till Lars Svensson, examinatorn för denna kurs som har varit så snäll, förstående, hjälpsam och tålmodig med oss.

Arian Alikashani och Ermin Fazlic, Göteborg, Mars 2024

Akronymer

Nedan är listan över akronymer som har använts i detta examensarbete listade i alfabetisk ordning:

API	Application Programming Interface
AWS	Amazon Web Services
FaaS	Function as a Service
GC	Google Cloud
IaaS	Infrastructure as a Service
KPI	Key Performance Indicator
PaaS	Platform as a Service
SaaS	Software as a Service

Innehåll

Figurer

Tabeller

1
Inledning

Med samhällets ökade digitalisering använder sig fler företag av teknologi för att hitta smartare och effektivare lösningar [14] . Detta är relevant för företag som erbjuder tjänster via nätet och vill att deras produkt är smidig och effektiv. Digitaliseringen har öppnat en marknad för molntjänster, där molnleverantörer tar hand om den bakomliggande infrastrukturen. Molntjänster avser leverans av datorresurser, såsom serverkapacitet, lagring, databaser, programvara med mera, över internet. Istället för att vara värd för dessa resurser på egna fysiska servrar eller enheter kan man komma åt dem på distans via ett nätverk av servrar och datacentra som ägs och underhålls av en tredjepartsleverantör [19]. I grund och botten är "cloud computing" som att hyra någon annans datorinfrastruktur, snarare än att köpa och underhålla sin egen. Detta gör det möjligt för individer och organisationer att få tillgång till kraftfulla datorresurser på begäran, utan att behöva göra betydande investeringar i förväg i hårdvara, mjukvara eller IT-personal. Genom att utnyttja molntjänster kan företag dra nytta av fördelar som skalbarhet, flexibilitet, kostnadseffektivitet och förbättrad säkerhet [16]. Dessutom möjliggör cloud computing nya sätt att arbeta, såsom fjärrsamarbete, dataanalys och artificiell intelligens, som skulle vara svåra eller omöjliga att uppnå med traditionella datormodeller.

1.1 Problembeskrivning

Molnleverantörer erbjuder olika tjänster, var och en med sina unika möjligheter, prismodeller och prestandaegenskaper. Följaktligen måste kunderna utvärdera varje leverantörs erbjudanden mot deras specifika behov, med hänsyn till aspekter som kostnad, skalbarhet, säkerhet, datahantering, interoperabilitet och tillförlitlighet. Valet av rätt molnleverantör är av största vikt eftersom det avsevärt påverkar ett företags effektivitet, flexibilitet och produktivitet. Ett felaktigt eller suboptimalt val kan leda till oavsiktliga konsekvenser, såsom ökade kostnader [22].

Men det ökande antalet av leverantörer på marknaden gör denna uppgift allt mer komplex. Leverantörer skiljer sig när det gäller datasäkerhet, teknisk support och avtalsvillkor, vilket ökar förvirringen. Dessutom kan den snabba innovationstakten inom molnbaserade tjänster leda till frekventa förändringar i tjänsteutbudet, vilket gör det ännu mer utmanande att göra ett välgrundat val.

Detta ställer potentiella kunder inför en valparadox som i molntjänst-världen är känd som "Cloud Paralysis" [17] [13]. Cloud Paralysis gäller inte bara om valet av

en leverantör, utan också implementationen av en lämplig lösning efter att man har valt en leverantör. Det beror på att varje leverantör erbjuder flera hundra olika tjänster och det kan orsaka att utvecklarna fastnar i en långvarig forsknings- och kompetensutbildningsfas för att hitta den bästa lösningen anpassad för just deras behov. Medan ökad konkurrens å ena sidan främjar innovation och potentiellt bättre prissättning, skapar den å andra sidan också en komplex beslutsmiljö. Kunderna måste navigera i denna komplexitet för att göra ett optimalt val, en insats som kräver mycket tid, kunskap och resurser.

Dessutom saknas etablerade och universellt accepterade ramverk eller metoder för att hjälpa kunder att effektivt utvärdera sina cloud computing-behov mot erbjudanden från olika leverantörer. Denna brist på standardisering lämnar kunderna utan en färdplan för att vägleda sin urvalsprocess, vilket gör det ännu mer utmanande att anpassa sina tekniska och affärsmässiga krav till en leverantörs kapacitet.

Utöver detta måste kunderna ta hänsyn till framtida tillväxt och potentiella förändringar i sina behov. Detta innebär att uppskatta framtida datavolymer, användarantal och nödvändiga beräkningsresurser, och sedan säkerställa att den valda leverantören kommer att kunna hantera dessa förändringar utan att orsaka betydande kostnadsökningar eller tjänsteavbrott.

1.2 Bakgrund

Wecoorp AB är ett fintech-bolag som startade 2021. Företaget utvecklar en applikation, Xpektor[1], som integrerar med kundernas bokföringssystem. Systemet är lämpligt att använda för redovisningsbranschen, banker och styrelseledamöter. Med hjälp av bokföringssystemets API hämtar applikationen värden som kundfakturor, leverantörsfakturor, periodiseringar med mera, för att kunna presentera dessa överskådligt. Dessa värden används sedan i en algoritm som ger en överblick över företagets ekonomi samt signalerar om något behöver åtgärdas. Applikationen ger även en beslutsmall som guidar användaren till att lösa olika problem som uppmärksammas. Dessutom har den även en funktion för att skicka e-post till valda mottagare när något kritiskt behöver uppmärksamhet. I och med Xpektors framgång inom branschen, krävs det en skalbar lösning och arkitektur för att kunna expandera och erbjuda deras tjänster till flera kunder. Detta har lett till ökad intresse hos Wecoorp att flytta lösningen till en molnleverantör.

1.3 Syfte

Syftet med denna uppsats är att ge exempel på hur företag kan undvika “Cloud paralysis” och utveckla en egen färdplan för att utvärdera sina specifika behov och välja den mest lämpliga leverantören av molnbaserade tjänster. Dessutom kommer vi att försöka belysa de viktigaste överväganden som bör ligga till grund för denna

[1] https://xpektor.se/

beslutsprocess, inklusive att identifiera affärsmässiga och tekniska krav, förstå de olika typerna av molntjänster och deras implikationer och utvärdera olika leverantörers erbjudanden mot dessa krav.

1.4 Mål

Målet med detta arbete är att hjälpa utvecklarna på Wecoorp att välja den bästa molnleverantören till Xpektor utifrån deras behov.

1.5 Avgränsningar

Eftersom det finns många molnleverantörer kommer projektet att begränsas till de tre största molnleverantörerna. När det gäller testning av FaaS [2] tjänsterna kommer runtimen att reduceras till bara .Net [11]. Testerna kommer att utföras på det närmaste servern till Sverige från en lokal dator. Dessutom kommer testerna reduceras bara till load testing och spiktesting på max 1 minut eftersom kostnaderna kan stiga upp väldigt fort. Funktionerna kommer att bara testas i kallstart, detta är eftersom man kan se varmstart beteendet efter att kallstarten har inträffat.

[2]Function-as-a-Service beskrivs i sektion 2.1

2
Teknisk Bakgrund

I följande kapitel kommer det ges en övergripande förklarning om “Cloud Computing”, “Serverless Computing”, “Benchmarking” och Microservice-arkitektur.

2.1 Cloud Computing

Cloud Computing är ett brett utbud av molntjänster som erbjuds av molnleverantörer [20]. Dessa tjänster levereras på begäran (on-demand/on-premise) till företag och kunder över internet. Tjänsterna som erbjuds är skapade för att ge enkel och prisvärd tillgång till applikationer och resurser, utan behov av egen infrastruktur eller hårdvara.

Molntjänster kan kategoriseras i fyra huvudtyper:

1. **Infrastructure as a Service (IaaS):** Denna tjänst tillhandahåller infrastrukturen som virtuella maskiner och andra resurser, som block- och filbaserad lagring, brandväggar, lastbalanserare (load balancers), IP-adresser, virtuella lokala nätverk mm. Exempel på IaaS är Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GC).
2. **Platform as a Service (PaaS):** Den här tjänsten används för applikationer och annan utveckling. Detta inkluderar utvecklingsverktyg, databashantering, business intelligence (BI)-tjänster, mm. PaaS är främst användbart för utvecklare och programmerare, och låter användaren att utveckla, köra och hantera sina egna appar utan att behöva bygga och underhålla den infrastruktur eller plattform som vanligtvis förknippas med processen. Utvecklaren kan använda inbyggda programvarukomponenter för att skapa sina applikationer, vilket minskar mängden kod de måste skriva själva. Exempel på dessa är Google App Engine, AWS Elastic Beanstalk och Heroku.
3. **Software as a Service (SaaS):** I denna tjänstemodell tillhandahålls de molnbaserade applikationerna till användaren, som en tjänst på begäran i form av en hel applikation som hanteras av en leverantör via en webbläsare. Exempel på SaaS är Google Workspace, Microsoft Office 365, Salesforce.
4. **Function as a Service (FaaS):** Detta är en kategori av molntjänster som erbjuder en plattform som tillåter en programmerare att utveckla, köra och hantera applikationsfunktioner utan komplexiteten att bygga och underhålla den infrastruktur som vanligtvis förknippas med att utveckla och starta en app. Exempel på FaaS är AWS Lambda, Google Cloud Functions och Azure Functions.

Molntjänsterna är vanligtvis prissatta med olika ”pay-as-you-go” prenumerationsmodeller. Kunder debiteras oftast, bara för resurser de förbrukar, till exempel hur lång tid det tar att köra en funktion, lagringskapaciteten eller antal virtuella maskiner som används.

2.2 Serverless computing

Serverless computing är en paradigm där molnleverantören automatiskt hanterar den infrastruktur som krävs för att köra och skala applikationer. Detta gör att utvecklare kan fokusera mer på att skriva koden för sina applikationer och mindre på att hantera servrar, distribution och skalning [9].

Termen “serverless” kan vara något missvisande eftersom det fortfarande finns servrar inblandade i att köra applikationer. Ansvaret för att hantera, skala och underhålla dessa servrar ligger dock helt i händerna på molnleverantören. Molnleverantörer använder sina egna datacenter och beräkningsresurser för att hantera och underhålla den molnbaserade applikationen eller tjänsten. Detta innebär att utvecklaren inte behöver oroa sig för serverdrift och kan fokusera på funktionaliteten i sina applikationer.

Serverless computing associeras ofta med FaaS-erbjudanden som AWS Lambda, Google Cloud Functions och Azure Functions. I en FaaS-modell skriver utvecklaren individuella funktioner som triggas av händelser, oftast i form av HTTP anrop. Dessa funktioner kan vara allt från en kodbit som ändrar storlek på en bild när den väl har laddats upp till en server, till en funktion som hämtar data från en databas när en användare gör en förfrågan. När en händelse utlöser en funktion, allokerar molnleverantören nödvändiga resurser, kör funktionen och frigör sedan omedelbart resurserna när funktionen har körts färdigt. Man betalar bara för den beräkningstid man förbrukar. Alltså, det kostar inget när koden inte körs.

När det kommer till FaaS finns det två nyckelkoncept som heter “cold start” och “warm start” eller kallstart och varmstart på svenska [1] [21]. Dessa termer beskriver tillståndet för den serverlösa funktionsinstansen vid tidpunkten för anropet och kan avsevärt påverka prestandan och latensen för anrop och därmed påverka applikationen som är beroende av dessa funktioner.

2.2.1 Cold start

En “Cold start” eller kallstart inträffar när en funktion anropas efter att ha varit inaktiv ett tag. När en begäran görs till en funktion måste molnleverantören allokera resurser, initiera funktionen och sedan köra funktionen. Denna process kan lägga till latens till funktionens svarstid eftersom funktionen måste initieras helt innan den kan börja bearbeta begäran. Kallstarter inträffar vanligtvis när en funktion anropas för första gången eller om en funktion inte har anropats på länge, den specifika varaktigheten varierar beroende på molnleverantör.

