Figurer

Tabeller

Kodlista

Akronymer och förkortningar

AWS	Amazon Web Services
CUR	Customer Usage Report
JSON	JavaScript Object Notation
SQL	Structured Query Language

Kapitel 1

Introduktion

I denna rapport undersöktes hur man kan korta ned tiden det tar att utvinna data ur en fil genom att förbättra den nuvarande arkitekturen. Dessa förbättringar bestod av att flytta steget för datafiltreringen till ett tidigare skede i kedjan. I samband med detta undersöktes också om dessa förbättringar kunde bidra till lättare implementation av framtida tjänster i produkten.

1.1 Bakgrund

I dagsläget skickas kundens användningsrapport till produkten. I denna rapport, som härstammar från Amazon Web Servicess (AWSs), finns mycket information om kundens användande av AWSs tjänster. Majoriteten av denna information är inte intressant för produkten. Den enda informationen som produkten behöver är vilken typ av tjänst som använts, hur mycket energi som gått åt när denna tjänst använts samt vilka datum tjänsten körts. Produkten filtrerar sedan ut all irrelevant information. Därefter filtreras data återigen för att få fram rätt information till rätt tjänst inom produkten och sparas i en databas. Sedan sker beräkningar för de olika tjänsterna och resultatet sparas. Därefter kan slutanvändaren få tillgång till sitt data.

1.2 Problem

Ett problem är att det tar för lång tid att filtrera ut relevant data ur användarrapporterna. Ett annat problem är den långa kedjan mellan rapporten och slutanvändaren för en specifik tjänst. I denna kedja modifieras data flera gånger för att passa in för den specifika tjänsten samt att hemsidan sedan ska

kunna hämta ut data med en rimlig responstid. Rimlig i detta sammanhang är maximalt tre sekunder.
Då uppstår frågan om man kan förbättra tiden som det tar att filtrera ut data ur originalrapporten? Därefter uppstår även en annan fråga, kan man korta ned kedjan mellan rapporten och användare och på det sättet göra en simplare arkitektonisk lösning som bidrar till enklare implementation av framtida tjänster?

1.3 Syfte

Syftet med denna rapport var att bidra med en förbättring på nuvarande lösning för uppdragsgivarens tjänst. Ett annat syfte var också att visa på hur man på ett effektivt sätt kan filtrera data innan användning. Detta är i synnerhet intressant då man arbetar med stora datamängder och enbart är intresserad av att extrahera specifik data ur datamängden.

1.4 Mål

Målet med detta arbete var att producera ett program som tar emot en fil med användardata och sedan returnerar önskad data snabbare och mer effektivt än nuvarande lösning. Detta blev också kravet på uppgiften, d.v.s. att den ska vara mer effektiv än nuvarande lösning. Denna lösning delades upp i tre delmål:

1. Öka effektiviteten hos uppdragsgivaren
2. Bidra med en lösning som på ett snabbt och effektivt sätt kan hämta ut önskat data ur en datamängd.
3. Inspirera till en mer generell arkitektur som möjliggör enklare implementation av nya tjänster

1.5 Metod

För att projektet kunde genomföras och svara på frågeställningen så behövdes viss kunskap inhämtas. Detta gjordes genom att ta del av dokumentation om tekniker som användes i produktion. Dessa tekniker bestod främst av Amazon och dess molnmiljö. Efter det kom metoden att bestå av att bygga själva programmet som tar emot data och sedan returnerar önskat data på ett snabbare och mer effektivt sätt än nuvarande lösning. Därefter skulle tidsåtgången

för den nya lösningen jämföras med den nuvarande lösningen för att kunna fastställa om det är en bra lösning. Utöver det skulle även arkitekturen jämföras för att se om den bidrog till en enklare implementation av framtida tjänster.

1.6 Avgränsningar

Rapporten avser endast specifika fall för uppdragsgivaren och kommer därmed inte att presentera någon generell lösningmetod för att hämta data på ett snabbt och effektivt sätt ur valfri datamängd. Rapporten ska snarare ses som tips på hur man kan lösa problemet med att hämta data ur en datamängd på ett effektivt sätt, men sättet kan behöva modifieras beroende på användningsområdet. Lösningen avser också endast data i parquettformat[1] då det är vad som används av uppdragsgivaren.

1.7 Disposition

I kapitel 2 presenteras relevant information och bakgrund till arbetet där läsaren kan få information om tekniker som använts och är nödvändig för att förstå lösningen. Kapitel 3 presenterar dels den vetenskapliga metoden som använts för att driva projektet framåt och dels den tekniska metoden som har använts för att bygga produkten som kan ge svar på frågeställningen. Kapitel 4 presenterar hur konstruktionen av lösningen har gått till och vilka val samt beslut som gjorts. Kapitel 5 visar resultatet av arbetet som gjorts samt en kort analys av dessa resultat och därefter följt av kapitel 6 där slutsats presenteras. Kapitel **??** går slutligen igenom rekommendationer samt eventuellt framtida arbete.

Kapitel 2

Bakgrund

I detta kapitel behandlas grundläggande information om hur nuvarande system är uppbyggt för att förtydliga skillnaden mellan den nya implementationen och den gamla. Därtill förklaras även några olika tjänster som används i båda lösningar.