2.2.2 Warm start

En “Warm start” eller varmstart inträffar när en funktion anropas medan dess resurser fortfarande är allokerade från en tidigare anrop. I det här fallet behöver molnleverantören inte gå igenom hela initieringsprocessen igen. Istället kan funktionen börja exekveras nästan omedelbart, vilket ger en mycket snabbare svarstid jämfört med en kallstart. Efter att en funktion har anropats håller molnleverantörer funktionen "varm"under en viss period, redo att snabbt svara på efterföljande anrop.

2.3 Benchmarking

I datorsammanhang är benchmarking en process där man jämför ett systems prestanda eller en uppsättning mätvärden mot standard-tester för att utvärdera systemets prestanda eller effektivitet [18]. Det handlar om att samla in data om hur systemet fungerar under specifika förhållanden och sedan analysera dessa data för att identifiera luckor och förbättringsområden. Andra anledningar till att man utför benchmarking är för att säkerställa att ett system kan uppfylla specifika prestandakrav och identifiera bästa praxis som kan användas för att förbättra ett system både kortsiktigt och långsiktigt.

2.3.1 Benchmark praxis

För att kunna utföra en benchmark måste man börja med att välja en produkt, service eller process för att benchmark:a och sedan överväga vad som kan vara lämpliga Key Performance Indicators (KPI:er) (nyckelprestandamått) [8]. Dessa är de mätvärden som kommer att användas för att jämföra prestanda. KPI:er kan vara relaterade till hastighet, noggrannhet, effektivitet, kvalitet etc. Efter man har definierat sina KPI:er, är det dags att samla in data och det görs genom att köra tester. När det gäller prestanda-testning av datorresurser i form av API:er så finns olika typer av tester som anses vara industri standard, nämligen “Load testing” och “Spike testing” [6]. Load testing (belastningstestning) handlar i första hand om att bedöma ett systems nuvarande prestanda i termer av parallella användare eller förfrågningar per sekund. Spike testing (spiktestning) görs genom att plötsligt öka eller minska belastningen som genereras av ett mycket stort antal användare, och observera systemets beteende. Målet är att avgöra om prestandan kommer att drabbas, om systemet kommer att misslyckas eller om det kommer att kunna hantera dramatiska förändringar i belastningen under en kort tid.

2.4 Microservice-arkitektur

En applikation byggd enligt microservice-arkitekturen är uppdelad i flera mindre, självständiga delar som kallas microservices. Dessa delar har endast ett tydligt syfte och kan i teorin driftsättas helt isolerade från de andra microservices. De kommunicerar genom att skicka meddelanden mellan de olika delarna vilket möjliggör att de körs på olika servrar och är byggda i olika programmeringsspråk.

3
Metod

Processen att analysera den bästa vägen för att överföra Xpektor till molnet, kommer kräva en kombination av onlineforskning, direkta insikter från molnleverantörer och feedback från produktägare, projektledare, utvecklare och användare. Detta avsnitt tydliggör den mångsidiga metoden som kommer att användas för att säkerställa en bra förståelse och omfattande bedömning av ämnet.

3.1 Informationsinsamling

Forskningsprocessen kommer att börja med en omfattande onlinesökning, främst driven av Google-sökningar för att identifiera olika molnleverantörer som finns på marknaden. Baserat på marknadsandelar och inflytande i branschen valdes tre leverantörer ut för vidare undersökning: Amazon Web Services, Microsoft Azure och Google Cloud Platform [15].

Genom att använda specifika sökord som "FaaS providers comparison", "cloud providers market share", "AWS vs Azure vs Google Cloud cost analysis", "Cloud provider benchmark", "cloud migration best practices" och "microservice cloud transition" kommer ett omfattande utbud av artiklar, blogginlägg och officiell dokumentation att granskas. Denna metod säkerställer en förståelse för både de teoretiska aspekterna och verkliga erfarenheter som delas av utvecklare och företag över hela världen.

För en detaljerad kostnadsanalys kommer officiell dokumentation och prissidor granskas för var och en av de tre stora FaaS-leverantörerna - AWS, Azure och Google Cloud. Utöver att söka bland artiklar och blogginlägg, kommer sökningar på Chalmers Biblitotek och Google Scholar att göras. Sökorden som kommer användas är "cloud migration", "FaaS benchmarking", och "cloud transition challenges". Sökresultatet filtreras därefter på de senaste 5 åren för att säkerställa relevans.

3.2 Intervjuer

För att säkerställa en sömlös övergång av Xpektor till molnet är en djup förståelse för dess behov och vilka problem de försöker lösa viktigt. Med tanke på de tekniska och operativa krångligheterna hos Xpektor är det pragmatiskt att utnyttja insikter från dem som är nära förknippade med dess utveckling, driftsättning och dagliga verksamhet. Både strukturerade och ostrukturerade intervjuer kommer att

organiseras med nyckelintressenter såsom projektledare, ledande utvecklare, systemarkitekter, driftteammedlemmar och även användarna. Dessa diskussioner kommer vara centrerade kring att förstå Xpektors funktionsspecifikationer, systemarkitektur och operativa utmaningar.

Exempel på frågor som kommer att ställas till de som jobbar på Xpektor är:

- Vilken var den primära visionen bakom skapandet av Xpektor och hur har denna vision utvecklats över tid?
- Vilken/vilka problem inom finansvärlden försöker Xpektor att lösa?
- Hur ser Xpektors nuvarande flöde ut?
- Kan du ge en översikt över den nuvarande hosting- och infrastrukturkonfigurationen?
- Hur interagerar de olika mikrotjänsterna med varandra?
- Finns det någon dokumentation om flödet och infrastrukturen?
- Vilka är de kända flaskhalsarna när det gäller prestanda?
- Vet du ungefär hur många funktionsanrop det görs dagligen?
- Hur skulle du beskriva kompetensnivån hos utvecklarna inför den kommande migrationen till cloud?
- Har utvecklarna erfarenhet av att utveckla en produkt genom att använda de inbyggda molnlösningarna som leverantören erbjuder?

Exempel på frågor som kommer att ställas till användarna är:

- På vilket/vilka sätt gör Xpektor ditt jobb lättare?
- Hur skulle du beskriva applikationens hastighet, särskilt under högtrafik?
- Har du stött på några fall där applikationen laggade eller inte svarade?
- Har du upplevt några driftstopp eller tekniska fel när du använder Xpektor? Om ja, hur ofta förekommer de och hur har de påverkat ditt arbetsflöde?
- Hur aktuella och exakta är e-postmeddelanden och varningar du får från Xpektor?
- På en skala från 1-10, hur nöjd är du med Xpektor och varför?
- Känner du till att Xpektor ska migrera till molnet? Isåfall vad har du för förväntningar?

3.3 Benchmarks

För att kunna utföra tester som kan ge någon värdefull insikt om prestandan hos molnleverantörerna, är första steget att designa och skapa funktioner på de olika molnleverantörerna. Sedan måste man välja sina KPI:er, skapa testing scenarios, utföra testerna och samla testdata.

3.3.1 Design

Det kommer skapas 3 olika funktioner i C# där varje funktion har en nivå. Ju högre nivå, desto mer komplicerad uppgift är det som kommer att utföras av funktionen.

1. Nivå 1 - “Hello world!”

   Syftet med funktionen är att utvärdera grundläggande svarstider och funktionsanrop. Denna funktion returnerar ett enkelt “Hello World”-meddelande.

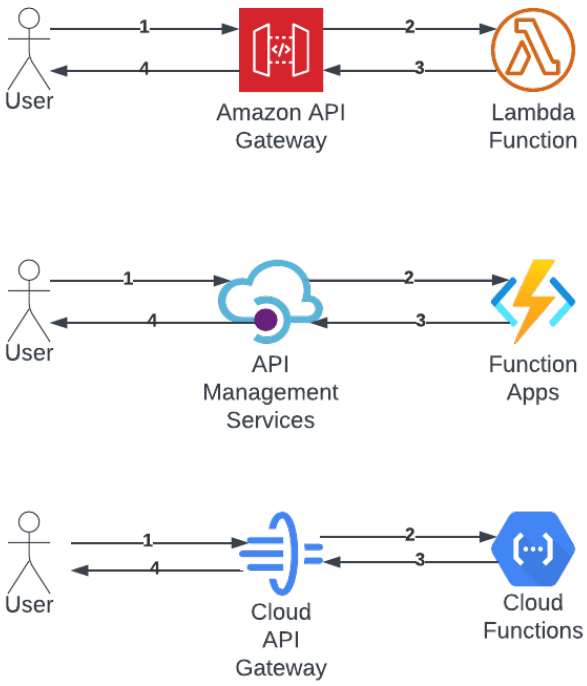
2. Nivå 2 - Sträng manipulation

   Syftet med denna funktion är att öka nivån från föregående funktion och utföra en faktisk operation på Requesten som har kommit in. Denna funktion tar in en textsträng och ger tillbaka samma textsträng med stora bokstäver.

3. Nivå 3 - Bildbehandling

   Syftet med denna funktion är att utföra en mer komplex operation med några beroenden än förra funktionen. Denna funktion tar in en bas64 bild och konverterar den till en 100x100 bild och svarar tillbaka med en bas64 respons.

Alla funktioner kommer vara kopplade till sina respektive molnleveratörers API Gateway.



Figur 3.1: Hur funktionerna är uppsatta i respektive molnleverantör.

Leverantörnamn	FaaS namn	API Gateway namn
AWS	Lambda Function	Amazon API Gateway
Azure	Azure Functions	API Management Services
Google Cloud	Cloud Functions	Cloud API Gateway

Tabell 3.1: Respektive molnleverantörs API Gateway tjänst och namn på FaaS tjänst.

3.3.2 KPI:er

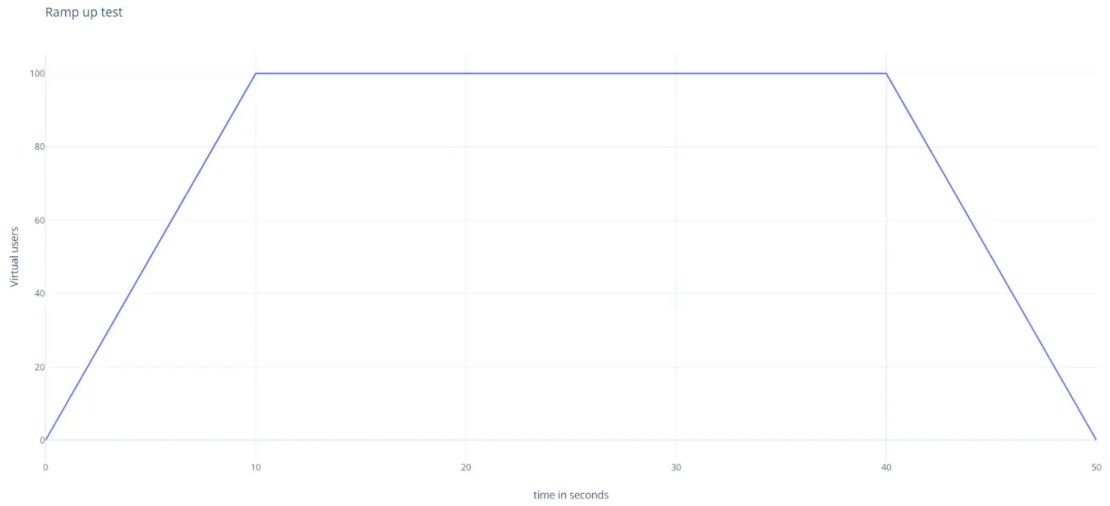
KPI:erna kommer vara baserade på prestanda och tillförlitlighet. När det gäller prestanda så är det mätningen av den genomsnittliga svarstiden för funktionen som är aktuell. Tillförlitlighet handlar om att funktionen fungerar och inte kastar slumpmässiga fel. Isåfall dokumentera eventuella fel, försöka ta reda på varför dessa fel uppstår.

3.3.3 Datainsamling från benchmarks

För att utföra tester kommer ett verktyg som heter K6 utnyttjas [2]. K6 är ett välkänt verktyg med öppen källkod som tillåter utvecklare att skapa skripter som gör anrop mot önskat API och efterliknar användarbeteenden. Resultat av dessa anrop kan sparas sedan antingen i CSV format eller JSON format; CSV kommer att väljas för denna studie. För att rita grafer av CSV-filerna kommer ett Python-skript skrivas med hjälp av Pandas [4] och Matplotlib [3]. Pandas är ett bibliotek som möjliggör läsning av CSV-filer och Matplotlib är ett bibliotek för att rita grafer.

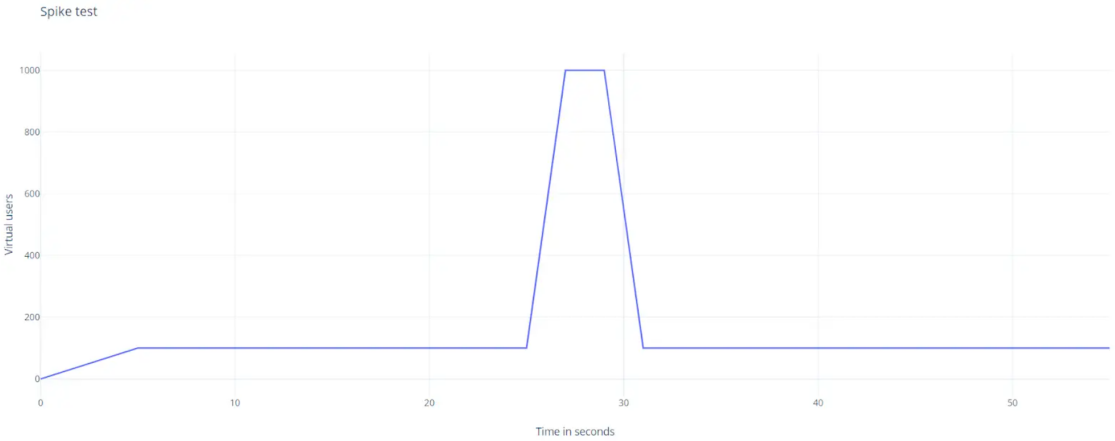
3.3.4 Testning scenario

Med hjälp av K6 kommer ett “Ramp up” loading test och ett spiktest att skapas. I Ramp up-testen ökar belastningen statiskt under en önskad tid till det fördefinierade antal virtuella användare som gör ett anrop per sekund. Sedan håller funktionen antal virtuella användare konstant tills belastningen går ner igen.



Figur 3.2: Visuell representation av ramp up test, y-axeln är antal virtuella användare, x-axeln är tid i sekunder.

I spiktesten, finns det plötsliga ökade antal användare som går ner lika fort. Detta kan hända flera gånger.



Figur 3.3: Visuell representation av spiktesten, y-axeln är antal virtuella användare, x-axeln är tid i sekunder.

3.4 Kostnadsanalys

3.4.1 Direkta kostnader

För att kunna göra en kostnadsanalys behöver det göras ett estimat på hur många funktionsanrop som Xpektor kommer att göra och hur länge dessa funktioner kommer att köras när de flyttas till cloud. Detta estimat baseras på nuvarande antal funktionsanrop som görs mot Xpektors API och antal anrop som görs mot tredjeparts API:er. För att kunna estimera hur många funktionsanrop det görs och hur länge dessa funktioner körs kommer det behövas en detaljerad analys av kodbasen

samt en rigorös genomgång av loggfiler. Detta estimat kommer sedan diskuterades med utvecklarna och arkitekten hos Xpektor för att kontrollera om det är en rimlig siffra. När alla parterna är överens om rimligheten av estimatet kommer arbetet för kostnadsanalysen att börja. Denna analys kommer göras med de kalkylatorer som erbjuds av molnleverantörerna, nämligen AWS Pricing Calculator[1], Azure Pricing calculator[2] och Google Cloud Pricing Calculator[3]. Där kan man välja vilka tjänster man kommer att använda, vilken region tjänsten kommer att distribueras till samt hur mycket av dessa tjänster som kommer att användas. Kalkylatorn beräknar sedan alla kostnader och redovisar det för användaren.

3.4.2 Indirekta kostnader

De indirekta kostnaderna handlar mest om personalutbildning men även support och ledning, potentiella driftstopp. På grund av komplexiteten för alla dessa variabler, kommer räkningen av de indirekta kostnaderna delegeras till produktägarna och ekonomerna.

[1]https://calculator.aws
[2]https://azure.microsoft.com/en-us/pricing/calculator/
[3]https://cloud.google.com/products/calculator

4
Resultat

I följande kapitel kommer resultaten från intervjuerna, benchmarks samt kostnadsanalysen att redovisas.

4.1 Intervju resultat

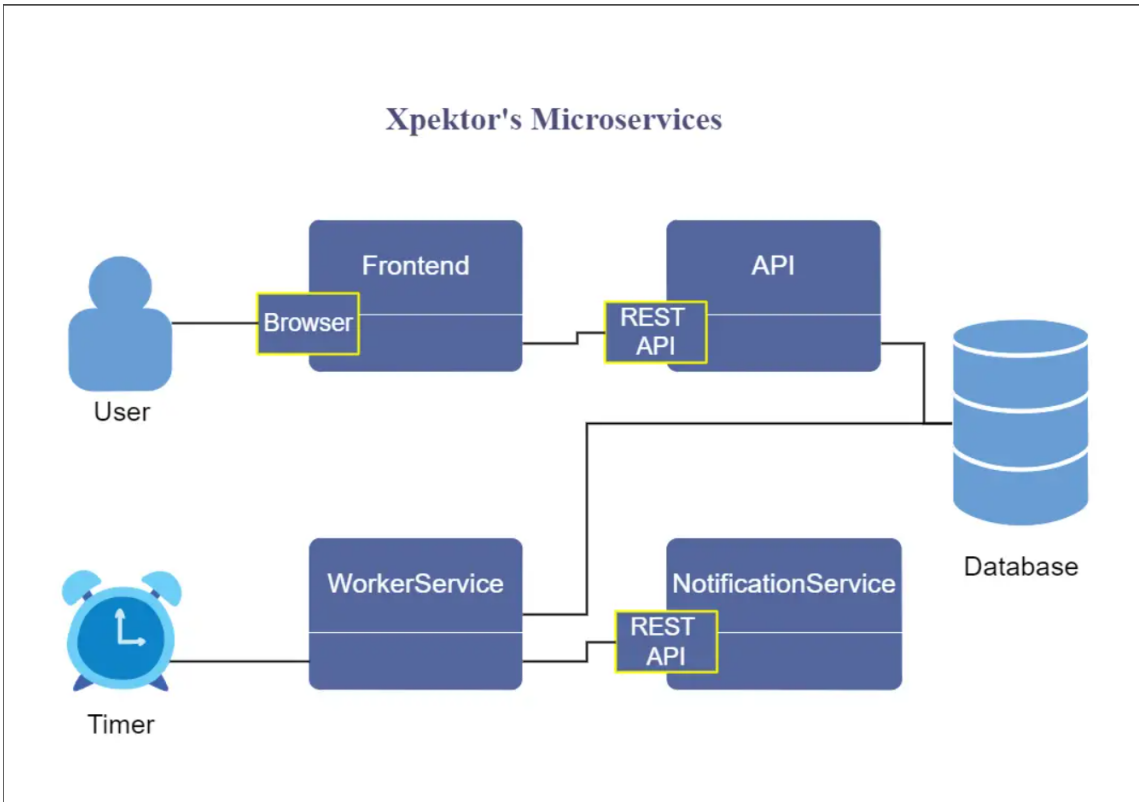
Xpektor började med visionen att med sin applikation föra vidare digitaliseringen av redovisningsbranschen genom att automatisera och tillhandahålla rapporter i realtid som används som underlag för rådgivning av redovisningsekonomer. Allt eftersom produkten byggdes upp insåg Xpektor att dessa rapporter i realtid kan användas utöver redovisningsbranschen. Med detta så har Xpektors vision utvecklats till att även bli en form av kreditupplysning som kan användas av finansbolag eller banker vid köp av fakturor.

4.1.1 Xpektor i sin nuvarande form

Xpektors server körs för närvarande på ett traditionellt webbhotell. Denna plattform tillhandahåller en virtualiserad Linux-miljö. Den virtuella maskinen har en lagringskapacitet på 50GB och kostar runt 24000 kr per år för Xpektor.

Applikationens backend är skriven i programmeringsspråket C# [10] med hjälp av ramverket .NET [11] och har en SQL databas [7] för lagring av information. Frontend-delen är skriven i programmeringsspråket Javascript [12] med hjälp av ramverket React och kommunicerar med backend via applikationens API. Utöver dessa två delar har applikationen ytterligare två delar som kallas “WorkerService” och “NotificationService”.

Xpektor implementerar en microservice-arkitektur genom att dela upp backenddelen i tre microservices. Dessa är API, WorkerService samt NotificationService. API:t ansvarar för kommunikation mellan backend och frontend och möjliggör funktioner som att logga in och hämta uppdaterade värden som presenteras för användaren. WorkerService delen körs kontinuerligt i bakgrunden på servern och gör beräkningar för att uppdatera databasen med aktuella värden. NotificationService körs däremot endast vid begäran och hanterar utskick av notifikationer till användarna. I figur 4.1 illustreras de olika delarna och vilka kopplingar dessa har mellan varandra. Som visat i figuren är både WorkerService och NotificationService helt bortkopplade från användarens upplevelse av applikationen.



Figur 4.1: Diagram av applikationens microservices där kopplingen mellan de olika delarna visas.

4.1.2 Applikationens behov i dag och framöver

Centralt för Xpektors design är dess Microservice-arkitektur. Detta designval kräver en infrastruktur som sömlöst stöder kommunikation mellan tjänster. Som komplement till detta är Xpektor beroende av en SQL-databas, som kräver högpresterande databasservrar, vilket säkerställer att systemet kan hantera transaktioner under perioder med högre trafik. Regelbundna säkerhetskopieringar och omedelbara skalningsmöjligheter är också avgörande med tanke på volymen och vikten av data som bearbetas.

Xpektors operativa kärna definieras av dess integration med olika redovisningssystem. Stabila, snabba och säkra integrationer är avgörande för att säkerställa att datahämtning sker korrekt och i rätt tid. Utöver detta är det viktigt att WorkerService:en, som körs kontinuerligt, har tillgång till robusta beräkningsresurser för att säkerställa att kunderna får den finansiella rapporter som de har lovats. Lika viktig är NotificationService, som måste vara utrustad för att hantera stora volymer av meddelanden, för att säkerställa att användarna omedelbart uppmärksammas på viktiga ekonomiska insikter.

Prestanda och skalbarhet är två pelare som Xpektors framgång beror på. Eftersom det handlar om finansiell data i realtid måste applikationen konsekvent leverera data med godtagbara responstider, även under perioder med hög användaraktivitet.

Dessutom, när Xpektor fortsätter sin uppåtgående bana och samlar fler användare, måste den underliggande infrastrukturen vara förberedd för ökad belastning, särskilt under finansiella toppar som bokslut och årsredovisningar.

Utöver dessa två pelare är säkerhet det absolut viktigaste med tanke på den känsliga karaktären hos finansiell data. Krypteringsstandarder måste tillämpas både när data överförs och när den lagras. Eftersom Xpektor har en mångsidig användarbas, som inkluderar styrelseledamöter, revisorer och ekonomer, krävs det en åtkomstkontrollmekanism som säkerställer att varje användare har lämpliga behörigheter.