2.1 Amazontjänster

Nedan kommer några urval av tjänster som Amazon tillhandahåller listas och förklaras och som används i produkten.

S3 behållare

En S3 behållare[2] är en tjänst från Amazon som lagrar objekt i molnet. Ett objekt kan vara i princip vad som helst, i detta arbetets fall kommer det vara parquetfiler. Man kan ladda upp och ladda ned objekt till en S3 behållare från vart som helst, så länge man har rätt adress och rätt behörigheter till behållaren. Varje gång någonting händer i behållaren, allt från att ett objekt skapas (laddas upp), raderas eller ändras, skapas en händelse. Denna händelse kan användas för att få information om vad som hände i behållaren.

AWS Lambda

AWS Lambda är en tjänst som möjliggör att en användare kan köra sin kod i molnet utan att behöva ta hänsyn till någon serverinfrastruktur eller liknande. AWS Lambda används i samband med olika händelser, dessa kan man välja att ställa in själv. AWS Lambda tillhandahåller all administration

runt användarens resurser. Detta kan vara allt från loggar, om funktionen ska använda parallelarism, hur mycket minne som ska användas o.s.v. Det enda användaren behöver för att komma igång är att skriva själva koden som ska köras.

Lager

För att använda bibliotek som inte stöds av Lambda internt så måste man lägga till lager[3] i sin Lambdafunktion. Ett lager är egentligen bara filer som innehåller önskade bibliotek. Man laddar ned de bibliotek som behövs och komprimerar filerna för att sedan ladda upp de i sin Lambdafunktion.

AWS Glue

AWS Glue[4] kan användas till många olika områden inom dataintegration och transformering men i detta arbete används det endast till att skapa en datakatalog av våra datafiler. Detta är fördelaktigt då man kan sätta ihop flertalet filer till en gemensam katalog som sedan uppdateras när nya filer läggs till. Den kan även partitioneras utefter månader eller dagar för att underlätta sökning av data. Man kan sedan ställa frågor till denna katalog som till en vanlig databas med hjälp av t.ex. Structured Query Languages (SQLs). För att denna datakatalog ska kunna byggas krävs konfiguration av en krypare (eng. crawler). Då behövs ursprung för data adderas, t.ex. en S3-behållare, samt där katalogen ska sparas. Denna katalog sparas t.ex. i den behållare man läser data ur.

AWS Athena

Athena[5] är en interaktiv tjänst som Amazon tillhandahåller. Den möjliggör för användaren att på ett enkelt sätt börja ställa frågor direkt till ett dataset. Det krävs endast hänvisning till en S3 behållare samt definiering av ett schema. Att definiera ett schema kan med fördel göras av tjänsten nämnd ovan, Glue. Frågorna kan ställas via språket SQL.

Användarrapport

En användarrapport[6], eller Customer Usage Reports (CURs), är en rapport som varje användare av Amazons tjänster kan begära att få ut av Amazon. Dessa rapporter är underlaget för data i arkitekturen som skall förbättras i detta

arbete. Rapporten innehåller all data om användaren i form av användning, kostnader, energiåtgång, region med mera.

2.2 Python

Produkten kommer använda sig av programmeringsspråket Python samt några tillhörande bibliotek. Python är ett högnivåspråk och använder sig av dynamisk typning[7], till skillnad från t.ex. Java som är statiskt typat[8]. Detta språk valdes då den dels har support i AWS Lambda samt att den nuvarande lösningen som teamet använder sig av är gjort i Python.

Pandas

Pandas är ett bibliotek till Python som fokuserar på analys och manipulation av data[9]. Det är ett väldigt behändigt instrument när man arbetar med olika sorters datamängder. Det som detta arbete mest kommer använda sig av är att läsa in data i en ”dataFrame”[10] som är en tvådimensionell tabell.

PyArrow

PyArrow är ett annat bibliotek som används i samband med Pandas och dess funktioner för att läsa data ur en parquet-fil. PyArrow används som motor i Pandas när en parquet-fil öppnas och läses i minnet[11]. Detta projekt använder endast PyArrow som motor i Pandas när man öppnar en fil och således ligger vidare förklaring av biblioteket utanför detta projekt.

Boto3

Boto3 är ett verktyg som man kan importera till Python för att kunna skapa, köra och konfiguera AWS tjänster i sin kod[12]. Det tillhandahåller ett bra API som gör det lätt att förstå hur det fungerar.

Kapitel 3

Metod

I detta kapitel presenteras de teoretiska projektmetoder som använts samt tekniska metoder som använts för att kunna genomföra undersökningen.

3.1 Bunges metod

Detta projekt har genomförts med hjälp av Bunges generella vetenskapliga metod som har anpassats till en mer teknologisk undersökning[13]. Nedan kommer metodens olika steg att listas samt förklaras på vilket sätt projektet har genomgått dessa.

1. Hur kan den aktuella problemställningen lösas?

Detta var det första steget i projektet efter att problemformuleringen definierats. Tillsammans med kontaktperson på företaget så diskuterades olika idéer. En idé som uppstod var att minska mängden data som Athena behövde gå igenom när frågor till data ställdes. Efter att jag undersökt nuvarande arkitektur så presenterades denna idé till företaget som en eventuell lösning på problemet. Företaget tyckte att det lät som en lösning som kunde vara värd att prova.