För att möjliggöra en bra användarupplevelse är det absolut nödvändigt att frontend:en levererar data med låg latens och är mobilanpassad.

Slutligen har intervjuerna gett en estimering av volymen av användartrafiken för att användas i testningssyfte. I dagsläget görs cirka 170 000 anrop med ett genomsnitt på 1 sekund körtid per dag. Med deras prognos för framtiden förväntas Xpektor hantera upp till 1 000 000 anrop med ett genomsnitt på 1 sekund körtid om dagen.

4.1.3 Utvecklarnas kompetens

Från intervjun framgick det att cirka hälften av utvecklarna har erfarenhet av att utveckla i molnmiljöer, mer specifikt Azure.

4.2 Benchmark resultat

I följande delkapitel redovisas resultaten av benchmarken i form av grafer och tabeller.

4.2.1 Load test

Följande figurer visar resultaten från Load testerna som gjordes mot respektive molnleverantör. Den blåa linjen representerar antal anrop per sekund och den röda linjen representerar antal misslyckade anrop per sekund.

4.2.1.1 Nivå 1

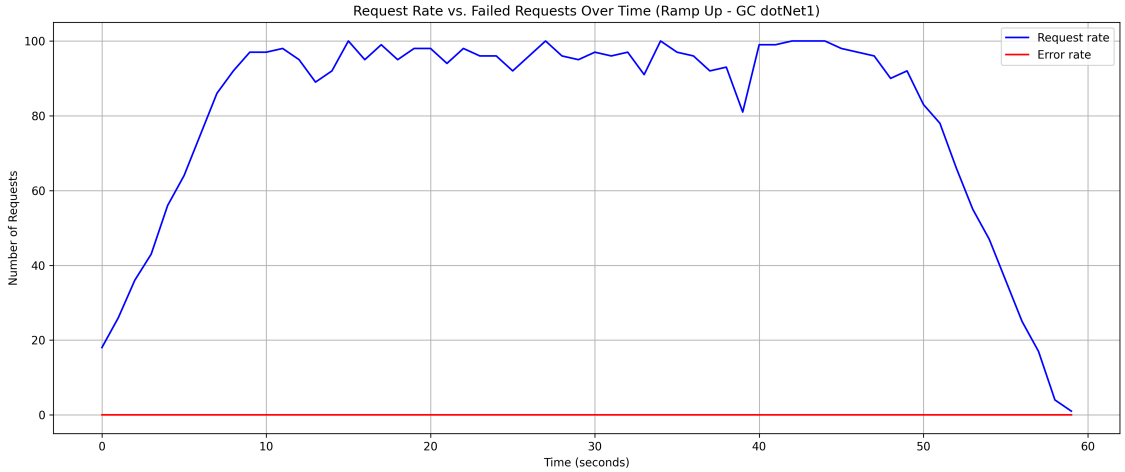
I figuren nedan visas resultatet av Load testen mot funktionen med nivå 1 i AWS. Antal anrop ökar i samband med ökad antal vituella användare. Efter sekund 10, börjar anropen misslyckas och detta ökar med tiden till slutet när antal vituella användare minskar.

Figur 4.2: Resultat av Load testen av AWS nivå 1.

I figuren nedan visas resultatet av Load testen mot funktionen med nivå 1 i Azure. Här ser man att Azure funktionen inte kastar några fel som AWS funktionen.

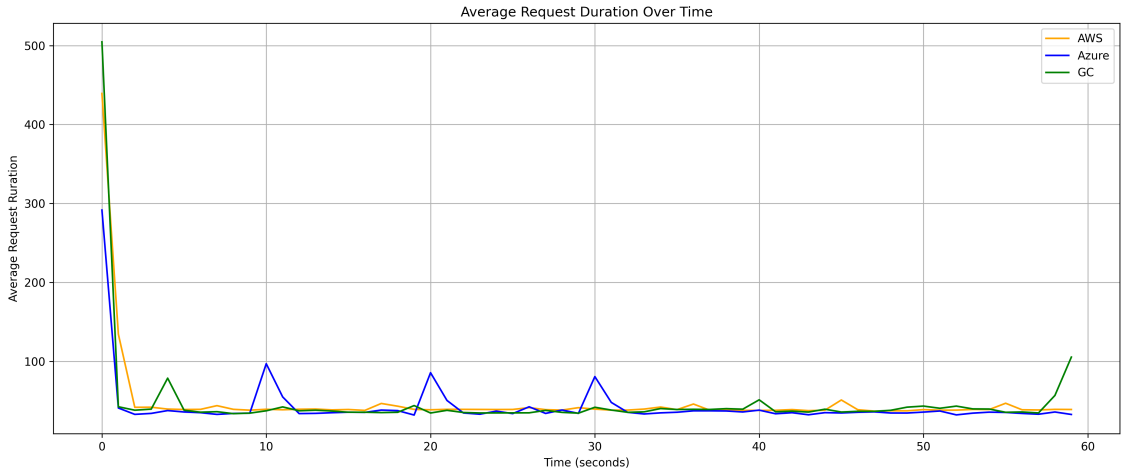
Figur 4.3: Resultat av Load testen av Azure nivå 1.

I figuren nedan visas resultatet av Load testen mot funktionen med nivå 1 i GC. Precis som Azure funktionen, kastar inte denna funktion några fel.



Figur 4.4: Resultat av Load testen av Google Cloud nivå 1.

I figuren nedan jämförs den genomsnittliga responstiden för varje molnleverantör. Azure har lägst kallstart men har spikar ungefär var tionde sekund tills sekund 30. GC har högst kallstart och spikar lite i början med dalar av efter den första spiken dock ökar responstiden mot slutet. AWS har en kallstart som är mellan Azures och GC:s kallstart och har minimala spikar.



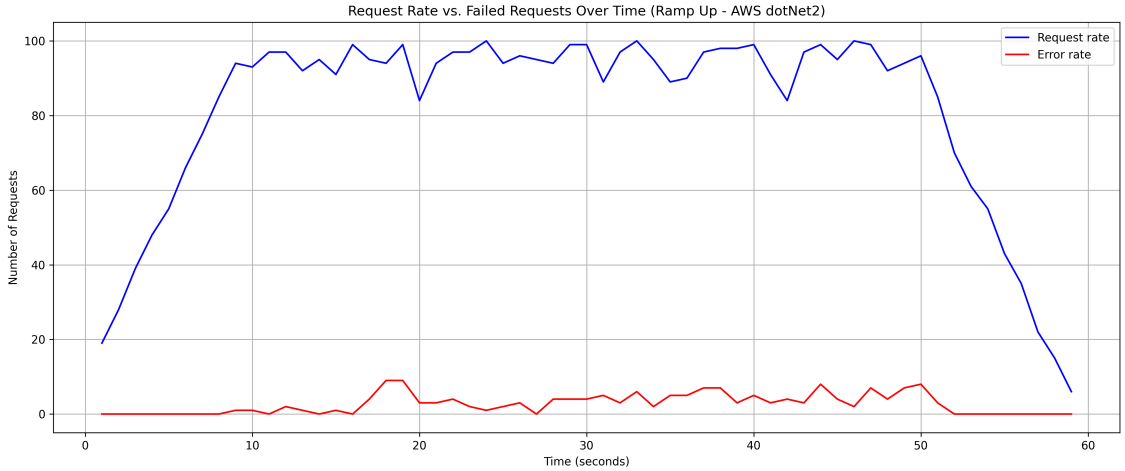
Figur 4.5: Genomsnittlig responstid för varje molnleverantör, rampup nivå 1.

Name	Tot. num. of reqs	Tot. num. of errors	Error Rate (%)	Highest Resp. Time (ms)	Lowest Resp. Time (ms)	Avg. Resp. Time (ms)
AWS	4827	238	4.93	791.03	18.32	40.91
Azure	4835	0	0	718.75	17.53	39.83
GC	4835	0	0	1150.80	17.55	40.15

Tabell 4.1: Load test resultatet för nivå 1

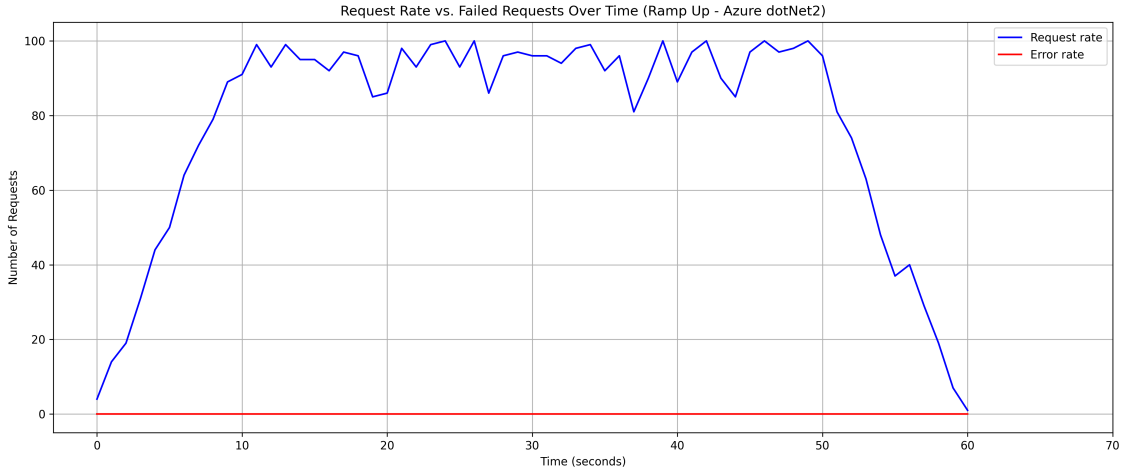
4.2.1.2 Nivå 2

I figuren nedan visas resultatet av Load testen mot funktionen med nivå 2 i AWS. Antal anrop ökar i samband med ökad antal vituella användare. Denna gången börjar anropen misslyckas innan sekund 10 och ökar med tiden till slutet när antal vituella användare minskar.



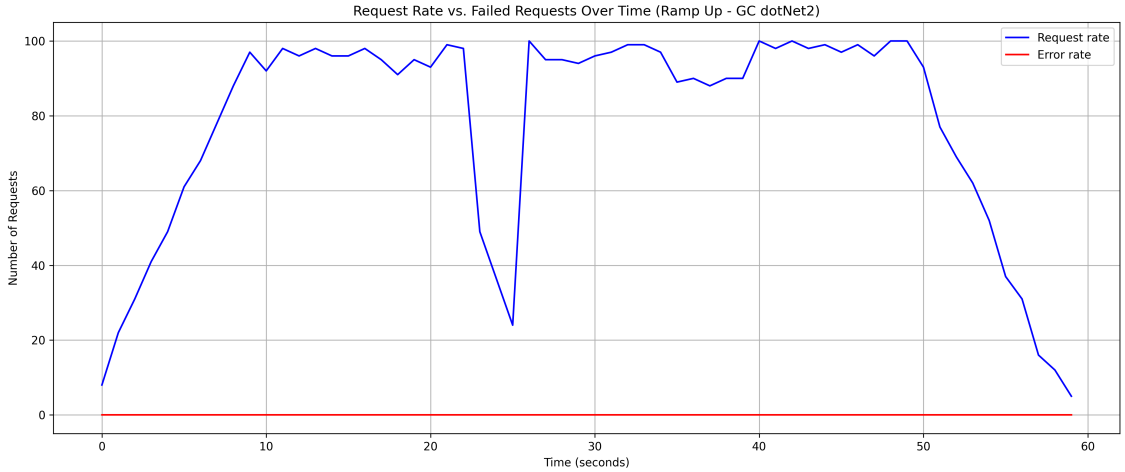
Figur 4.6: Resultat av Load testen av AWS nivå 2.

I figuren nedan visas resultatet av Load testen mot funktionen med nivå 2 i Azure. Här ser man att Azure funktionen, precis som innan, inte kastar några fel som AWS funktionen.