2. Utveckling av produkt

Produkten för att lösa problemet kommer använda likadan teknik som redan finns i den befintliga arkitekturen för att lätt kunna testas samt eventuellt integreras. Dessa tekniker tillhör alla Amazonsfären och består av Lambda, Athena, S3, och Glue. Utöver dessa tekniker kommer själva koden

i Lambdafunktionen skrivas i Python. Vidare kommer bibliotek anpassat till datahantering användas, nämligen Pandas samt PyArrow.

3. Underlag

Nästa steg i processen var att hitta information om dessa tekniker. En källa till information var teamet på företaget. Möten på veckovis basis planerades in och för mindre och snabba frågor fanns även mail/chat tillgängligt. Vidare så finns god dokumentation om alla dessa tekniker på Amazons hemsida och dessa användes flitigt, se mer detaljerad beskrivning i avsnitt 2.1.

4. Utveckling av produkt

Produkten blev utvecklad utefter underlaget i steg 3 och med fokus på slutsatsen från steg 1. Då produkten visar sig fullgod följs detta av steg 5.

5. Modell

En modell har skapats. Modellen skapades redan i steg 1 och har sedan stått till grund för utvecklingen av produkten. Modellen av den föreslagna produkten är således densamma. Modellen presenteras i figur 4.2

6. Konsekvenser av modell

Modellen har agerat som stomme under utvecklingen för att ha en sorts mall att följa för arkitekturen. Den kan också ses som en rättningsmall då den var likadan för den färdiga produkten som för den planerade produkten i början av processen.

7. Tillämpning av modell

Produkten har testats med riktig data för att simulera verkligheten. Utfallet blev som förväntat, alltså tillfredsställande. P.g.a. detta följer steg 9.

8. Identifiering av brister

Inga brister upptäcktes under testningen av produkten.

9. Utvärdering

Resultatet har utvärderats i förhållande till nuvarande implementation. Utifrån dessa jämförelser har nya förbättringsförslag identifierats, se mer i kapitel 5.

3.2 Undersökningsparadigm

Undersökningen bygger på en fallstudie av ett specifikt problem för det berörda teamet. En förutbestämd ansats är att en mindre datamängd borde i bästa fall förminska tiden det tar för en fråga till en datamängd att returnera ett svar och i värsta fall kommer tiden vara oförändrad men att kostnaden för tjänsten kommer att minska med den mindre datamängden då den baseras på mängden data som måste gås igenom.

3.3 Teknisk metod

För att kunna testa den tänkta lösningen samt jämföra med den nuvarande behövs en teknisk produkt byggas. Produkten byggs och utvecklas i autentisk AWS-miljö. För att få hög reliabilitet används autentisk data, d.v.s. samma data som används i produktion. Denna data sparas i en behållare. Sedan filtreras datat med hjälp av en Lambdafunktion. Därefter sparas datat i en ny behållare. Genom detta kan vi nu mäta och försöka uppnå målen i sektion 1.4. Dels genom att jämföra tiden det tar att köra frågor på datamängden med Athena samt tiden det tar att köra samma frågor på ofiltrerat data, och dels genom att mäta tiden det tar att filtrera datat via Lambda jämfört med liknande filtrering via Athena. Dessa två mätningar kommer att ge svar på om lösningen fungerar eller inte.

3.4 Validitet

I följande avsnitt kommer rapporten avhandla delarna om hur validitet kan garanteras samt hur lösningen ger en rättvis bild av resultatet.

Validitet av metoden

Bunges vetenskapliga metod kommer att valideras genom genomfört och avklarat projekt. Om projektet blir avklarat så har metoden bidragit till en struktur för att lösa problemet.

Validitet av data

Metoden kommer att ge korrekta svar. Detta p.g.a. att underliggande data kommer att vara samma när testerna körs. Undersökningen kommer alltså använda exakt samma datafiler för den gamla och den nya lösningen. Detta medför också att man kan validera undersökningen genom att se att utdata är detsamma efter mätningarna för de båda lösningarna.

3.5 Analys av data

Produkten kommer analyseras med hjälp av loggar som skapas i AWS-miljöer. Ur dessa loggar kan man se hur lång tid det tar för en tjänst att bli klar. I mitt fall blir det hur lång tid det tar att hämta ut data. En annan analys som görs är en analys av kostnad. Då kostnaden beror på datamängden som gås igenom borde det även här bli en mätbar skillnad. Denna information finns också tillgänglig genom loggarna som skapas.