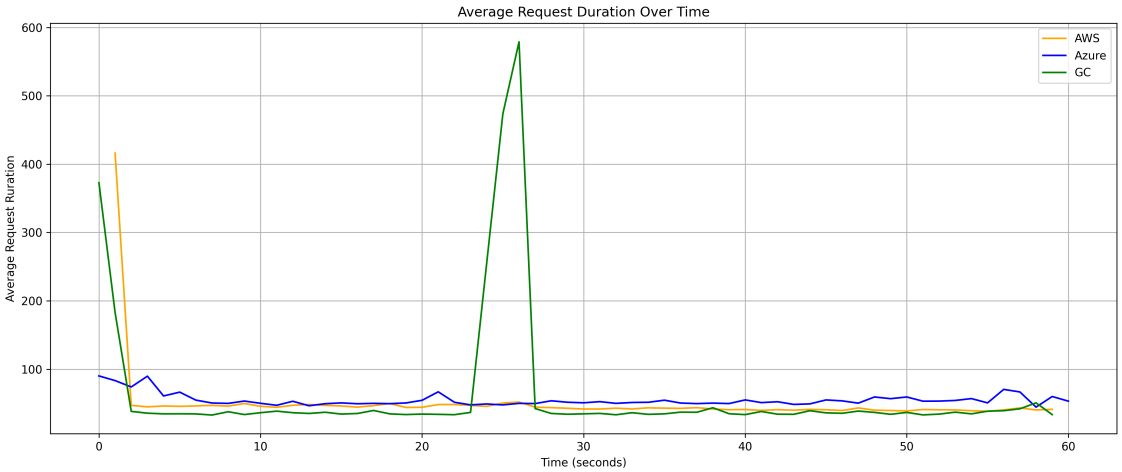
Figur 4.7: Resultat av Load testen av Azure nivå 2.

I figuren nedan visas resultatet av Load testen mot funktionen med nivå 2 i GC. Funktionen kastar inte några fel men vi ser att antal anrop per sekund minskas drastiskt vid sekund 25 och detta sammanfaller med den ökade responstiden som visas i figur 4.9. Anledningen till det minskade antal anrop är att de virtuella användare väntar på ett svar och därmed inte gör nya anrop.



Figur 4.8: Resultat av Load testen av Google Cloud nivå 2.

I figuren nedan jämförs den genomsnittliga responstiden för varje molnleverantör. Azure har återigen lägst kallstart. AWS har högst kallstart och GC har en kallstart som är lite mindre än AWS.



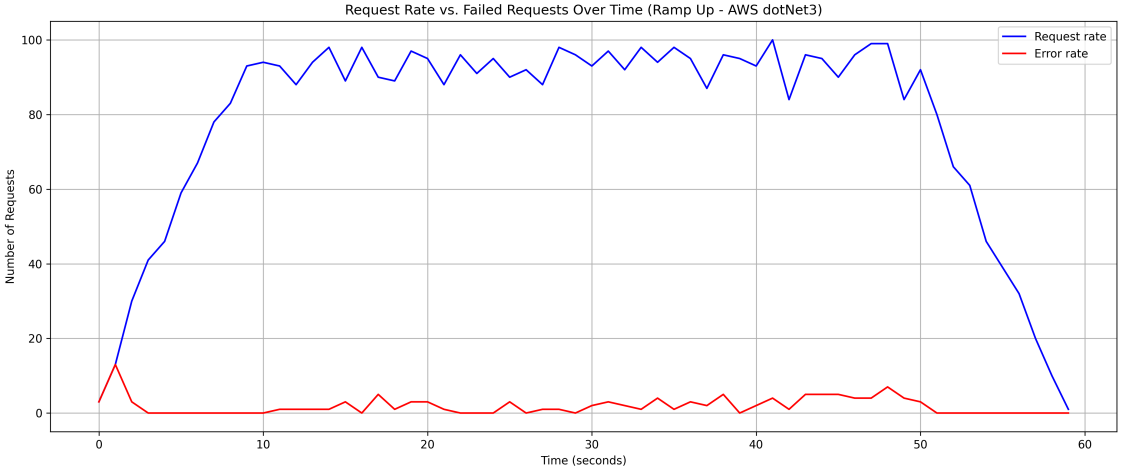
Figur 4.9: Genomsnittlig svarstid för varje tjänst, rampup nivå 2.

Name	Tot. num. of reqs	Tot. num. of errors	Error Rate (%)	Highest Resp. Time (ms)	Lowest Resp. Time (ms)	Avg. Resp. Time (ms)
AWS	4801	159	3.3	856.31	18.96	45.30
Azure	4746	0	0	160.79	23.07	52.78
GC	4621	0	0	2263.45	17.78	51.15

Tabell 4.2: Load test resultatet för nivå 2

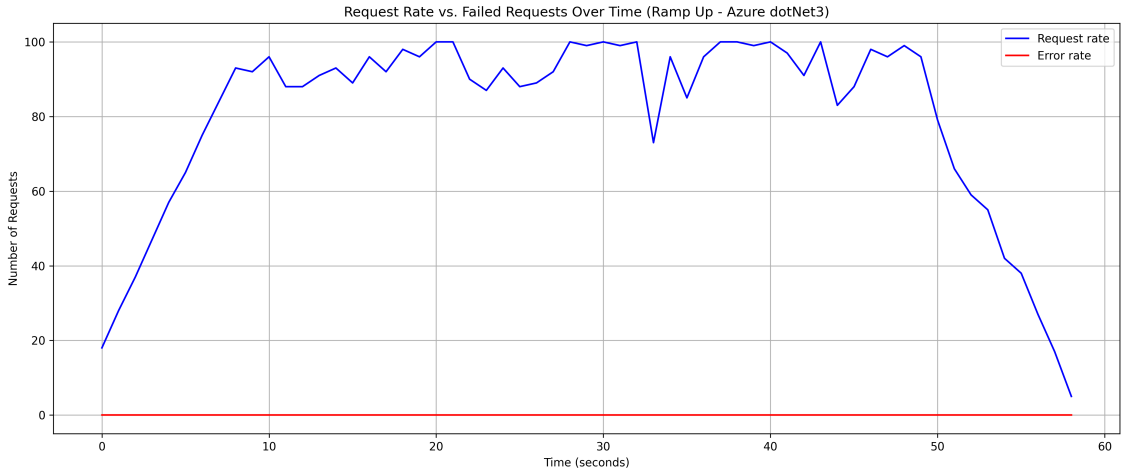
4.2.1.3 Nivå 3

I figuren nedan visas resultatet av Load testen mot funktionen med nivå 3 i AWS. Redan början så kastar funktione fel, detta dalar av en stund tills sekund 10 och då börjar funktionen kasta fel igen.



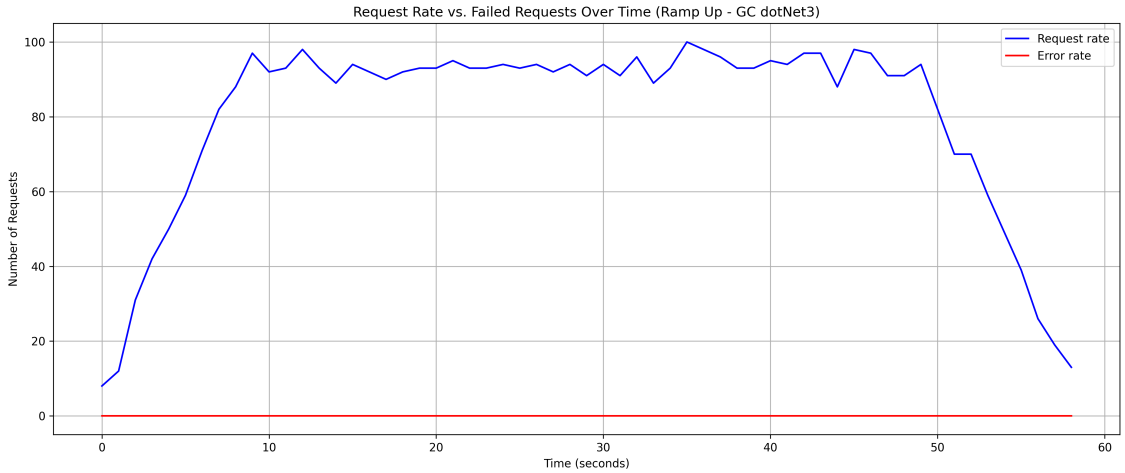
Figur 4.10: Resultat av Load testen av AWS nivå 3.

I figuren nedan visas resultatet av Load testen mot funktionen med nivå 3 i Azure. Inga fel denna gång heller.



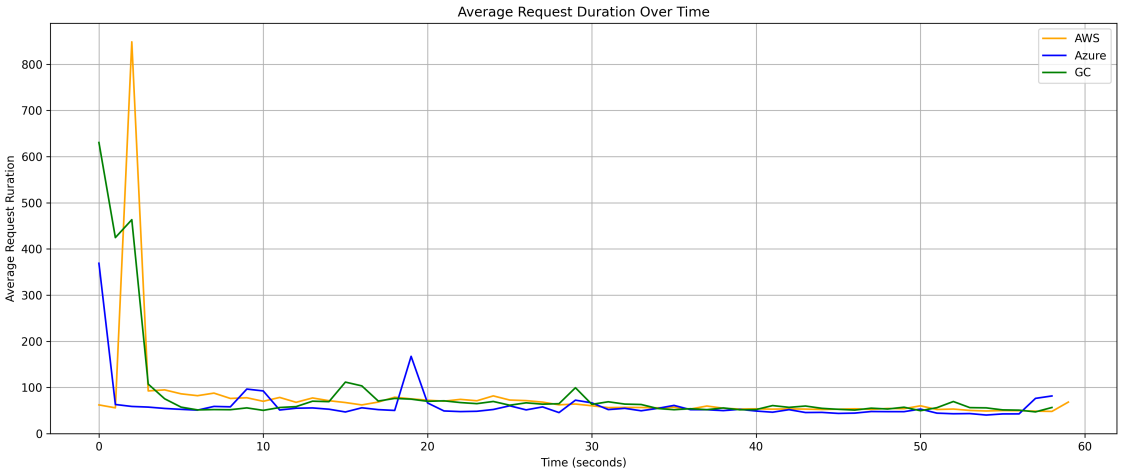
Figur 4.11: Resultat av Load testen av Azure nivå 3.

I figuren nedan visas resultatet av Load testen mot funktionen med nivå 3 i GC. Funktionen kastar inte några fel.



Figur 4.12: Resultat av Load testen av Google Cloud nivå 3.

I figuren nedan jämförs den genomsnittliga responstiden för varje molnleverantör. I figuren verkar det som att det är AWS som har lägst kallstart men detta är på grund av att AWS kastar fel i början. Den sanna kallstarts tiden syns en stund efteråt. Med det sagt så har Azure, återigen, lägst kallstart och GC kommer på plats två.



Figur 4.13: Genomsnittlig svarstid för varje tjänst, rampup test nivå 3.