Kapitel 4

Föreslagen arkitektur

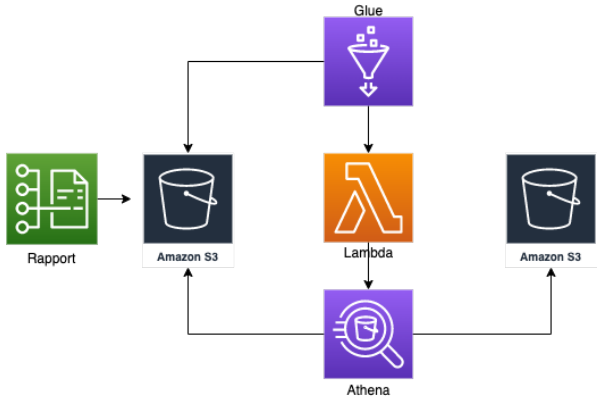
I detta kapitel presenteras den tekniska lösningen som byggts för att kunna genomföra undersökningen. Den går igenom olika konfigurationer som är nödvändiga samt kod som körs i vissa funktioner. Vidare presenteras bilder över den nya arkitekturen samt den gamla för att illustrera skillnader.

4.1 Arkitektur

Nedan visas skillnader på nuvarande arkitektur samt den nya arkitekturen som utvecklats.

Nuvarande arkitektur

Den nuvarande lösningen, se figur 4.1, är sammansatt av de olika tjänsterna som listats i 2.1. En eller flera användarrapporter laddas upp till en S3 behållare. I det verkliga fallet handlar det om många olika rapporter per dag från olika användare. Alla dessa rapporter hamnar i samma behållare. Sedan används Glue för att skanna av behållaren. Denna skanning sker med ett förutbestämt tidsintervall. När skanningen sker och det finns ny data kommer datakatalogen att uppdateras med den nya informationen. Detta kommer att sätta igång Lambdafunktionen som i sin tur sätter igång Athena som kör en ny fråga till den ursprungliga behållaren. Genom denna fråga som består av SQL kod filtreras sådan data ut som är av intresse för själva tjänsten. Resultatet sparas i en ny behållare. Från den nya behållaren finns sedan andra steg i kedjan men dessa är inte av intresse för detta arbete.



Figur 4.1: Nuvarande arkitektur

Föreslagen arkitektur

I 4.2 visas den föreslagna arkitekturen. Det finns två stora skillnader jämfört med den nuvarande arkitekturen i 4.1.

Figur 4.2: Föreslagen arkitektur

Filtreringen sker med Python istället för Glue och Athena samt att resultatet sparas som en `parquet-fil` istället för en tabell.

4.2 Konfiguera S3 behållare

För att kunna använda en S3 behållare måste man skapa den samt konfiguera inställningar på ett sådant sätt att användarens behov kan tillgodoses. Under arbetet har två behållare använts, en som man läser data från, och den andra som man skriver det filtrerade data till.

Läsa från S3 behållare

En S3 behållare skapas genom Amazons grafiska användargränssnitt. I detta fall heter behållaren ”nhbucket-read”. Därefter måste man konfiguera behållarens säkerhetsinställningar. Detta görs genom att definiera en policy för behållaren i form av en JavaScript Object Notations (JSONs)-fil, se 4.1.

Kod 4.1: Definiton av en policy för en behållare med rättigheter att läsa från den

```
{
    "Version": "2012-10-17",
    "Id": "Policy1650890435647",
    "Statement": [
        {
            "Sid": "Stmt1650890423376",
            "Effect": "Allow",
            "Principal": {
                "AWS": "AWSIAMUSERNAME"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::nhbucket-read/*"
        }
    ]
}
```

I denna policy beskrivs vad en specifierad användare kan göra. I detta arbetets fall tillåter vi användaren att hämta ett objekt i resursen ”nhbucket-read”. Rersursen är alltså S3 behållaren vi använder oss av. Det riktiga användarnamnet har ersatts av ”AWSIAMUSERNAME” p.g.a. säkerhetsskäl. Detta har även gjorts i 4.2.

Skriva till S3 behållare

För den andra behållaren som ska ta emot filer behövs policyn definieras på samma sätt som ovan men med en annan funktion. Behållaren ska kunna skrivas till.

Kod 4.2: Definiton av en policy för en behållare med rättigheter att skriva till den

```
{
    "Version": "2012-10-17",
    "Id": "Policy1650890435647",
    "Statement": [
        {
            "Sid": "Stmt1650890423376",
            "Effect": "Allow",
            "Principal": {
                "AWS": "AWSIAMUSERNAME"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::nhbucket-write/*"
        }
    ]
}
```

4.3 Konfiguera AWS Lambda

För att en Lambdafunktion ska ha någon nytta behöver vi definiera en händelse som triggar funktionen att köra igång. Detta görs genom att först välja vart händelsen ska komma ifrån och sedan vilken specifik händelse som ska trigga funktionen. Vi väljer att koppla vår behållare som heter "nhbucket-read" till funktionen och sedan händelsen "ObjectCreatedByPut". Detta betyder att när en fil skapas i behållaren så skickas en händelse till vår lambdafunktion vilket resulterar i att koden i funktionen körs.

Kod 4.3: AWS Lambda händelse

```
S3:nhbucket-read
arn:aws:s3:::nhbucket-read
```

EventType : ObjectCreatedByPut

Utöver händelsekonfigurationen i listing 4.3 så måste vi även definiera vad funktionen ska vara tillåten att göra. Detta görs som i S3 behållarna genom en policy i JSON.

Kod 4.4: Definiton av en policy för AWS Lambda och dess rättigheter

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::nhbucket-read/*"
        },
        {
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::nhbucket-write/*"
        },
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "arn:aws:logs:eu-north-1:
            906791142533:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:eu-north-1:906791142533:
                log-group:/aws/lambda/parquetFilter:*"
            ]
        }
    ]
}
```

Som vi ser i 4.4 så har vi alltså tillåtit funktionen att hämta ett objekt (”GetObject”) från ”nhbucket-read”, samt att skapa ett objekt (”PutObject”) i ”nhbucket-write”. Läsaren kan också notera några andra händelser som är tillåtna men dessa är bara för intern logging och definieras automatiskt när man skapar funktionen.
Nu finns det alltså en arkitektur som tillåter en användare att skapa objekt i behållare 1. När objektet skapas skickas en händelse till Lambdafunktionen som är kopplat till behållare 1. Sedan körs kod definierat i avsnitt 3.4 och därefter skapas en fil i behållare 2.

4.4 Kod i AWS Lambda

I följande avsnitt kommer koden som körs i Lambdafunktionen att presenteras. Den har i uppgift att ta emot en fil, filtrera den och sedan ladda upp den nya filtrerade filen.

Senast skapad fil

För att kunna hämta rätt fil från behållaren så måste programmet veta vilken fil som är den senast skapade. Detta görs med hjälp av händelsen som tas emot när en fil skapas i behållaren. I händelsen finns all information och därifrån kan man hämta namnet på filen som skapades. Koden i 4.5 tar emot en händelse om argument. Ur händelsen kan sedan fältet ”Records” hämtas ut som är en lista med objekt. Därefter extraheras det senaste objektet som ligger på första plats i listan och ur den kan sedan filnamnet som ligger under ”key”-fältet hämtas ut och returneras.

Kod 4.5: Kod i Python för att hämta filnamnet på den senast skapade filen

```
def getLatestFileName(event):
    eventArr = event.get('Records')
    eventObj = eventArr[0]
    return eventObj.get('s3').get('object').
    get('key')
```

Nedladdning

I 4.6 finns koden för hur man hämtar en fil från en S3 behållare.

Kod 4.6: Kod i Python för att hämta en fil

```
def download(fileName):
    s3 = boto3.client('s3')
    s3.download_file(BUCKET_FROM, fileName, '/tmp/'
    + fileName)
    print("download ok")
    return True
```

Metoden kräver ett filnamn som argument. Filnamnet är namnet på den senast skapade filen som beskrivs i 4.4. Sedan definieras vilken typ av client som används genom boto3. Vi använder här en S3 behållare. Sedan laddas filen ned till Lambdas temporära lagring och sparas med sitt ursprungliga filnamn.

Filtrering

I detta steg tas ett filnamn emot som argument. Det är filen som skall filtreras bort av onödig data. I metoden definieras de kolumner som är intressanta under `cur_columns`. Därefter skapas en `dataFrame` genom Pandas och dess läsning av parquet-filen. Denna `dataFrame` innehåller nu endast de kolumner som vi har sagt ska vara med. Därefter sparas datat igen till en parquet-fil i den temporära lagringsmappen för Lambda. Allt detta visas i 4.7

Kod 4.7: Kod i Python för att filtrera en parquet-fil och sedan spara den filtrerade filen

```
def pFilter(file):
    cur_columns = ['line_item_usage_account_id',
    'product_region', 'pricing_unit', 'product_vcpu',
    'product_instance_type', 'line_item_usage_start_date',
    'line_item_usage_end_date', 'line_item_resource_id',
    'line_item_usage_amount', 'line_item_product_code']
    df = pd.read_parquet('/tmp/' + file, 'pyarrow',
    columns=cur_columns)
    df.to_parquet('/tmp/' + file)
    print("filter , done")
```

Uppladdning

I 4.8 ser vi hur en fil laddas upp till en behållare. Funktionen tar emot ett filnamn som argument, filen som skall laddas upp, och försöker sedan ladda

upp den till den fördefinierade behållaren.

Kod 4.8: Kod i Python för att ladda upp en fil till S3 behållare

```
def upload(fileName):
    s3 = boto3.client('s3')
    try:
        response = s3.upload_file('/tmp/' + fileName,
        BUCKET_TO, fileName)
    except ClientError as e:
        logging.error(e)
        return False
    print("upload ok")
    return True
```

Lambda hanterare

När en händelse skickas till Lambdafunktionen så är det metoden i 4.9 som körs. Den tar emot händelsen och använder sedan den för att ta reda på vilken fil det är som triggat händelsen. Detta kommer vara den senast skapade filen som vi vill filtrera. Sedan laddas filen ned, filtreras och slutligen laddas upp i en annan behållare.

Kod 4.9: Kod i Python som körs när Lambdafunktionen triggas av en händelse

```
def lambda_handler(event, context):
    print("initiated")
    downFileName = getLatestFileName(event)
    if download(downFileName) == True:
        pFilter(downFileName)
        upload(downFileName)
        print("done, saved file: " + downFileName)
    else:
        print("failed")
```

4.5 AWS Glue

Innan man kan använda Athena så måste tjänsten Glue användas. Som nämnt i 2.1 så skannar Glue av filerna på en specifierad plats. I detta fallet bygger tjänsten upp en katalog av metadatat i våra filer. Med hjälp av denna katalog

kan sedan Athena användas och ställa frågor till. Filerna skannas med hjälp av en krypare (eng. crawler). Den måste konfigueras till att läsa från rätt behållare. Behållaren den ska läsa från är den behållare där de filtrerade filerna hamnar, $nhbucket - write$.

4.6 AWS Athena

Athena används i produkten för att filtrera data på liknande sätt som gjorts i Lambda, se 4.7. Detta används sedan som en jämförelse i körtider. I koden nedan visas frågan som ställs till data för att extrahera relevant information.

Kod 4.10: Kod som körs av Athena för att filtrera data

```
SELECT
line_item_usage_account_id as AccountId,
product_region as Region,
product_instance_type as InstanceType,
pricing_unit as UsageUnit,
product_vcpu as VCpus,
line_item_usage_start_date as StartDateUtc,
line_item_usage_end_date as EndDateUtc,
line_item_resource_id as ResourceId,
SUM(line_item_usage_amount) as UsageAmount
FROM
"nhdatabase"."nhbucket_write"
WHERE
line_item_line_item_type IN ('Usage',
'DiscountedUsage', 'SavingsPlanCoveredUsage')
AND
line_item_usage_amount != 0
AND
line_item_product_code IN ('AmazonEC2')
AND
product_operation LIKE 'RunInstances%'
AND
NULLIF(product_vcpu, '') IS NOT NULL
GROUP BY
1,2,3,4,5,6,7,8
```

4.7 Datainsamling

Viss data som ligger till grund för undersökningen, alltså själva `parquet-filerna`, innehåller känslig information från olika användare. Dessa filer har anonymiserats på förhand genom att ta bort information som kan koppla datat till en specifik användare, såsom kontonummer eller användarid.

Kapitel 5

Resultat och Analys

I detta kapitel kommer resultatet och analysen att presenteras. Resultatet består dels av mätningar och jämförelser mellan körtider för Athena samt jämförelser mellan körtider för Lambda och Athena.

5.1 Huvudsakliga resultat

Här presenteras tabeller med resultat av mätningar gjorda i AWS. Som nämnt ovan finns det två olika typer av mätningar. En mätning som visar skillnaden på filtrerat och ofiltrerat data där båda körs via Athena samt en annan mätning som visar tiden det tar att filtrera data via Lambda. Tabellen 5.1 visar mätvärden från filtrerad och ofiltrerad data när båda datamängderna körs genom Athena.

Data	Total(MB)	Skannad(MB)	Tid (s)
Ej filtrerad	268.3	33.03	5.652
Filtrerad	68.8	28.46	3.565

Tabell 5.1: Resultat av mätning med Athena

Tabellen 5.2 visar mätvärden från ofiltrerad data som körts genom Lambda.

Data	Total(MB)	Tid (s)
Ej filtrerad	268.3	2.762

Tabell 5.2: Resultat av mätning med Lambda

5.2 Analys av resultat

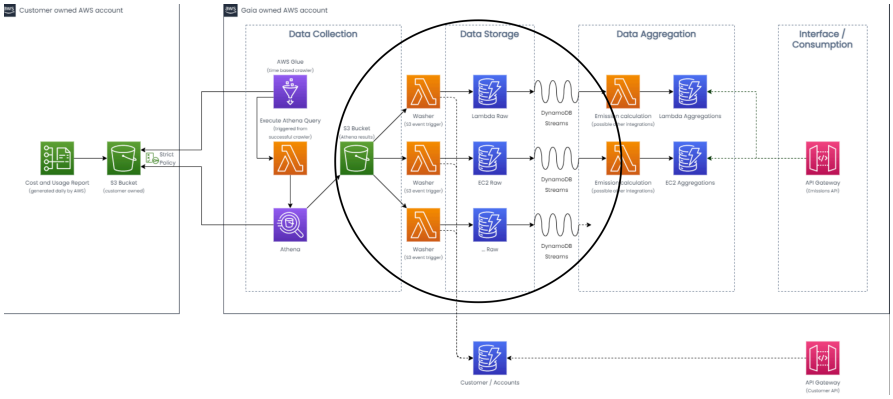
Mätningarna visar på en stor procentuell förbättring när man använder sig av filtrerad data jämfört med ofiltrerad data. Mängden data som skannas av Athena förblir ungefär detsamma, vilket också är rimligt då data är lagrat kolumnvis, men den totala storleken på data verkar ändå spela stor roll för körtiden. Om vi jämför tiderna i 5.1 så får vi en förbättring på över en tredjedel. Då det inte finns så mycket insyn i hur många av AWSs tjänster fungerar under ytan kan man bara spekulera. En kvalificerad gissning baserad på mätdata är att tiden en körning tar inte är proportionell mot storleken på datamängden. Detta kan visa på att det kanske finns en stor tidskostnad för att starta själva tjänsten. Kör man sedan igenom en stor mängd data kommer denna initiala tidsåtgång att bli försumbar, men har man en mindre mängd data som används blir tidsåtgången stor.

Athena eller Lambda

Den nuvarande filtreringen i produktionsmiljön görs genom Athena. Att först filtrera data och sedan köra det i Athena är inte alltid optimalt då det ibland resulterar i samma indata som utdata, dock inte alltid. Då behöver man också jämföra hur lång tid filtreringen tar genom Lambda, då man i vissa fall helt kan skippa steget där Athena används. I tabell 5.2 ser vi också att tiden det tar att filtrera data i Lambda jämfört med Athena är kortare. Så oavsett vilket tillvägagångsätt man väljer så får man ett positivt resultat, en kortare körtid, om man väljer att filtrera data innan man fortsätter att jobba med det.

Analys av ny arkitektur

Man kan nu jämföra den gamla arkitekturen som används i produktionen som illustreras av figuren 4.1 med den nya arkitekturen som illustreras av 4.2. Den stora skillnaden är att man direkt kan hoppa över steget där man behöver Athena för att filtrera bort onödig data. Istället är data reda filtrerat och man kan nu fokusera på att ta fram rätt data för rätt tjänst. Man kan även nu se hur denna lösning bidrar till en förenkling av den totala arkitekturen och möjligheten att lägga in nya tjänster i figur 5.1.



Figur 5.1: Illustration av steg som kan tas bort

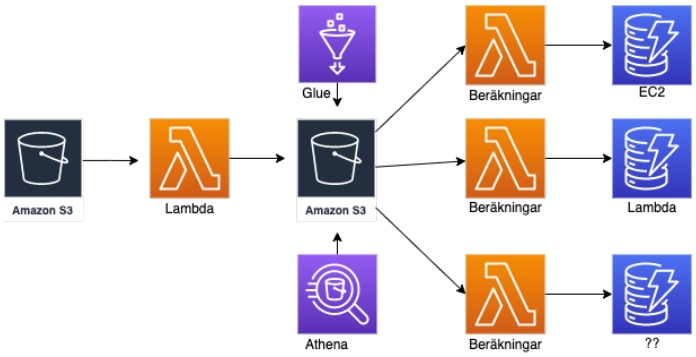
Den inringade delen visar hur man i den gamla arkitekturen behövde behandla data ytterligare efter filtrering och sedan spara undan information för respektive tjänst i en separat databas. Genom att ha färdigfiltrerade filer kan filerna bara sparas i sin behållare och man kan ställa frågor till dessa filer utan kravet att det måste ligga i en databas eller liknande.

Slutligen kan vi i figur 5.2 se hur den nya arkitekturen skulle kunna förändra helheten där man kan ta bort steget med lagring.

5.3 Analys av tillförlitlighet

Resultatet är pålitligt av följande anledningar:

- Samma data har använts som underlag till alla mätningar.
- Samma frågor har använts då data körts genom Athena.
- Tidtagningen är hämtad från AWS egna tidtagningssystem via deras loggar.



Figur 5.2: Framtida arkitektur

- Data hämtas och lagras på samma ställe under alla mätningar.
- Storleken på datamängden är mycket lik den i produktionsmiljö som kan köras under en dag

Vidare är metoden korrekt då samma frågor till data har använts i testmiljön som i den nuvarande lösningen som används i produktionsmiljön. Detta betyder att all hantering av data sker på ett liknande sätt som det hade skett i produktionsmiljön och därmed kan slutsatsen dras att resultatet också är giltigt för en produktionsmiljö och inte enbart under testomständigheter.

Kapitel 6

Diskussion och slutsatser

I detta kapitel kan man läsa om slutsatsen av arbetet samt tillhörande diskussion och framtida undersökningar inom området som kan vara intressanta.

6.1 Slutsats

Genom den produkt jag tagit fram kan man se att mål nummer två i sektion 1.4 har uppfyllts. Detta påvisas genom mätningar presenterade i sektion 5.1. Mätningarna visar tydligt att för den datamängden som tjänsten i snitt hanterar så blir tiden som går åt att filtrera och hämta ut data kraftigt reducerad. Genom den nya arkitekturen som illustreras i figur 4.2 kan man också visa på att mål nummer tre i 1.4 blivit uppfyllt. Detta är dock inte lika enkelt att bevisa men genom att ta bort ett helt steg från den gamla arkitekturen så blir lösningen mindre komplex. Därefter slipper man spara undan data i databaser då filerna består som parquett-filer och dessa kan enkelt sparas i S3-behållare. Det blir då enklare att implementera nya tjänster och koppla extrahering av data till den tjänsten till de redan filtrerade filerna istället för olika databaser, se figur 5.2. Till sist så har det första målet i 1.4 uppnåtts genom en kombination av ovan nämnda mål. En snabbare lösning i kombination med enklare och mer effektiv arkitektur kommer att öka effektiviteten hos uppdragsgivaren.

6.2 Diskussion

I följande kapitel kommer projektets resultat samt metoder diskuteras. Kapitlet är uppdelat i en teknisk del och en teoretisk del.

Projektmetod

Bunges vetenskapliga metod som beskrivits i 3.1 har varit ett bra redskap för att bygga någon slags ramverk kring projektet. Det var ett bra sätt att strukturera upp arbetet kring tydliga milstenar. Från planering av projekt och vilken typ av lösning som skulle kunna fungera till modellering och skapande av produkten. Genom metoden fick man ett bra mått på var i projektet man låg och vilka steg som fanns kvar att göras innan man var färdig.

Teknisk metod

Den tekniska metoden samt produkten som byggts har varit tillräcklig för att lösa problemet samt uppfylla målen. Däremot så skulle den nuvarande produkten inte vara redo att helt ersätta befintlig arkitektur i produktionsmiljö. Man kan mer se produkten som ett verktyg för att mäta skillnaden samt inspirera till en annan sorts arkitektur och lösning än den nuvarande. Min produkt tar inte hänsyn till data som inkommer under flera dagar. Den filtrerar bara data och sparar undan det i en ny behållare. För att produkten mer skulle kunna passa in i produktionsmiljö så skulle utdata behövas partitioneras efter tid och datum.

Resultat

Som redan nämnts i sektion 5.2 så blev resultatet positivt. Lösningen förkortade tiden med mer än en tredjedel jämfört med nuvarande lösning med helt autentisk data. Resultatet visar på att storleken på datamängden spelar roll, dock inte på skannad data då den är lagrad kolumnvis. Men resultatet kan inte förklara varför storleken på datamängden påverkar körtiden även om den skannade datamängden förblir nästan samma. Detta kanske kan vara ett sidospår att undersöka i någon annan undersökning.

6.3 Begränsningar

Det fanns en del faktorer som begränsade mitt arbete och eventuellt resultatet någorlunda. Ett problem var användningen av autentisk data. Den innehåller känslig information om företag såsom kostnadsplaner, användning av tjänster o.s.v. För att kunna arbeta vidare med denna data behövde viss data som kunde identifiera kunden anonymiseras. Detta gjordes av uppdragsgivaren. Det resulterade i att filerna behövde öppnas och sedan återigen komprimeras.

Men här uppstod ett problem då komprimeringen av filerna aldrig på ned på samma nivå som när man fick de direkt från Amazon.

6.4 Rekommendationer

Baserat på resultaten rekommenderas det för uppdragsgivaren att filtrera data genom Lambdafunktionen istället för Athena. Detta minskar körtiderna markant. Genom att filtrera data så fort den kommer in blir det också enklare att lägga till nya funktioner då man inte behöver filtrera bort olika tjänster utan dessa kan sparas, se figur 5.2.

6.5 Framtida arbete

Ett problem som kan vara värt att undersöka är hur man skulle kunna minsta tidsåtgången ännu mer. En faktor som upptäcktes under testerna var att de olika regionerna som man kan välja för tjänsterna kan spela en roll på tidsåtgången. Skulle något annat filformat eller kompression av filerna kunna påverka tiden? Det uppenbara framtida arbetet vore dock att följa rekommendationen från 6.4 och implementera en liknande lösning som presenterats i rapporten, men att anpassa den för produktionsmiljö.

Kostnadsanalys

Kostnaden för AWS Lambda har inte tagits hansyn till i resultaten i denna rapport. Detta skulle kunna vara ett hinder till att implementera den föreslagna lösningen beroende på om kostnaderna skulle vida överstiga de för Athenafiltrering. Detta borde dock inte vara fallet då kostnaderna för AWS Lambda baseras på körtid och körtiderna var inte speciellt långa. Men med ökad datamängd kan man självklart nå en punkt då det blir ohållbart, men det lämnas upp till någon annan att undersöka.

References

[1] Apache, “Apache Parquet,” https://parquet.apache.org/, 2022, [Online; accessed 13-April-2022]. [Page 3.]

[2] Amazon, “Amazon S3,” https://aws.amazon.com/s3/, 2022, [Online; accessed 10-May-2022]. [Page 5.]

[3] ——, “AWS Lambda Layers,” https://docs.aws.amazon.com/lambda/latest/dg/invocation-layers.html, 2022, [Online; accessed 25-May-2022]. [Page 6.]

[4] ——, “AWS Glue,” https://aws.amazon.com/glue/, 2022, [Online; accessed 10-May-2022]. [Page 6.]

[5] ——, “Amazon Athena,” https://aws.amazon.com/athena/, 2022, [Online; accessed 12-May-2022]. [Page 6.]

[6] ——, “AWS Cost and usage reports,” https://docs.aws.amazon.com/cur/latest/userguide/what-is-cur.html, 2022, [Online; accessed 12-May-2022]. [Page 6.]

[7] “General Python FAQ - Python 3.10.4 Documentation,” https://docs.python.org/3/faq/general.html, 2022, [Online; accessed 10-May-2022]. [Page 7.]

[8] Oracle, “Dynamic typing vs static typing,” https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html#:~:text=Java, 2015, [Online; accessed 10-May-2022]. [Page 7.]

[9] “pandas: powerful Python data analysis toolkit,” https://github.com/pandas-dev/pandas, 2022, [Online; accessed 10-May-2022]. [Page 7.]

[10] “DataFrame - Pandas 1.4.2 documentation,” https://pandas.pydata.org/docs/reference/frame.html, 2022, [Online; accessed 10-May-2022]. [Page 7.]