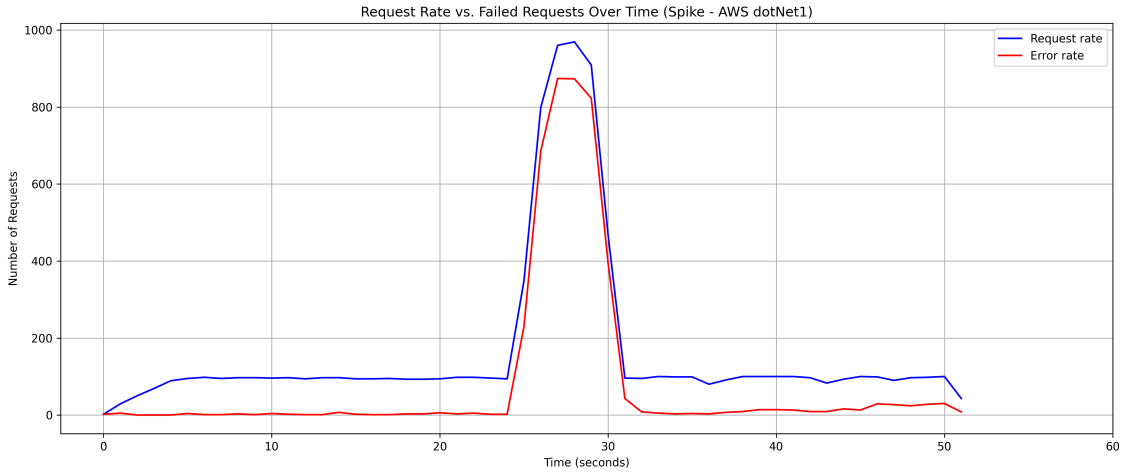
Name	Tot. num. of reqs	Tot. num. of errors	Error Rate (%)	Highest Resp. Time (ms)	Lowest Resp. Time (ms)	Avg. Resp. Time (ms)
AWS	4700	111	2.4	2511.88	26.28	69.18
Azure	4745	0	0%	1310.24	19.34	57.55
GC	4710	0	0	3291.19	37.69	67.65

Tabell 4.3: Load test resultatet för nivå 3

4.2.2 Spike test

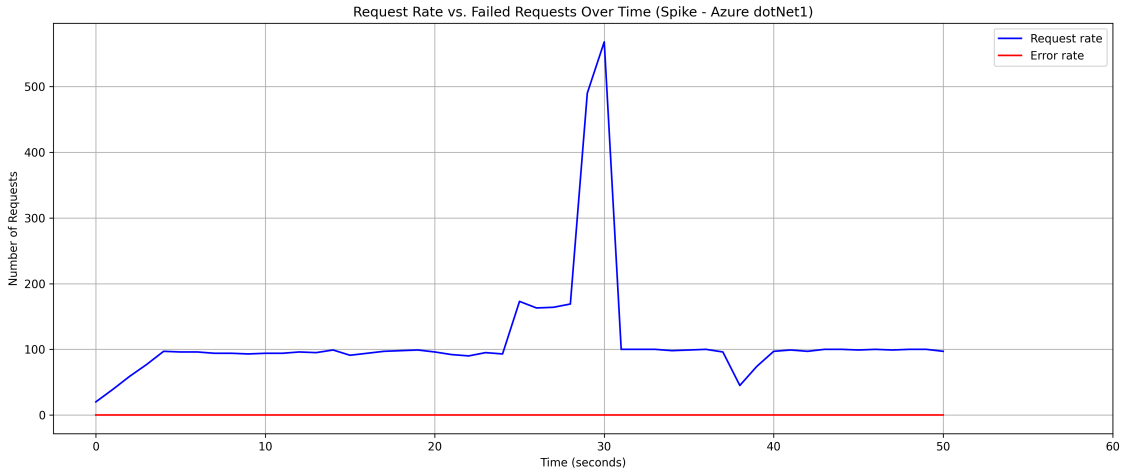
4.2.2.1 Nivå 1

I figuren nedan visas resultatet av Spike testen mot funktionen med nivå 1 i AWS. När antalet virtuella användare spikar, så börjar funktionen kasta fel för nästan varje anrop. Detta fortsätter tills antalet virtuella användare minskar igen.



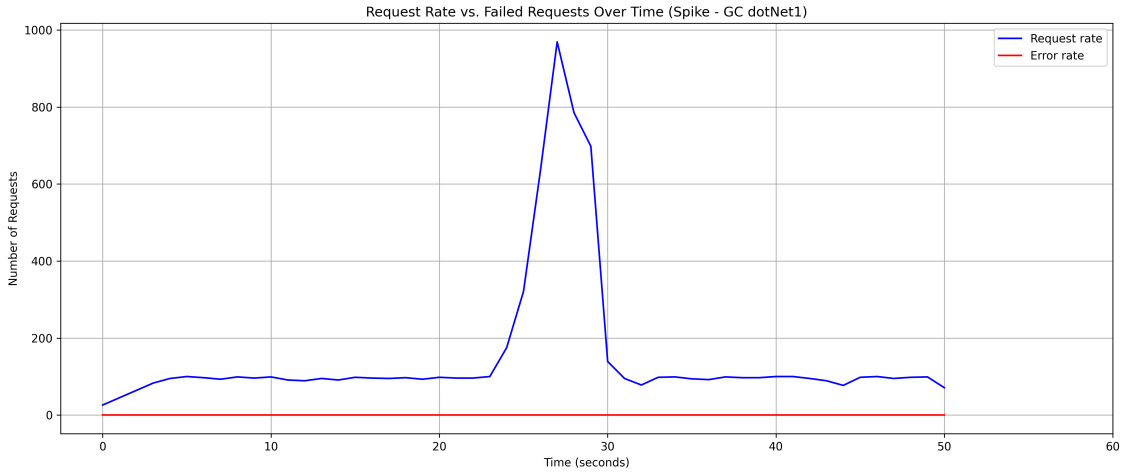
Figur 4.14: Resultat av Spike testen av AWS nivå 1.

I figuren nedan visas resultatet av Spike testen mot funktionen med nivå 1 i Azure. Azure hanterar det ökade antalet virtuella användare betydligt mycket bättre än AWS och kastar inga fel.



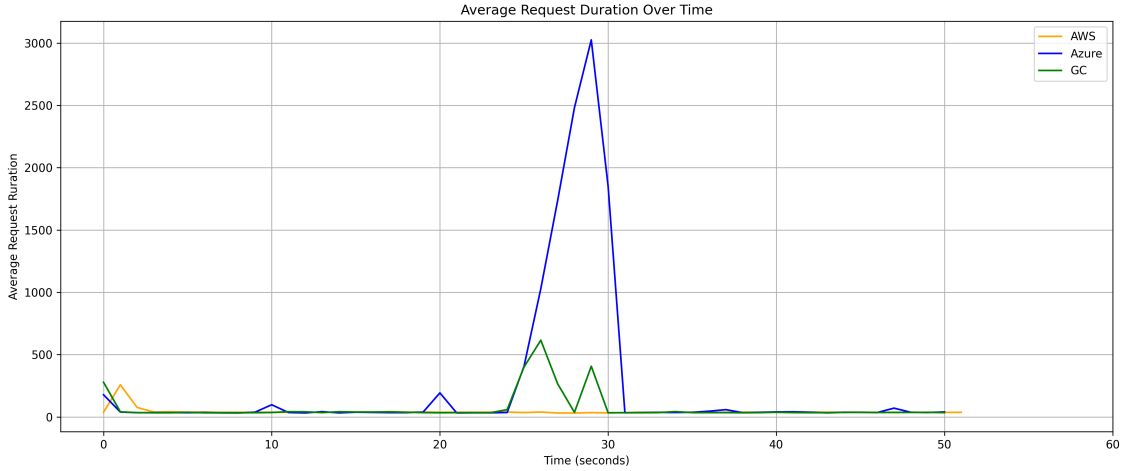
Figur 4.15: Resultat av Spike testen av Azure nivå 1.

I figuren nedan visas resultatet av Spike testen mot funktionen med nivå 1 i GC. Precis som Azure funktionen kastar den inga fel och kan även hantera högre antal anrop.



Figur 4.16: Resultat av Spike testen av Google Cloud nivå 1.

I figuren nedan jämförs den genomsnittliga responstiden för varje molnleverantör. AWS har återigen en falsk kallstartstid och därför är det Azure som har en lägsta kallstarten. Dock ser vi här att Azure funktionen kämpar har mycket högre responstid när antalet virtuella användare spikar. GC hanterar detta mycket bättre vilket leder till att den kan hantera fler anrop jämfört med Azure. AWS funktionen spikar inte i responstid eftersom den kastar fel på alla anrop som komer in när antalet virtuella användare ökar.



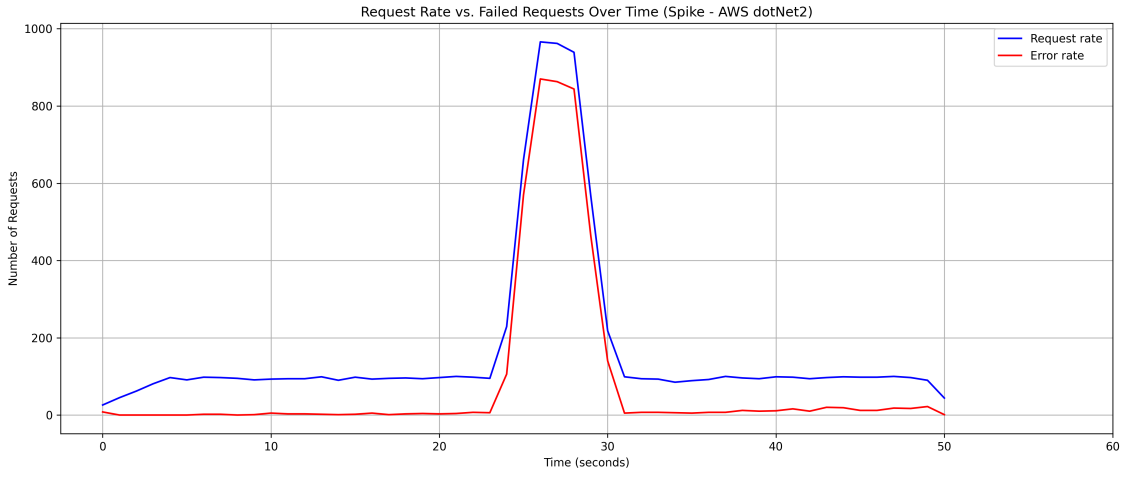
Figur 4.17: Genomsnittlig svarstid för varje tjänst, spike test nivå 1.

Name	Tot. num. of reqs	Tot. num. of errors	Error Rate (%)	Highest Resp. Time (ms)	Lowest Resp. Time (ms)	Avg. Resp. Time (ms)
AWS	8565	4257	49.70	1775.20	18.45	37.23
Azure	5815	0	0	3604.01	15.69	628.69
GC	7723	0	0	1249.48	16.89	162.79

Tabell 4.4: Spike test resultatet för nivå 1

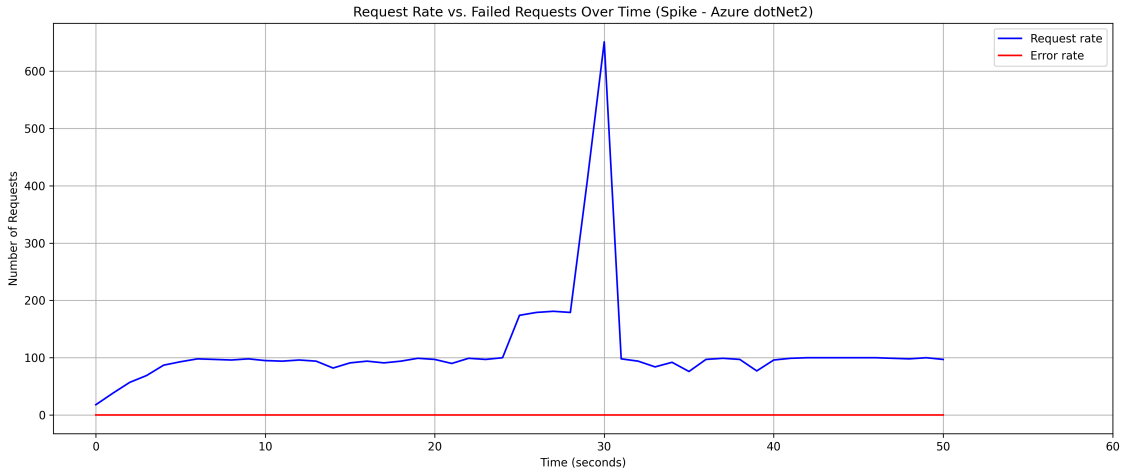
4.2.2.2 Nivå 2

I figuren nedan visas resultatet av Spike testen mot funktionen med nivå 2 i AWS. Vi ser en liknande beteende som tidigare.



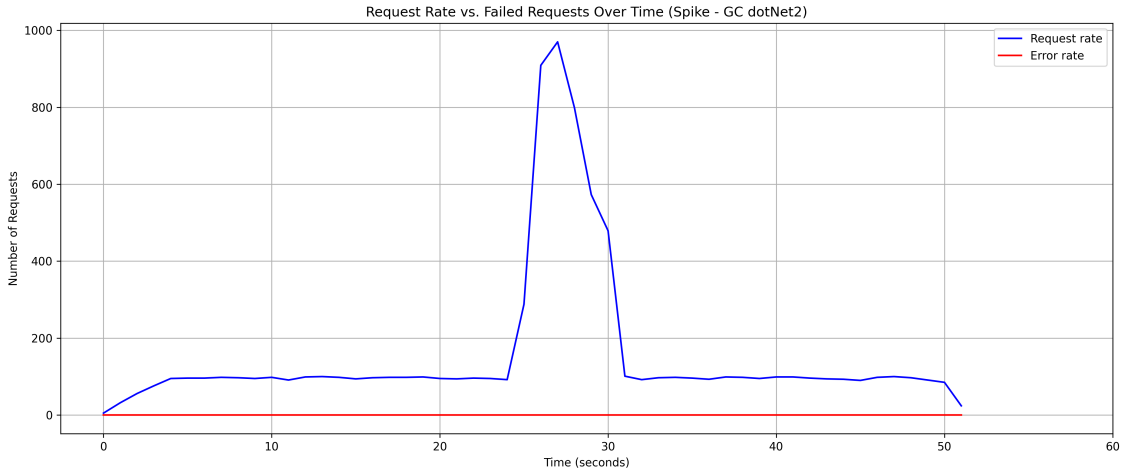
Figur 4.18: Resultat av Spike testen av AWS nivå 2.

I figuren nedan visas resultatet av Spike testen mot funktionen med nivå 2 i Azure. Alla anrop lyckas.



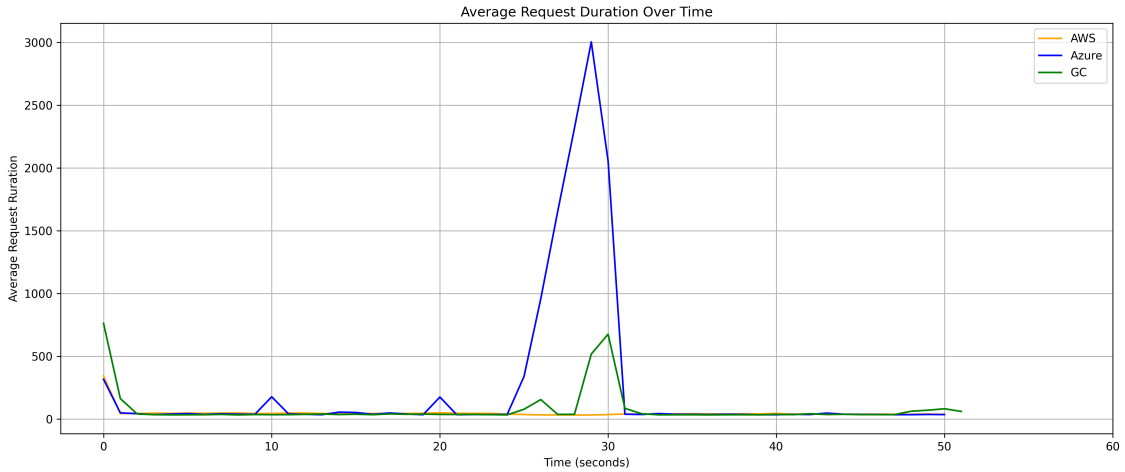
Figur 4.19: Resultat av Spike testen av Azure nivå 2.

I figuren nedan visas resultatet av Spike testen mot funktionen med nivå 2 i GC. Alla anrop lyckas här också.



Figur 4.20: Resultat av Spike testen av Google Cloud nivå 2.

I figuren nedan jämförs den genomsnittliga responstiden för varje molnleverantör. Vi ser en liknande beteende som föregående jämförelse.



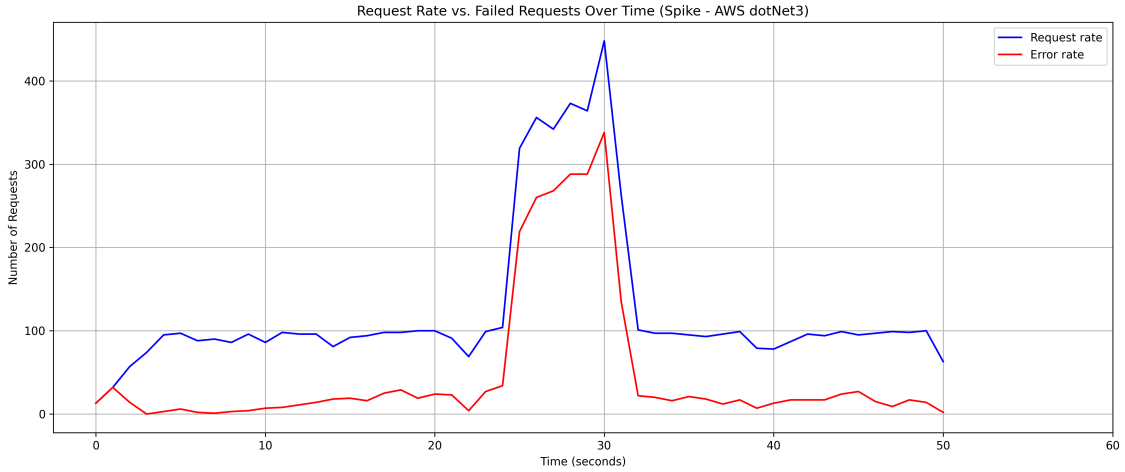
Figur 4.21: Genomsnittlig svarstid för varje tjänst, spike test nivå 2.

Name	Tot. num. of reqs	Tot. num. of errors	Error Rate (%)	Highest Resp. Time (ms)	Lowest Resp. Time (ms)	Avg. Resp. Time (ms)
AWS	8516	4142	48.64	873.75	17.05	38.73
Azure	5840	0	0	3763.99	15.14	633.57
GC	8143	0	0	1390.17	17.25	125.27

Tabell 4.5: Spike test resultatet för nivå 2

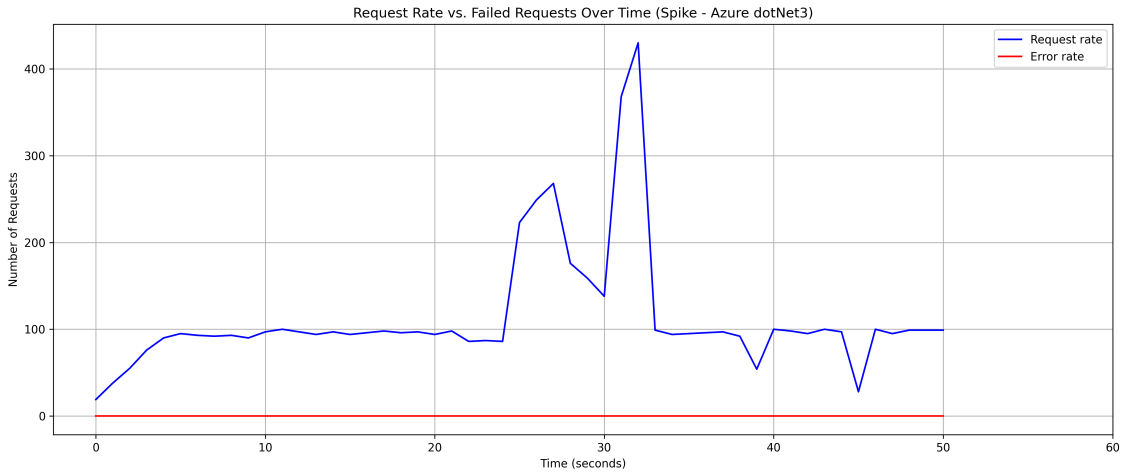
4.2.2.3 Nivå 3

I figuren nedan visas resultatet av Spike testen mot funktionen med nivå 3 i AWS. Liknande beteende som innan.



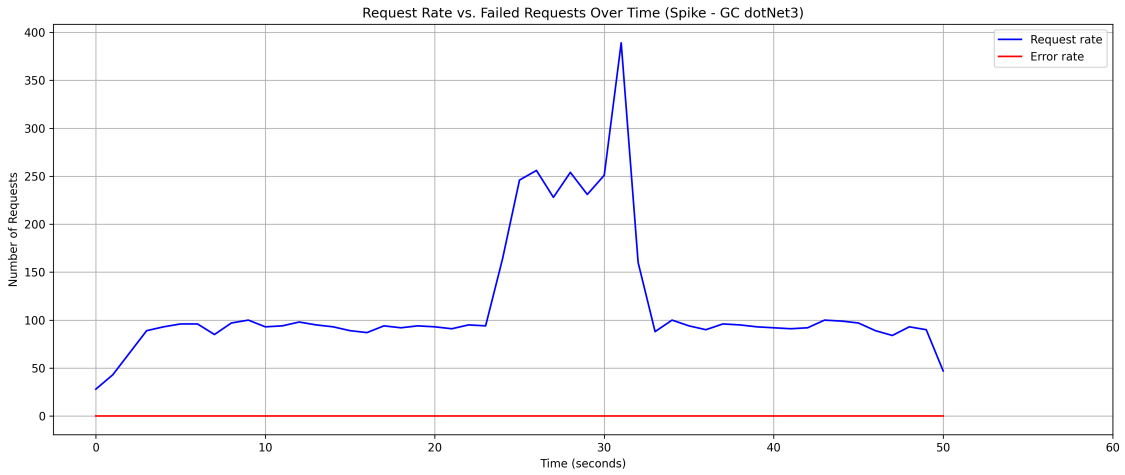
Figur 4.22: Resultat av Spike testen av AWS nivå 3.

I figuren nedan visas resultatet av Spike testen mot funktionen med nivå 3 i Azure. I denna test ser vi att det blir inga fel, dock klarar inte Azure funktionen av lika många anrop per sekund och därför ser vi att den dala i mitten.



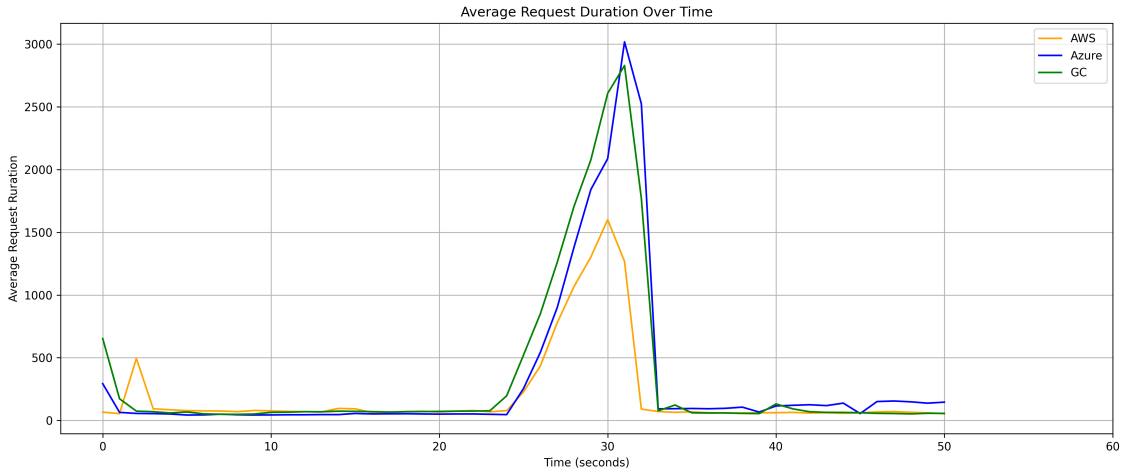
Figur 4.23: Resultat av Spike testen av Azure nivå 3.

I figuren nedan visas resultatet av Spike testen mot funktionen med nivå 3 i GC. Inga fel här heller men nu presterar GC funktionen nästan som Azure funktionen då denna test är lite tyngre.



Figur 4.24: Resultat av Spike testen av Google Cloud nivå 3.

I figuren nedan jämförs den genomsnittliga responstiden för varje molnleverantör. Azure funktionen har lägst kallstart eftersom det är en falsk kallstart från AWS igen. I denna figure ser vi att responstiden är nästan samma för både GC och Azure.



Figur 4.25: Genomsnittlig svarstid för varje tjänst, spike test nivå 3.

Name	Tot. num. of reqs	Tot. num. of errors	Error Rate (%)	Highest Resp. Time (ms)	Lowest Resp. Time (ms)	Avg. Resp. Time (ms)
AWS	6358	2457	38.64	3101.07	24.26	427.14
Azure	5806	0	0	4853.95	21.32	647.36
GC	5904	0	0	5591.68	32.88	659.43

Tabell 4.6: Spike test resultatet för nivå 3

4.3 Kostnadsanalys resultat

I följande delkapitel framgår kostnaden för varje molntjänst i form av en tabell baserat på 1 000 000 per dag med en genomsnittlig körtid på 1 sekund per anrop. Utöver detta användes ett allokerat minne på 128 MB för varje instans samt valt den närmaste regionen till Sverige.

Leverantör	Pris/månad (USD)
AWS	61.63
Azure	59.40
GC	81.78

Tabell 4.7: Tabell över kostnaderna för de olika leverantörerna.

5
Diskussion

I följande kapitel diskuteras resultaten från intervjuerna och benchmarken samt bristerna med testerna. Utöver detta diskuteras vad bästa lösningen för Xpektor är.

5.1 Diskussion om benchmark resultaten

Resultatet från benchmarken när det gäller kallstart var som förväntat. Som det syns i graferna hade alla leverantörer en rimlig kallstart där AWS och Azure hade en likvärdig fördröjning vid kallstart och var snabbare än GC i de flesta scenarion.

Vid spiktesten så hanterade GC det bäst med 0% fel och lägst genomsnittlig responstid. Vidare hade AWS bättre responstid än Azure. Det som särskiljer var att AWS Lambda var den enda FaaS tjänsten som producerade fel med upp till 50% av anropen vid en av spiktesterna. Detta beror på att AWS har ett förinställt tak på antal parallella instanser som kan ökas vid begäran [5]. När alla instanser är upptagna så får anropen 503 HTTP respons kod, vilket kan tolkas som att servern inte kan hantera anropen eftersom att den är för upptagen.

Överlag hade Azure lägre fördröjning vid kallstarten jämfört med AWS och GC, dock hade Azure högst responstid vid toppen av spike testerna där responstiden kunde gå hela vägen upp till 3 sekunder medan de andra leverantörerna höll sig under 1 sekund. Det enda scenariot där detta inte stämde var spike test nivå 3 där Azure och GC hade en likvärdig responstid och AWS var ungefär hälften. I genomsnitt, hade AWS lägst responstid i detta scenario men detta är på grund av att AWS kastade fel så fort alla instanser var upptagna medan de andra leverantörerna försökte hantera detta genom att köra igång flera instanser vilket kan ha lett till den högre responstiden jämfört med AWS.

Även om benchmarking och prestandatester är värdefulla metoder för att utvärdera FaaS erbjudanden och molnleverantörer, finns det brister, begränsningar och utmaningar som bör noteras.

1. **Benchmarks representerar inte verkliga applikationer:** Funktionerna som testades representerar inte verkliga applikationer. Verkliga applikationer kan bli extremt komplexa, med beroenden av flera tjänster, databaser, tredje parts API:er och mer. Om en funktion är beroende av en databas som fungerar långsamt, till exempel, skulle ett benchmarktest av bara funktionen i sig inte

fånga denna prestandaflaskhals.

2. **Begränsad omfattning:** Benchmark-tester fokuserar ofta på isolerade aspekter av ett system. De kan till exempel testa en funktions prestanda under hög belastning men inte ta hänsyn till andra aspekter som säkerhet, underhållbarhet eller funktionens integration med andra komponenter. Att fokusera på individuella funktioner kan skymma större arkitektoniska överväganden. Valet av en molnleverantör beror ofta på mer än bara prestanda för enskilda funktioner. Det kan påverkas av faktorer som övergripande ekosystem, kompatibilitet med befintlig teknik, stöd för specifika programmeringsspråk, prismodell, datacenterplatser och leverantörens meritlista för tillförlitlighet och support.

3. **Brist på långsiktiga data:** Benchmarks körs vanligtvis över en kort stund, medan riktiga applikationer måste fungera tillförlitligt under långa tidsperioder.

4. **Testkostnader:** Prestandatestning av molnfunktioner kan bli kostsamma, särskilt i stor skala. Varje anrop av en funktion och tillhörande beräkningstid, minnesanvändning och dataöverföringar faktureras av leverantören. Detta går hand i hand med förra punkten. Testningskostnader kan orsaka att testerna inte körs lika intensivt som det borde.

5.2 Diskussion av kostnadsanalys

I följande delkapitel diskuteras både direkta och indirekta kostnader för de olika molnleverantörerna.

5.2.1 Direkta kostnader

Google Cloud är en tydlig förlorare när det gäller de direkta kostnaderna. Detta är eftersom Azure och AWS har en liten prisskillnad medan Google Cloud är cirka 30% dyrare för samma antal anrop. Dock kan detta vara vilseledande när det gäller helheten av kostnaden för molntjänsten då flera olika tjänster kan komma att behövas att användas som kostar olika mycket beroende på leverantör.

5.2.2 Indirekta kostnader

De indirekta kostnaderna blir ungefär samma oavsett val av molnleverantör. Ett sätt att minska de indirekta kostnaderna är om utvecklarna och personalen redan har kompetens i en av molnleverantörerna, då sänks personalutbildningskostnaden samt tiden som går åt att migrera vilket leder till en reducerad utvecklingskostnad. Därför är det viktigt att ha utvecklarnas och personalens tidigare kompetens i åtanke när man ska flytta till molnet. Eftersom utvecklarna hos Xpektor har sedan tidigare erfarenhet av att arbeta med Azure kommer valet av Azure som leverantör leda till en snabbare och därav billigare migrering till molnet.

6
Slutsats

I en tid som domineras av digital transformation ställs organisationer inför det komplexa valet av molnleverantörer. Detta examensarbete påbörjade en resa för att utvärdera och granska migreringen av den webbaserade applikationen, Xpektor, till en molnplattform, i förhoppning av att redovisa hur beslutsproccesen när det gäller val av en molnleverantör kan se ut.

Genom att analysera Xpektors behov, prestandatestning av framstående molnleverantörer och en kostnadsanalys, styr resultaten mot en rekommendationen av Microsoft Azure som den föredragna molnleverantören för Wecoorp. Azure visade inte bara tillfredsställande prestanda i de utförda testerna, utan visade sig också vara ett ekonomiskt pragmatiskt alternativ. Dessutom, utöver bara mätvärden och priset, framträdde den mänskliga faktorn, expertis och tidigare kompetens hos utvecklarna med Azure, som en avgörande faktor i denna beslutsprocess.

Även om migreringen av Xpektor till Azure har en synergi mellan behov och befintliga kompetenser, är det viktigt att påpeka att migrering till molnet och valet av en leverantör är mångfacetterad och styrs av många komplexa faktorer som är unika för varje applikation.

Sammanfattningsvis avslutas detta examensarbete inte bara med en rekommendation utan förespråkar för en kontinuerlig bedömning av den snabbt utvecklande marknaden av cloud computing, för att säkerställa att de IT-beslut som tas sammanfaller med de organisatoriska målen inom ett företag.

Litteraturförteckning

[1] AWS author. Lambda execution environments. https://docs.aws.amazon.com/lambda/latest/operatorguide/execution-environments.html, 2023. [Online; accessed 15-October-2023].

[2] K6 author. What is k6 cloud? https://k6.io/docs/cloud/. [Online; accessed 28-Mars-2024].

[3] Matplotlib author. Matplotlib: Visualization with python. https://matplotlib.org/. [Online; accessed 28-Mars-2024].

[4] Pandas author. 10 minutes to pandas. https://pandas.pydata.org/docs/user_guide/10min.html. [Online; accessed 28-Mars-2024].

[5] AWS authors. Lambda function scaling. https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html, 2023. [Online; accessed 15-October-2023].

[6] SauseLabs authors. Top 10 biggest cloud providers in the world in 2023. https://saucelabs.com/resources/blog/api-load-testing-vs-performance-monitoring, 2022. [Online; accessed 15-October-2023].

[7] Solarwinds authors. Sql database definition. https://www.solarwinds.com/resources/it-glossary/sql-database. [Online; accessed 28-Mars-2024].

[8] Peter Bogetoft and Peter Bogetoft. Introduction to benchmarking. *Performance Benchmarking: Measuring and Managing Performance*, pages 1–21, 2012.

[9] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. The rise of serverless computing. *Communications of the ACM*, 62(12):44–54, 2019.

[10] Microsoft contributors. A tour of the c# language. https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. [Online; accessed 28-Mars-2024].

[11] Microsoft contributors. Introduction to .net. https://k6.io/docs/cloud/, 2023. [Online; accessed 28-Mars-2024].

[12] Mozilla contributors. What is javascript? https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript. [Online; accessed 28-Mars-2024].

[13] David Ricketts. Cloud paralysis – the paradox of choice. Available at https://www.linkedin.com/pulse/cloud-paralysis-paradox-choice-david-ricketts/, 2018. [Online; accessed 15-October-2023].

[14] Eurostat. Cloud computing - statistics on the use by enterprises. Available at https://ec.europa.eu/eurostat/statistics-explained/index.php?

title=Cloud_computing_-_statistics_on_the_use_by_enterprises. [Online; accessed 15-October-2023].

[15] Marcus Law. Top 10 biggest cloud providers in the world in 2023. https://technologymagazine.com/top10/top-10-biggest-cloud-providers-in-the-world-in-2023, 2023. [Online; accessed 15-October-2023].

[16] Mahendra Prasad Nath, Roopa Sridharan, Amit Bhargava, and Thariq Mohammed. Cloud computing: an overview, benefits, issues & research challenges. *idea*, 1(3), 2019.

[17] Tony Scott. Microsoft cio tony scott: How to avoid "cloud paralysis". Available at https://blogs.microsoft.com/blog/2011/04/27/microsoft-cio-tony-scott-how-to-avoid-cloud-paralysis/, 2011. [Online; accessed 15-October-2023].

[18] Wikipedia contributors. Benchmarking — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Benchmarking&oldid=1169568855, 2023. [Online; accessed 15-October-2023].

[19] Wikipedia contributors. Cloud computing — Wikipedia, the free encyclopedia. Available at https://en.wikipedia.org/w/index.php?title=Cloud_computing&oldid=1179919351, 2023. [Online; accessed 15-October-2023].

[20] Wikipedia contributors. Cloud computing — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Cloud_computing&oldid=1179919351, 2023. [Online; accessed 15-October-2023].

[21] Wikipedia contributors. Cold start (computing) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Cold_start_(computing)&oldid=1176465669, 2023. [Online; accessed 15-October-2023].

[22] Zachary Stank. Choosing a cloud service provider: Pros and cons of some popular options. Available at https://tinyurl.com/vpu63fsf, 2020. [Online; accessed 15-October-2023; Shortened URL by https://tinyurl.com/app].

INSTITUTIONEN FÖR DATATEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS