



# **AWS Cost Management and Trend Analysis**

**Clarifying client cost information and reporting**

Eetu Juvonen

Bachelor's thesis

May 2023

Bachelor of Engineering, Information and Communication Technologies

**Juvonen, Eetu**

**AWS Cost Management and Trend Analysis**

Jyväskylä: Jamk University of Applied Sciences, May 2023, 42 pages

Degree Programme in Information and Communication Technologies. Bachelor's thesis

Permission for open access publication: Yes

Language of publication: English

**Abstract**

Qvantel Finland Oy, a software and services providing Finnish company, assigned a thesis subject about cost management and analysis in AWS as a task for an employee. The cost management involved fixing falsely comprised cost information about clients and the way the information was generated. The goal was to create an automatic cost reporting solution with correct values and displayable information.

The solution was conducted using Python, a programming language, and boto3, an AWS proprietary software development kit. During learning AWS's different services and terminology, scripting testing was done in AWS' internal commandline, AWS CloudShell.

The final solution consisted of a cost value correcting report generative script and a new implemented storage solution for the report in AWS. The final script was shared with a team in the company via Git, a tool to ease co-working between developers.

As the result for the solution, the main goals of the project were partially achieved. The script generated a cost report containing corrected information verified by the thesis' writer and the company's thesis assigner. The results were deemed suitable for further development and later for cost reporting.

**Keywords/tags (subjects)**

AWS, AWS Cost Explorer, AWS CloudShell, AWS S3, cost management, cost information, Git, Python, boto3,

**Miscellaneous (Confidential information)**

-

**AWS Kustannushallinta ja Trendianalyysi**

Jyväskylä: Jamk University of Applied Sciences, Toukokuu 2023, 42 sivua

Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisulupa avoimessa verkossa: Kyllä

Julkaisun kieli: englanti

**Tiivistelmä**

Qvantel Finland Oy, suomalainen ohjelmisto- ja palveluyritys, antoi työntekijälle opinnäytetyön aiheeksi kustannushallinnan ja analyysin AWS:ssa. Kustannushallintaan kuului asiakkaiden väärin koostettujen kustannustietojen korjaaminen ja tiedon tuottamistavan tarkastelu. Tavoitteena oli luoda automaattinen kustannusraportointiratkaisu oikeilla arvoilla ja esitettävillä tiedoilla.

Ratkaisu toteutettiin Python-ohjelmointikielellä ja AWS:n omaa ohjelmistokehitystyökalua, boto3:a käyttäen. AWS:n eri palveluiden ja terminologian oppimisen yhteydessä skriptitestaus suoritettiin AWS CloudShell:ssa, AWS:n sisäisessä komentokehotteessa.

Lopullinen ratkaisu koostui kustannusarvoja korjaavasta raportin generoivasta skriptistä ja uudesta toteutetusta varastointiratkaisusta raportille AWS:ssa. Lopullinen skripti jaettiin tiimin kanssa Gitin avulla, joka on kehittäjien yhteistyötä helpottava työkalu.

Ratkaisun tuloksena projektin tärkeimmät tavoitteet saavutettiin osittain. Skripti generoi kustannusraportin, joka sisälsi korjatun tiedon, joka oli vahvistettu opinnäytetyön kirjoittajan ja yrityksen opinnäytetyön antajan toimesta. Tulokset katsottiin sopiviksi jatkokehitystä varten ja myöhemmin kustannusraportointia varten.

**Avainsanat (asiasanat)**

AWS, AWS Cost Explorer, AWS CloudShell, AWS S3, kustannusten hallinta, kustannustiedot, Git, Python, boto3

**Muut tiedot (salassa pidettävät liitteet)**

-

## Contents

<b>1</b>	<b>Thesis introduction – diving into the problem at hand.....</b>	<b>7</b>
1.1	Assignment background.....	7
1.2	Methology .....	7
<b>2</b>	<b>Analyzing assumptions and prerequisites .....</b>	<b>8</b>
2.1	Chapter structure .....	8
2.2	Goals.....	8
2.3	Scripting, programming, coding.....	9
2.4	Points of success .....	10
<b>3</b>	<b>Conceptual framework .....</b>	<b>11</b>
3.1	Viewpoint and methods.....	11
3.2	Amazon Web Services (AWS) .....	11
3.2.1	Billing Console.....	14
3.2.2	EC2 .....	14
3.2.3	Cost Management Console.....	15
3.2.4	Saving Plan .....	16
3.2.5	Tags .....	17
3.2.6	Regions and Zones .....	17
3.2.7	Lambda .....	18
3.2.8	EventBridge.....	19
3.2.9	AWS S3 bucket .....	20
3.2.10	AWS CLI .....	21
3.2.11	AWS CloudShell.....	22
3.3	Python .....	23
3.4	Boto3 .....	24
3.5	Git.....	25
<b>4</b>	<b>Start of scripting .....</b>	<b>26</b>
4.1	NDA .....	26
4.2	Practicing.....	26
4.3	Main points of focus.....	27
4.4	Establishing the scripts.....	28
4.5	The first script .....	29
4.6	The second script .....	37
4.7	Final implementations .....	39

<b>5</b>	<b>Results and notifications.....</b>	<b>41</b>
<b>6</b>	<b>Discussion.....</b>	<b>42</b>
	<b>List of references.....</b>	<b>43</b>

## Figures

Figure 1. Top 10 Cloud Service Providers (Zhang, 2023) .....	12
Figure 2. AWS Regions (Zhang, 2023) .....	12
Figure 3. Microsoft Azure Regions (Zhang, 2023).....	13
Figure 4. GCP Regions (Zhang, 2023) .....	13
Figure 5. Lambda file processing (Lambda, 2023) .....	18
Figure 6. Lambda mobile backends (Lambda, 2023) .....	19
Figure 7. File upload to S3 bucket (Carty, 2023).....	21
Figure 8. Git example (What is Git, 2023).....	25
Figure 9. CloudShell icon.....	28
Figure 10. CloudShell terminal.....	28
Figure 11. Test folder .....	29
Figure 12. Python script file .....	29
Figure 13. Specifying libraries, client and dates .....	30
Figure 14. Dictionaries .....	31
Figure 15. Retrieving record type -data .....	31
Figure 16. Savings Plan Covered Usage .....	32
Figure 17. Total cost of account.....	32
Figure 18. Substracting the account costs .....	33
Figure 19. List of dictionaries .....	33
Figure 20. Correcting the costs .....	34
Figure 21. Creating the DataFrame.....	34
Figure 22. Dropping the empty row.....	34
Figure 23. CSV-file formatting.....	35
Figure 24. Date of today .....	35
Figure 25. File name .....	35
Figure 26. S3-bucket file upload .....	36
Figure 27. A successful file upload to S3.....	36
Figure 28. Downloading the file.....	36
Figure 29. Plan of script development.....	38

Figure 30. Plan of automation development.....	38
Figure 31. Git config (Perveez, 2023) .....	39
Figure 32. Git status (Perveez, 2023) .....	39
Figure 33. Git add (Perveez, 2023).....	40
Figure 34. Git commit (Perveez, 2023) .....	40
Figure 35. Git add origin to remote (Perveez, 2023) .....	40
Figure 36. Git push (Perveez, 2023) .....	40

# **1 Thesis introduction – diving into the problem at hand**

## **1.1 Assignment background**

Having the billing information and costs from different corporate clients not organized is not ideal, nor is the manual labor to organize them and calculate everything accordingly. That is the problem presented by Qvantel Finland Oy and is the purpose of this thesis - bringing clarity to client's cost information and managing them to deliver accurate and easy to understand results.

Cost optimization has different definitions regarding cloud service providers. According to a column by Visu Sontam, in AWS (Amazon Web Services) cost optimization is divided into pillars (Sontam, 2022): Right size, increase elasticity, leverage the right pricing model, optimize storage and measure, monitor, and improve. In right sizing instance types and sizes are matched to the workload, keeping in note the performance and lowest possible cost. Elasticity is meant by the ability to better utilize resources based on application usage. (Sontam, 2022) Leveraging the right pricing model allows the cloud service providers to pay for their resources by their organization's needs in the most cost-effective way. Optimizing storage is meant by analyzing use case, data types, access patterns and IOPS requirements, meaning input and output operations per second. Lastly, by measuring, monitoring, and improving the cloud service provider can develop the cost optimization by examining cost evaluations, monitoring, and analyzing resource usage and by these improving costs. (Sontam, 2022).

## **1.2 Methology**

The thesis research is conducted using existing services found in AWS (Amazon Web Services), while developing own solutions for better cost utilization, cost management and cost reporting. The solutions will be made with the use of analyzing existing data and making necessary additions to the solution based on that. Additionally, making use of existing research regarding the subject is added. The research is either examined from internal documentation in the assignment company or from publicly available sources such as the internet.

Anything of the previously mentioned can't be achieved without properly examining where the company's costs are distributed, if they are necessary, can they be lowered and if they should be

changed to a more cost-efficient counterpart. This thesis covers the cycle of the process for better cost management.

## **2 Analyzing assumptions and prerequisites**

### **2.1 Chapter structure**

Before starting any actual work, analysis, or cost managing, it is important to take some aspects into consideration. What exactly are the main goals for the assignment? What must be done to achieve those goals? What can go wrong? In this chapter these points are handled in a more detailed view and thought through to achieve the best possible outcomes.

### **2.2 Goals**

As mentioned earlier in the thesis introduction, bringing clarity to how the client's billing information is displayed is a big goal. In this situation, client's more detailed costs and billings are observed through a list, in which every individual cost point is listed as a single entity. This makes it not only hard for employees to observe any mistakes in costs or potential increases in some areas, but also for the clients themselves. Any small bump in costs could go unnoticed and can be a major problem eventually. So, the clarification by gathering same area costs to one joined information packet could be the answer. Or by following the current trends and displaying the billing information in a detailed but understandable graph, such as a pie graph, block charts and so on.

The second goal is related to automation. As you will learn later in this thesis, the cost data has some problems with its integrity. To verify that the cost data contents are valid, a manual verification must be done by each billing period to avoid accidental false billing. There are two reasons why this is problematic. Firstly, this causes unnecessary manual labor by personnel inside the company to calculate billing information which should already be correct and valid. This in turn causes a problem when the billable client wants to, for example, know how much their costs have increased from last month's bill. The number will not be always accurate either if there are human calculation errors involved. This could be solved with two steps: correct the cost data so it displays the right information and generate an automatic process for the cost report to include the necessary information for the situations like mentioned in the previous paragraph.



One goal is to advance in any corresponding technology used in the thesis. In retrospection, an area to be advanced in would be any type of scripting/programming or coding since the automation process would potentially involve some of these. Having looked through all these different problems and what goals they generate for this thesis, some goals are not achieved only by “it is now achieved.”

## 2.3 Scripting, programming, coding

Since different terms will be examined later in detail, it is important to understand briefly what is then talked about. The differences between terms *scripting*, *programming*, and *coding* can be unclear to some. Therefore, this section is established to bring some clarity to these concepts.

So, what are the differences? According to Emily Gillespie-Lord, “A scripting language is primarily used to build websites and web applications” (Gillespie-Lord, 2023). So, since scripting is not used to build innovative programs, it is used to link the pieces of the program together and the script is then run by the program. If scripting is used to link pieces of a program together, programming is used to develop a new program from nothing (Gillespie-Lord, 2023). That leaves us only with coding, in which programming languages are translated to a form which the computer can understand (Gillespie-Lord, 2023). Examples given of a coded language in Gillespie-Lords work are computer and machine languages, either being binary or consisting of zeros and ones (Gillespie-Lord, 2023).

After examining these terms, which one of these is the most suitable for use in this thesis? As stated by Gillespie-Lord, “scripting is used to automate the execution of tasks” (Gillespie-Lord, 2023). In addition, as listed in the goals section of this thesis, there was a need of automating the process of cost management, thus relieving some of the manual labor. Also, there was also a requirement and need to utilize existing technologies in AWS. These points further strengthen the role of scripting as the most suitable concept for this thesis and searching for efficient and versatile solutions.

## 2.4 Points of success

After deciding the way of approach, it is important to think through the requirements for the success of this assignment. What is needed for the scripting to succeed? How much preparations must be done to ensure this? In retrospection, in the previous chapter the term scripting was unpacked though its main characteristics. Since it is used to connect different pre-existing technologies together, getting familiar with the corresponding technologies is important.

One important aspect in any new technical project is the documentation. Technical documentation is vital for the longevity of the final product, service, or program since it will be used as a reference point by other people in the future. The documentation is used to demonstrate what has been done, how to use the product or service, how it works and what its features are. The documentation also serves with point of continuing if the development process has not been finished in its entirety. This could involve developing later plans of implementation for the aspects not finished or for to further enhancing the existing aspects. (Cloud Services New York City. 2020)

As another important point for technical documentation can be seen that it enhances collaboration. The technical documentation helps to give a coherent understanding for every team involved in the process later, helping team collaboration on future projects. This same general understanding of how the solution, product or service works is needed for to all the teams to be on the same page. (Cloud Services New York City. 2020).

More importantly in this case the actual implementation must be recognized as functional. If the solution is not recognized functional in the way it was designed to be, development plans will be conducted for later use. Though the way of implementation was free, some end results were seen more necessary than others. They will be covered later in the scripting and end results -sections. So, to success in this project, a level of planning the project, implementation, and testing in addition with result analysis are a necessity.

### **3 Conceptual framework**

#### **3.1 Viewpoint and methods**

The thesis is written from a research and development standpoint. The main point of view from which this thesis derives from is the aim to achieve the most cost effective and work efficient choices possible. Most problems are engaged with by utilizing terms and technologies found in AWS (Amazon Web Services) but since the assignment may require Python programming, integrating terms to the explanations from Python too is a necessity.

The term AWS is applied further on instead of Amazon Web Services for its memorability and length. Also, the concepts including Saving Plan, Region, and for example Cost Management are introduced, as they entail key elements for finding methods to reach the desired outcomes of the thesis. Subheadings included within these main concepts also provide supporting theoretical information. In the next headings, the concepts listed above, and other relevant concepts are introduced in more detail, as well as how they are necessary when explaining the theoretical basis for this thesis.

#### **3.2 Amazon Web Services (AWS)**

"Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud platform, offering over 200 fully featured services from data centers globally" (What is AWS, n.d.). In short, AWS is a cloud platform to securely host services and applications. It is utilized by millions of customers in their businesses for cost lowering, becoming more agile and for faster innovation. (What is AWS, n.d.).

According to AWS, (AWS Benefits, n.d.), users can select operating systems, programming languages, a web application platform, database and "other services you need". Also, one of its perks is a virtual environment for loading the software and services customers' applications would need. This eases the migration process for existing applications and preserves options for building new solutions.

Below Figure 1 from Mary Zhang in Dgtl Infra displays the cloud providers in size comparison (Zhang, 2023). The sizes are measured in Region and Availability Zone counts.

#	Cloud Service Provider	Regions	Availability Zones
1	Amazon Web Services (AWS)	26	84
2	Microsoft Azure	60	116
3	Google Cloud Platform (GCP)	34	103
4	Alibaba Cloud	27	84
5	Oracle Cloud	38	46
6	IBM Cloud (Kyndryl)	11	29
7	Tencent Cloud	21	65
8	OVHcloud	13	33
9	DigitalOcean	8	14
10	Linode (Akamai)	11	11

*Figure 1. Top 10 Cloud Service Providers (Zhang, 2023)*

Figure 2 displays how the Regions have scattered across the globe and where AWS is planning to place their next Regions. According to the article, AWS's has grown its net sales "from ~\$8 billion in 2015, to more than \$17 billion for 2017, reaching \$35 billion by 2019, and now generating almost \$74 billion on an annual basis." (Zhang, 2023). The growth is explained by AWS's expansion starting in 2015, where the Availability Zone count started increasing from 32 in 2015 to 52 in 2017 till now reaching 84 in count. The net sales growth is also said to be affected by AWS launching thousands of new features over time, fueling the expansion (Zhang, 2023).



*Figure 2. AWS Regions (Zhang, 2023)*

Zhang also shares similar figures regarding Microsoft Azure and Google Cloud Platform (GCP).

Though not the same in Region and Availability Zone count than AWS, Figure 3 displays Microsoft Azure's Region placement in a similar way (Zhang, 2023), from which can be observed the Region vast placement being similar to AWS.



*Figure 3. Microsoft Azure Regions (Zhang, 2023)*

Lastly there is GCP, which having in 2023 34 regions and 103 availability zones places it as the third largest cloud service provider (Zhang, 2023). Having less regions and availability zones than the top 2 providers is shown in below Figure 4 as less locations around the globe, situating throughout the United States, Americas, Europe, and Asia Pacific (Zhang, 2023).



*Figure 4. GCP Regions (Zhang, 2023)*

### 3.2.1 Billing Console

The first service handled is Billing Console, which in correspondence to its name is responsible for any billing related information and service inside AWS. AWS reports (Billing Console, n.d.), that Billing Console “allows users to easily understand AWS spending, view and pay invoices, manage billing preferences, tax settings, and access additional Cloud Financial Management services”. Billing Console has several features: one is quickly evaluating if monthly spend is in line with prior periods, forecast, or budget (Billing Console, n.d.). By investigating, users can take corrective actions in a timely manner. According to AWS official documentation, “the AWS Bills page provides a monthly view of the user’s chargeable costs. For monthly billing periods that have not yet closed, the Bills page will display the most recent estimated charges based on services metered to date” (Billing Console, n.d.). AWS Billing Console is one of the two views for reviewing the billing information used in this thesis. The billing information is essential to be understood clearly and simply for optimization purposes and for the clarity development.

### 3.2.2 EC2

Referring to AWS’s official documentation, (What is Amazon EC2, n.d.) AWS EC2 (Amazon Elastic Compute Cloud) is simply put a virtual computer, which can be rented from Amazon for different purposes. Amazon provides many different types and differently priced EC2 instances, so called virtual machines (What is Amazon EC2, n.d.). The machine is configured for its specific use with AMI (Amazon Machine Image). These AMI’s are in short, preconfigured templates, “that package the bits you need for your server (including the operating system and additional software)”.

EC2’s, like many other AWS services, do cost. An answer at AWS to a user’s question (How are Amazon EC2 instance-hours billed? n.d.): “I want to know how I am billed for my Amazon Elastic Compute Cloud (Amazon EC2) instances”. According to the official, “EC2s are billed by either the hour or the second, based on the specifics (instance size, operating system, Region where the instance is launched)” (How are Amazon EC2 instance-hours billed? n.d.). Only the running-state of an instance is billed. No other state is billed. A point to be noted, is that hourly billed instances are billed always by minimum one hour of its launch, meaning when it enters the running-state.

“By second” billing is done in a way, that the user is billed for a minimum of 60 seconds for a started instance). So, for example, if a user stops an instance before the 60 second mark, they will still be billed the remaining of the 60 seconds (How are Amazon EC2 instance-hours billed? n.d.).

However, there is an exception to billing and that is Free Tier. Amazon offers many different types of offers, which consist of 3 different types: Free trials, 12 months free and always free (AWS Free Tier, n.d.). These three offer types are spread across many different services: computing, storage, database, machine learning etc. How these free tiers work, is that the service has been specified with limitations. If these limitations are followed, the services are free. For example, EC2 only supports specific instance types, which are covered by the Free Tier. These instances are limited in memory, storage capacity or processing power and can only be ran 750 hours per month (AWS Free Tier, n.d.).

### **3.2.3 Cost Management Console**

“AWS Cost Management Console is integrated closely with the Billing Console” (What is cost management, n.d.). The difference between the two is that Billing Console is mostly used for ongoing payment managing, when Cost Management Console is used for future cost optimization (What is cost management, n.d.). Utilizing both in combination makes them a valid asset for this thesis. Looking at some of its use cases: with a tagging strategy called Organize, users can construct cost allocation and governance foundation (What is cost management, n.d.). In Report tagging strategy, by using Cost Explorer (a feature in Cost Management Console), users can view in detail their cloud spend and cost data. This can help raise awareness of the user’s cloud spend (What is cost management, n.d.).

Lastly there is Purchase, in which users can use free trials and programmatic discounts based on their workload pattern and needs (What is cost management, n.d.). AWS includes Saving Plans, which help users in running their services or programs more cost efficiently, tailored specifically for their needed usage.

### 3.2.4 Saving Plan

Saving Plans are a feature in AWS, which help further in reducing costs and bill (by up to 72%) (Saving Plans, n.d.). The point is that Saving Plans demand a commitment to a fixed hourly spent for a one- or three-year term, which according to AWS (Saving Plans, n.d.), reduces the overall cost for the service compared to On-Demand prices, which is a pricing model that enables the user to pay when their service is used and needed. Saving Plans are categorized in three different groups, have different pricings and different savings compared to On-Demand pricing:

Compute Saving Plans – Compute Saving Plans group includes Amazon EC2, AWS Fargate and Lambda, which are AWS provided services. Amazon EC2 section enables to select any instance type, size, and tenancy. It also enables any Region and Availability Zone and any operating system. (Saving Plans, n.d.)

EC2 Instance Saving Plans – after selecting an AWS Region and instance type (one-or three-year term is locked to an instance family and Region), provided choice options include any instance size, any Availability Zone, any operating system, and any tenancy. There is also Sage Maker -Saving Plans, but those are not analyzed in this thesis. (Saving Plans, n.d.)

Why these saving plans are to be noted? Well, according to Amazon (Saving Plans Billing, n.d.) official, “AWS Cost Explorer shows you details on the savings realized with your Savings Plans.” This means that Saving Plans coverage is shown in the account billing from Cost Explorer. They display a certain percentage value of the service it is assigned to (depends on the Saving plan) as a bill, which of course is not billed. The reason for the growing difficulty of manual cost report calculations is explained in more detail later in this thesis together with suggestions and listing in how it can be developed.



### 3.2.5 Tags

Tagging is essential for organizing and clarifying cost related information. According to AWS, (Tags, n.d.) users can use tags to organize resources, and cost allocation tags to track AWS costs on detailed level. After activating cost allocation tags, AWS uses the cost allocation tags to organize resource costs on a cost allocation report, to make it easier to categorize and track AWS costs. There are “two types of allocation tags, AWS generated tags and user-defined tags”. (Tags, n.d.). Both tag types must be activated separately for them to appear in AWS Cost Explorer or on a cost allocation report. AWS Tags are labels that the users can apply to AWS resources such as EC2 instances, S3 buckets, and RDS databases. Tags are metadata that can help users to organize, search, and manage your AWS resources more effectively (Tags, n.d.).

### 3.2.6 Regions and Zones

EC2s are hosted in multiple locations world-wide, which are composed of Regions, Availability Zones, Local Zones, Outposts and Wavelength Zones -Regions are separate geographic areas, Availability Zones are isolated locations within Regions and Local Zones are areas for resource, such as compute and storage, placed closer to end-users. (Regions and Zones, n.d.). Regions are designed to be isolated from each other, achieving greatest possible fault tolerance and stability. Resources are also tied to a specific Region, making the viewing of resources also tied to one Region at a time. (Regions, n.d.)

Availability Zones are isolated locations inside Regions. Instances can be distributed across different Availability Zones so if an instance fails, the ran application can be designed to handle requests from a different Availability Zone. (Availability Zones, n.d.)

Local Zones are extensions of an AWS Region, placed nearer to end-users. They have their own connections to the internet and are designed as low latency serving communications for local users. (Local Zones, n.d.)

Outposts is a service that extends AWS infrastructure, services, APIs, and tools to on-premises for customers. In short, it is a pool of AWS compute and storage capacity deployed at a customer site. This pool is operated, monitored, and managed by AWS as part of a Region. (Outposts, n.d.)

### 3.2.7 Lambda

The main script will be implemented to AWS Lambda. According to AWS documentation, (Lambda, n.d.) Lambda is a serverless, event-driven compute service in AWS, which enables the script to be run without any servers or provisioning. The benefits for using Lambda are that you only pay for what you use, and it expands over 200 different AWS services and SaaS (Software as a service) applications (Lambda, 2023).

Below Figure 5 displays how Lambda works in terms of file processing (Lambda, 2023). So, for example when a photo is taken it must be stored somewhere, in this case inside an Amazon S3 Bucket. After this, a Lambda function is triggered, meaning that the preset function for the file processing starts. Lambda then runs the image resizing code and resizes it automatically to different end user devices, such as mobile phones and tablets (Lambda, 2023).

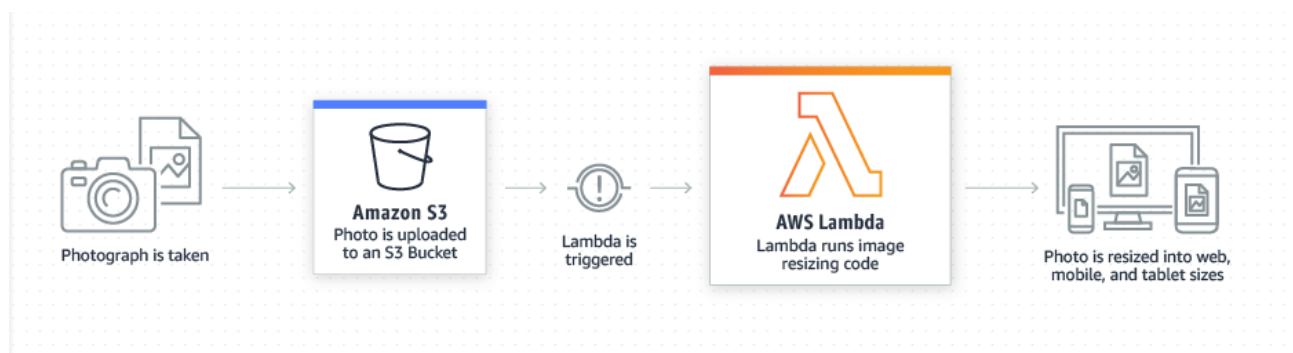


Figure 5. Lambda file processing (Lambda, 2023)

The next Figure 6 (Lambda, 2023) demonstrates how the status update of a user is updated directly to the user's friends. After the user posts the status update, Amazon API Gateway (API standing for Application Programming Interface) makes a REST API call to endpoint. Explanation of REST API: "A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer and was created by computer scientist Roy Fielding." (REST API, 2020).

After this Lambda is once again triggered and runs the code to look up the friends list to update the status notification. Amazon SNS then forwards the update notification to the friends of the user, who will finally receive the status update. Aryush Jain clarifies Amazon SNS brilliantly:

“Amazon SNS (Simple Notification Service) offered by AWS (Amazon Web Services) is an overseen service that provides message delivery or sending of messages to subscribing endpoints or clients” (Jain, 2023).

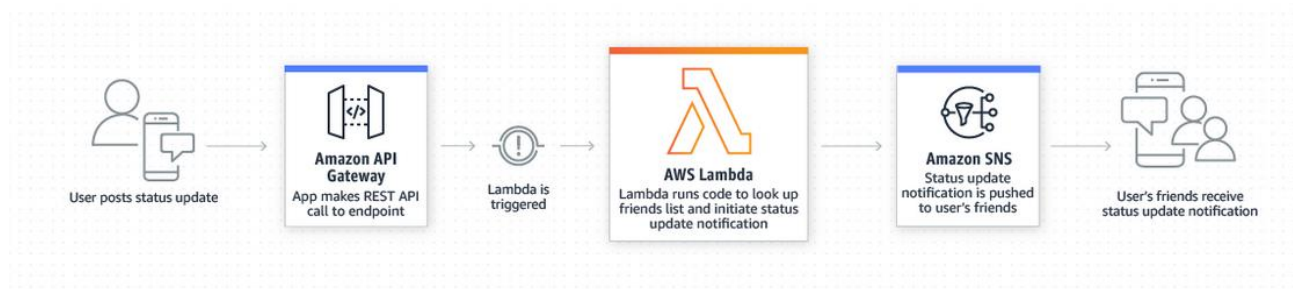


Figure 6. Lambda mobile backends (Lambda, 2023)

### 3.2.8 EventBridge

According to AWS Documentation (What is Amazon EventBridge? 2023), EventBridge is a serverless service which uses events and connects application components together. This eases the process to build scalable event-driven applications. It can be used for example to route events from sources – home-grown applications, AWS services and third-party softwares – to consumer applications.

How does EventBridge work? An event is first received, which is a change in the environment. Then EventBridge applies a rule to route the event to the target. The matching is done either with an event pattern or on a schedule. Regarding cost tracking, EventBridge utilized resources can be assigned with a custom label or a tag (What is Amazon EventBridge? 2023).

There is also identity-based, and resource-based policies used to control the access to EventBridge. Identity-based policies utilize controlling access to EventBridge via permissions of a group, role, or a user. Resource-based policies are assigned to each resource to gain access inside EventBridge. These resources might be for example a Lambda function or Amazon SNS topic. (What is Amazon EventBridge? 2023)

In an example video from the lecturer Eric Johnson, Principal Developer Advocate of AWS, is explained in more detail how EventBridge works. The video can be found in the same page with the previous documentation. (What is Amazon EventBridge? 2023)

### 3.2.9 AWS S3 bucket

AWS S3 (Simple Storage Service) is a highly scalable and durable cloud storage service provided by Amazon Web Services. An S3 bucket is a container for storing objects, which can be any type of data, such as documents, images, videos, and application data. Referring to Kinza Yazar's definition of S3, "An Amazon S3 bucket is a public cloud storage resource available in Amazon Web Services (AWS) Simple Storage Service (S3) platform. It provides object-based storage, where data is stored inside S3 buckets in distinct units called *objects* instead of *files*." (Yazar, 2023). The definition explains, that the S3 buckets are like file folders, so they can be used to store, retrieve, back up and access objects (Yazar, 2023).

AWS S3 bucket has several features to be enabled (Yazar, 2023). One is Versioning control, which helps in preserving every version of an object after performing an operation in the bucket. Versioning control is used in case there have been an overwrite for an object which want to be reversed or an earlier version of the object is wanted for other reasons.

If the objects are wanted to be copied to other buckets, this can be achieved with Object replication. As the name suggests, this can replicate objects between buckets. There are two different replication configurations: Cross-Region Replication and Same-Region Replication. Cross-Region Replication is used when the origin and target buckets reside in different AWS Regions, the Same-Region Replication is therefore used in situations where the all the buckets involved in the replication process reside within the same AWS Region (Yazar, 2023).

As the S3 service has several pros to be used as a storage service: high availability, determining how easily and readily the service can be used. The high availability here is guaranteed by using Availability Zones or Regions (Yazar, 2023). Another pro is that S3 provides unlimited server capacity, so users don't have to worry about interruptions or failures in hard drives or other services.

For the cons, the S3 service does not include a directory structure. All buckets are in a list and the objects inside them. And since S3 is not a physical storage device or service, retrieving or upload can be met with latency or sometimes even failures (Yazar, 2023).

### 3.2.10 AWS CLI

The AWS Command Line Interface (AWS CLI) is according to David Carty's documentation "an Amazon Web Services tool that enables developers to control Amazon public cloud services by typing commands on a specified line." (Carty, 2023). AWS CLI is one of many methods for a developer to use and create or manage AWS tools. A reason for a developer to choose AWS CLI over other methods is that it offers more granular control of services and enables script automation.

AWS CLI is downloadable and installable with many different configurations. Below Figure 7 is an example how with AWS CLI the user can for example upload a file to their S3 bucket (Carty, 2023). The command "aws s3" is typed, to indicate which resource the CLI is going to call. The command section "cp myfolder" is simply: copy files from a folder called myfolder to S3 bucket. Then the defined path to the resource is added, followed by "--recursive". The recursive adds a way to copy every file from the folder and its subfolders to the wanted destination. If the recursive section was not added, only the files from "myfolder" would be uploaded and not also the "subfolder's" (Carty, 2023).

```
$ aws s3 cp myfolder s3://mybucket/myfolder --recursive
upload: myfolder/file1.txt to s3://mybucket/myfolder/file1.txt
upload: myfolder/subfolder/file1.txt to s3://mybucket/myfolder/subfolder/file1.txt
```

*Figure 7. File upload to S3 bucket (Carty, 2023)*

AWS CLI supports three different kinds of output formats: JSON, tab-delimited text, and ASCII-formatted table (Carty, 2023). The output can be filtered by a developer with a query and pagination behavior modified when the command's task is to retrieve a long list of objects. As a useful tool in CLI, developers can use *help* (Carty, 2023) which at the end of a string suggests options and information for that particular command. The AWS CLI source code is available in GitHub as committed by AWS (Carty, 2023).

### 3.2.11 AWS CloudShell

AWS CloudShell is a web-based shell environment that provides access to a pre-configured virtual machine running Amazon Linux 2. It allows users to easily run and manage command-line scripts and tools within the AWS Management Console without the need for local installations or setup (AWS CloudShell, 2023).

According to the official AWS documentation, CloudShell provides pre-installed tools and utilities commonly used for AWS resource management, such as the AWS CLI, Git, and Python. Users can also install additional tools or dependencies to their CloudShell environment. CloudShell sessions are automatically terminated after 20 minutes of inactivity (AWS CloudShell, 2023).

Referring to AWS FAQs (AWS CloudShell FAQs, 2023) CloudShell does not include any additional charge. If the user uses other AWS resources with Cloudshell to create or run their applications, they pay only for what they use and when they use it. Interacting with CloudShell from a local machine is possible as well, given with an example of the possibility of upload a file to the CloudShell instance home directory through the user's browser from their local machine.

As what comes to differences between CloudShell and EC2 Instance Connect, EC2 Instance Connect only lets the user to connect to existing EC2 instances in their account. CloudShell however does not require any resources in the user's account. (AWS CloudShell FAQs, 2023). To put this simply, while EC2 Instance Connect is great for connecting to existing EC2 instances, CloudShell is suitable from running AWS CLI commands and is suitable for general scripting.

### 3.3 Python

Python is a programming language planned to be used in this thesis' practical step. It is an interpreted, object-oriented, high-level programming language with dynamic semantics (What is Python? n.d.). AWS is also highly compatible with different programming languages because Python has an extensive library for different usages. They are also non-chargeable for all major platforms and are freely distributed (What is Python? n.d.). According to Courser's Beginner Guide (Coursera, 2023) "Python is commonly used for developing websites and software, task automation, data analysis, and data visualization". The same guide includes a Stack Overflow 2022 Developer Survey, which reveals that "Python is the fourth most popular programming language, with respondents saying that they use Python almost 50 percent of the time in their development work".

Even though Python is the claimed as the world's fourth most used programming language (Coursera, 2023), it is most often used for the areas not seen by the consumers and users, such as the back end of a website or application. The back end for these might include sending data to and from servers, processing data and communicating with databases.

As for why Python is deemed as a suitable scripting language for this, the Coursera guide explains that "If you find yourself performing a task repeatedly, you could work more efficiently by automating it with Python" (Coursera, 2023). As the goal with scripting in this thesis is to build an automated solution for cost clarification, Python is a great option for this task also since it has a simple, easy to learn syntax.

### 3.4 Boto3

According to Real Python Tutorial (Bolovan, 2023) Boto3 is the name of the Python SDK for AWS, SDK meaning software development kit. “It allows you to directly create, update, and delete AWS resources from your Python scripts. For Boto3’s features, according to the official AWS documentation (AWS SDK for Python (Boto3), 2023) boto3 “has two distinct levels of APIs”. One is Client (low-level) which provide mappings to the underlying HTTP APIs. The other is Resource, which hides explicit network calls. Resource however instead provides “resource objects and collections to access attributes and perform actions.” (AWS SDK for Python (Boto3), 2023).

One feature also which Boto3 comes with is Waiters. They automatically poll for existing, pre-defined status changes in all the AWS resources. Examples of these are when the Running-state of a EC2 instance is checked with a waiter, or when a new DynamoDB, meaning Dynamo Database, table is created, and the waiter is set to check when it will be in use. (AWS SDK for Python (Boto3), 2023). Waiters are available from both distinct levels of APIs, Client, and Resource.

What are the differences in the usage? When comparing programmatical workload, Client needs it more. That is because most of the Client operations give a dictionary as a response, which then the user must parse by themselves. In Resource, the SDK does that for the user. (Python, Boto3, and AWS S3: Demystified, 2023).

Although the Resource method has more programmatical support from the SDK, Client might give some performance improvements, but the code becomes less readable than with using Resource. According to Real Python Guide, “Resources offer a better abstraction, and your code will be easier to comprehend.” (Python, Boto3, and AWS S3: Demystified, 2023).



### 3.5 Git

According to an article by Sayeda H. Perveez, (What is Git, 2023) "Git is a DevOps tool used for source code management. It is a free and open-source version control system used to handle small to very large projects efficiently." To put it simply, Git allows to better keep track of different versions of the code and resources, which improves productivity and shareability between different parties. An example situation is presented in the figure below.

In the Figure 8 below is demonstrated how Git could be used. Inside a business organization there are several developers, who share project responsibilities. Since they all have the identical source code on their local systems, Git helps them to keep track of their individual changes (What is Git, 2023). Each developer pushes their code changes to the Git, so they all know who has done what and when. The updated code can then be pulled from the Git Distributed Server to keep updating locally, after which the update is pushed back at Git Distributed Server. All pushes have messages, so they can differentiate their own changes (What is Git, 2023).

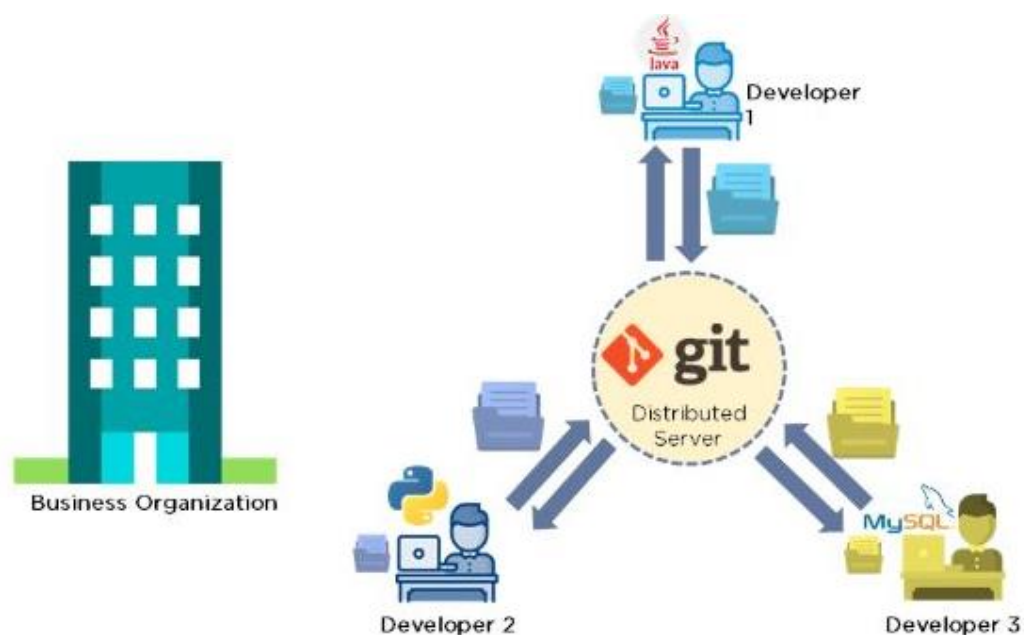


Figure 8. Git example (What is Git, 2023)

## 4 Start of scripting

### 4.1 NDA

Before starting any kind of documentation on what and how the scripting was done, a non-disclosure agreement (NDA) must be mentioned. The information handled by the scripts and this assignment itself is sensitive in nature since they include financial information not only of the assignment company (Qvantel Finland Oy) but also the company's clients. The non-disclosure agreement was concluded with the following individuals: the writer of this thesis, an individual from the company and the thesis director from Jamk University of Applied Sciences in Jyväskylä. Its purpose is to guarantee that any information regarding the assignment will not be revealed to outside personnel.

However, the agreement does not restrict everything regarding the subject. The actual scripting process can still be documented here, just not by the greatest accuracy. Every aspect containing specific financial information about Qvantel and its partners and clients will be censored. In this case, the credibility of the end results will be confirmed by the individuals tied by the NDA. So, if any aspect of the scripts is censored or left out, keep in mind that they originally contained sensitive information. Those sections will be still explained through and mentioned if and why there seems to be missing information.

The final product containing the sensitive information is checked by the assignment company and after that valuated if it is suitable for later use. Though this would require that the desired results explained in the introduction have been achieved.

### 4.2 Practicing

No scripting process is fluent, it will just get easier when the process of learning has been conducted. For the sake of learning more about Python for this thesis process, it was found that doing through trial and error is the best teacher. A goal was set. There needed to be a list of total costs of one account in AWS. This was fortunately quite straightforward since Boto3 acted as a great tool for help. When a single smaller goal was met, new ones were added. Boto3 contains very straightforward documentation on how to use it in Python scripting.

Some unavoidable scripting errors were encountered – like in cases where scripting was done wrong, and the script returns a specific error message. These errors were, however, solved and the thesis process and searching for solutions was able to continue.

### **4.3 Main points of focus**

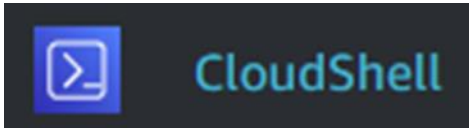
The actual problems need to be thoroughly looked at as well before starting any kind of scripting. Earlier in the introduction a clarity problem was mentioned. The problem is that the savings plans coverage (now connected to one account) are not connected automatically to the correct customers. This means that every report needs to be manually examined to gather the savings plans information for the correct customers.

The second problem is that Cost Explorer has no clear indication how much the monthly customer bills have increased or decreased. This might seem like a secondary problem but reaches much deeper. Of course, if the monthly bill has increased significantly, it will be noticed quite fast and cost optimization work will be conducted after examining what caused the increase. But if the increase in monthly costs happens gradually, only small amounts at a time but at a linear fashion, this might indicate that the cost increase has no plan of slowing down. This might be noticed once the bills have increased significantly even though some cost reduction would have been possible earlier.

As a third problems there is reliability. When the customers want to know how their costs have distributed across different services by time, the information must be shared with reliability. This means that the information available to be shared with the customer must be accurate in its form of presentation. Either as the values, which can then be clarified more with past values to demonstrate cost fluctuation or by presenting the information with the previously mentioned trend analysis way, for example as a chart. For customers not being aware of the actual values of their costs, the visual presentation might be a better option to display how much the costs have changed and where they are distributed.

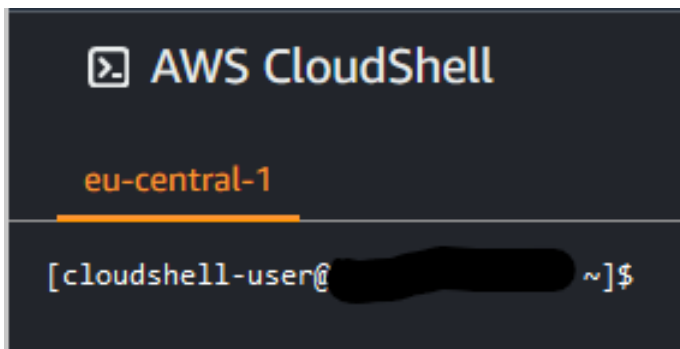
## 4.4 Establishing the scripts

There are many ways in which the scripting can be done and what was chosen was to perform the scripting straight in AWS CloudShell. This avoided the extra working step of setting up AWS CLI on a personal computer or work environment and was the fastest way to access AWS's different APIs, since no extra permissions were needed to establish.



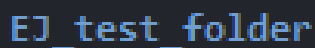
*Figure 9. CloudShell icon*

This next figure illustrates what it looks like inside AWS CloudShell. Do be noted that the blurred section contains an ip-address, which won't be displayed due to sensitivity and restriction reasons. Cloudshell-user is the default username in CloudShell and the ip-address contains the unique session information.



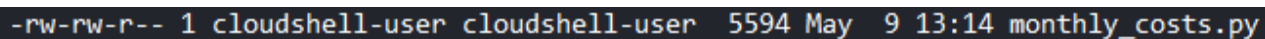
*Figure 10. CloudShell terminal*

During the project, a new folder to store all the files created in the project was needed, so a folder was named after the writer's initials and by the reason for the folders existence.



*Figure 11. Test folder*

The actual script is a file, ran by a command “python3” and the file name. In this project, the script was named `monthly_costs.py` but it can be changed in the future if seen necessary. Below Figure 12 is a screenshot from the command line. The “-rw-rw-r—” typing describes who have permissions to access or use this file and how. The markings r (read), w (write) and x (execute) are assigned to each file and folder and are changed to suit the individual situation. The permissions are declared for three entities: owner, group and public. The first 3-staged section is for the owner of the file. In this case the executing rights have been removed, and only read and write permissions have been enabled. The next section is designated for the group, which means that every user belonging in the same group than the owner has the same rights, in this case the same read and write permissions. The last section is reserved for public, meaning any other user, and is left without permissions. The permissions are an important aspect in information security and are used to limit the access to sensitive and important files or folders. If this seems unclear, an access permissions list (Access Permissions, n.d.) explains everything in more detail.



*Figure 12. Python script file*

## 4.5 The first script

Now that the permissions and ownerships have been handled, the actual scripts need to be examined. Something to be noted is that the scripts displayed here do not contain the specific information about cost values or to which client they belong to. If the scripts need visual explanation, they will be added to the section in question.

The process is gone through step by step by figure at a time. The final script was quite long so it could not fit to only one figure. Also, if the figure holds at places markings “\*\*\*\*\*” or blurred with a block, this means that there was sensitive information falling under the NDA and was therefore

hidden for this analysis. For future clarification and documentation purposes the figures of the script also include comments what each of the script's section does: format is #, and then the sentence.

First, any necessary libraries needed in the script are imported. Following the boto3-documentation (boto3) the client needs to be established to which AWS API to access, in this case Cost Explorer API, after which the timeframe for the script to be gathering data from is determined. The used dates here are the date of today, being 9.5.2023, and the start- and end-dates of the previous month.

```
import boto3
from datetime import datetime, timedelta
import csv
import pandas as pd
import os

#Specify the CostExplorer client
ce_client = boto3.client('ce')

#Get the date of today
now = datetime.now()

#Specify the start of timeperiod to the first day of previous month
start_date = datetime(now.year, now.month -1, 1).strftime('%Y-%m-%d')

#Specify the end of timeperiod to the first day of current month
end_date = datetime(now.year, now.month, 1, 1).strftime('%Y-%m-%d')
```

*Figure 13. Specifying libraries, client and dates*

Next, a dictionary “linked\_accounts” to hold the account IDs and names needs to be established. The account ID is a 12-digit unique number found inside Cost Explorer, but since there is a need for the account cost data to be displayed in the end results with their names, the names must be clarified separately. Also, an empty dictionary to later hold the savings plans coverage -data is established.

```
#Dictionary holding all the account ids and corresponding names
linked_accounts = {
    '*****': '*****',
    '*****': '*****',
    ...
}
#Dictionary to save the savings plan costts
total_cost_by_account = {}
```

Figure 14. Dictionaries

Next, the boto3-method “get\_cost\_and\_usage” (get\_cost\_and\_usage, 2023) is established to retrieve all the different record type -data from Cost Explorer with the previously specified time period. The “time.sleep(1)” is a set function of the time-library which ensures that the Cost Explorer API does not get flooded in calls during the script running process and waits for a second to guarantee this.

```
#Specify the information to get cost of all record_types
for account_id, account_name in linked_accounts.items():
    time.sleep(1)
    result = ce_client.get_cost_and_usage(
        TimePeriod={
            'Start': start_date,
            'End': end_date
        },
        Granularity='MONTHLY',
        Metrics=['UnblendedCost'],
        GroupBy=[
            {
                'Type': 'DIMENSION',
                'Key': 'LINKED_ACCOUNT'
            },
            {
                'Type': 'DIMENSION',
                'Key': 'RECORD_TYPE'
            }
        ],
        Filter={
            'Dimensions': {
                'Key': 'LINKED_ACCOUNT',
                'Values': [account_id]
            }
        }
    )
```

Figure 15. Retrieving record type -data

After the process seen in Figure 11, a loop to gather all the savings plan covered usage per account as “total\_cost\_by\_account” is created. Once one account’s information has been gathered, the loop automatically starts the next account’s data gathering. The gathered savings plan covered usage -data is summed with the corresponding unblended cost of the account.

```
#Get the cost amount of record type SavingsPlanCoveredUsage
results_by_dimension = result['ResultsByTime'][0]['Groups']
for group in results_by_dimension:
    cost = float(group['Metrics']['UnblendedCost']['Amount'])
    record_type = group['Keys'][1]
    if record_type == 'SavingsPlanCoveredUsage':
        total_cost_by_account[account_name] = total_cost_by_account.get(account_name, 0) + cost
```

Figure 16. Savings Plan Covered Usage

The process is then repeated with the total cost of each account without the savings plan -filter. Two different queries to the Cost Explorer API were needed, since the unblended cost of each account has different filters than the savings plan covered usage. (get\_cost\_and\_usage, 2023)

```
#Specify the information for the unblended costs
for account_id, account_name in linked_accounts.items():
    result = ce_client.get_cost_and_usage(
        TimePeriod={
            'Start': start_date,
            'End': end_date
        },
        Granularity='MONTHLY',
        Metrics=['UnblendedCost'],
        GroupBy=[
            {
                'Type': 'DIMENSION',
                'Key': 'LINKED_ACCOUNT'
            },
        ],
        Filter={
            'Dimensions': {
                'Key': 'LINKED_ACCOUNT',
                'Values': [account_id]
            }
        }
    )

    #Get the unblended total cost for each account
    results_by_dimension = result['ResultsByTime'][0]['Groups']
    for group in results_by_dimension:
        cost = float(group['Metrics']['UnblendedCost']['Amount'])
        total_unblended_cost_by_account[account_name] = cost
```

Figure 17. Total cost of account



As mentioned earlier, the savings plans were all billed from one account, therefore not giving correct total cost information on the other accounts. Here in Figure 14 the savings plan covered usage of each account is subtracted from the specific account's total cost. The syntax “-=” subtracts the values and saves the “\*\*\*\*\*\_unblended\_cost” with the new value.

```
#Subtract the total cost of all linked accounts from the unblended total cost of ***** account
*****_unblended_cost = total_unblended_cost_by_account.get('*****', 0)
for account_name, total_cost in total_cost_by_account.items():
    if account_name != '*****':
        *****_unblended_cost -= total_cost
```

*Figure 18. Subtracting the account costs*

Then a list of two dictionaries is established: one holding the subtracted account's cost data and the other to contain the other accounts' data. First in the dictionary the content name is established and then the content itself by the name. In this case, the first dictionary has the subtracted account's name and its total unblended cost content. Since the used value is an existing value for 'Cost' writing it manually is not desirable, the “\*\*\*\*\*\_unblended\_cost” can be inserted there. The f-letter before the insertion makes it possible to add other variables to the cost information, in this case the dictionary value is wanted to be displayed by 2-decimal accuracy (:.2f). The other dictionary is left empty at this stage since the cost for the rest of the accounts is not yet accurate.

```
#List of dictionaries to write to a CSV file
csv_data = [
    {'Account': '*****', 'Cost': f"*****_unblended_cost:.2f"},
    {'Account': '', 'Cost': ''}
]
```

*Figure 19. List of dictionaries*

The savings plan covered usages are then added to the right accounts' total costs, skipping the previously data containing account, since the information that was just subtracted should not be added back in. The for-loop here checks each account in the order they are set in the account ID-list. When the account not wanted to be summed with savings plan covered usage -data again comes, the loop skips that and does nothing to its values, then continues with the other accounts. In this case that specific account was listed first in the first account ID-s and names -list, so after

this skip the loop sums the corresponding savings plan costs to the right accounts. When the accounts are finished, the data is then appended to the previously created list of dictionaries “csv\_data”.

```
#Skip ***** account to avoid false amounts and add the Saving Plans costs to correct account
for account_name, total_cost in total_unblended_cost_by_account.items():
    if account_name == '*****':
        continue
    if account_name in total_cost_by_account:
        total_cost += total_cost_by_account[account_name]
    csv_data.append({'Account': account_name, 'Cost': f"{total_cost:.2f}"})
```

*Figure 20. Correcting the costs*

To display the data in a readable way, this gathers the list “csv\_data” to a pandas Data Frame. During the script testing, the CSV-file showed an empty row between the originally savings plan data holding account and the rest of the accounts. This is because the account and cost adding is done in two different methods and times. The empty row does not have any functional negativity, it just affects to the way the data is displayed, so this was to be fixed.

```
#Creating a DataFrame from the data
df = pd.DataFrame(csv_data)
```

*Figure 21. Creating the DataFrame*

An index after the account is selected (the empty row). The index is marked as empty after which the right columns ‘Account’ and ‘Name’ are defined.

```
#Determine the index to drop later
index_to_drop = df[df['Account'] == '*****'].index + 1

#Determining to drop the empty row
df = pd.concat([df.drop(index_to_drop), pd.DataFrame([''] * df.shape[1], columns=['Account', 'Cost'])])
```

*Figure 22. Dropping the empty row*

For the Data Frame to be saved in a CSV-file, some formatting was seen necessary. For this, pandas' "to\_csv" method can be used to move the data frame to the CSV-file, in which the formatting options for the CSV is put in parenthesis. The 'index=False' ignores the index-column, therefore removing the numbers from each row. The encoding 'utf-16' ensures that the CSV-file-format is suitable to be opened in other apps, including Excel. And the separations '\t' is a syntax for the Tab-character to be used as column separators. This was used also to force the columns to their own cells in Excel.

```
#Converting the dataframe to a CSV file and uploading it to S3
csv_buffer = df.to_csv(index=False, encoding='utf-16', sep='\t').encode()
```

*Figure 23. CSV-file formatting*

For the reports to follow a unified naming style, the current date of the script run was deemed suitable. For this the date of today is used and inserted in "file\_name".

```
#Date-str used in file naming
date_str = now.strftime('%Y-%m-%d')
```

*Figure 24. Date of today*

```
file_name = f"{date_str}_monthly_costs.csv"
```

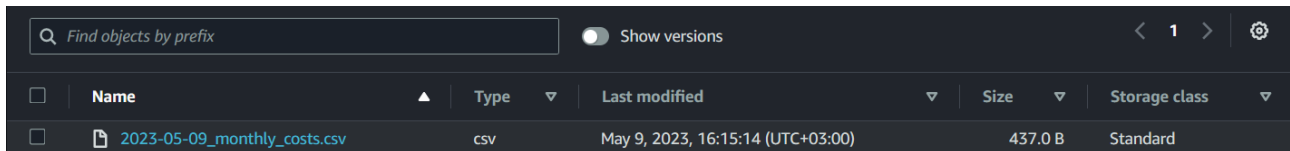
*Figure 25. File name*

For the report storing purposes, a new S3-bucket was created, and the name defined in the script. Boto3 includes a ready function for an automatic S3-bucket upload of the file when the script is run (boto3). In the "put\_object" -function the bucket name, file name and the contents of the file are set in parenthesis. The S3-bucket was created in AWS separately as Standard-class and was set to only allow access from certain individuals. The bucket itself is owned by the root-account.

```
#S3 bucket name
bucket_name = 'monthly-cost-reports-*****'

s3 = boto3.client('s3')
s3.put_object(Bucket=bucket_name, Key=file_name, Body=csv_buffer)
```

Figure 26. S3-bucket file upload

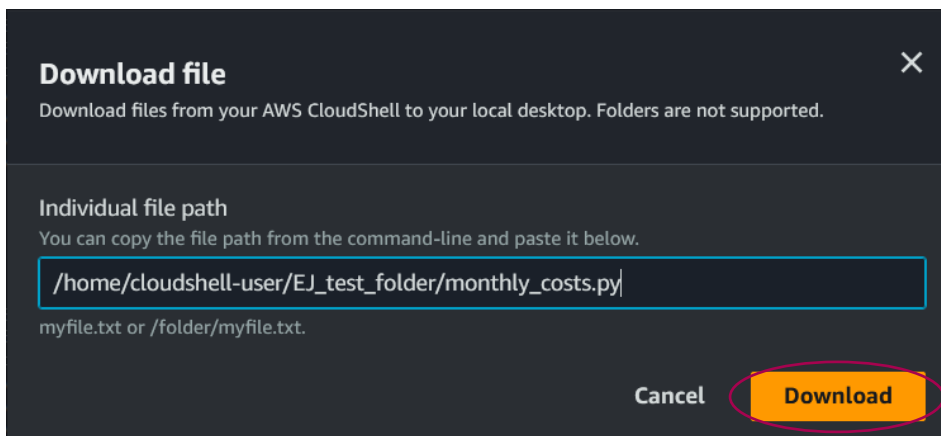


The screenshot shows the AWS S3 console interface. At the top, there is a search bar labeled 'Find objects by prefix' and a toggle for 'Show versions'. Below this is a table listing the objects in the bucket. The table has columns for 'Name', 'Type', 'Last modified', 'Size', and 'Storage class'. One object is listed: '2023-05-09\_monthly\_costs.csv' with a type of 'csv', last modified on 'May 9, 2023, 16:15:14 (UTC+03:00)', size of '437.0 B', and storage class of 'Standard'.

Name	Type	Last modified	Size	Storage class
2023-05-09_monthly_costs.csv	csv	May 9, 2023, 16:15:14 (UTC+03:00)	437.0 B	Standard

Figure 27. A successful file upload to S3

For the script to be used later for development purposes by other personnel, it needed to be downloaded from AWS. Luckily, CloudShell has a simple user interface (UI) and therefore only requires the full directory-path to the file for the download to succeed.



The screenshot shows a 'Download file' dialog box. It has a title bar with a close button (X). The main text says 'Download files from your AWS CloudShell to your local desktop. Folders are not supported.' Below this, there is a section titled 'Individual file path' with the instruction 'You can copy the file path from the command-line and paste it below.' A text input field contains the path '/home/cloudshell-user/EJ\_test\_folder/monthly\_costs.py'. Below the input field, there is a hint 'myfile.txt or /folder/myfile.txt.' At the bottom right, there are two buttons: 'Cancel' and 'Download'. The 'Download' button is highlighted with a red oval.

Figure 28. Downloading the file

## 4.6 The second script

The second script was meant to calculate monthly cost changes for the corrected monthly total costs of each account during the last three months. However, the script was not finished and therefore not added here for two reasons. The comparing section of the script would not work, since the cost values are in string-format and could not be changed to a float-format, a format to which calculations can be done. The reason for the format change -error was not solved.

The second reason is that it turned out simply too long for this document. The only functioning way the same cost information three times with three different time windows were to be achieved was to repeat the process three times. To put it simply, the script contains the same process three times with slightly distinctive characteristics. This makes the script long in processing-time, but also easily uses up the threshold of Cost Explorer calls. Even with the shorter script, `time.sleep(1)` needed to be added to give the script a second of breathing room between calls. The same results must be achieved in a more efficient way and with a cleaner and more understandable script.

However, this does not mean the longer script has been scrapped entirely. It achieved gathering the correct total cost information from different time periods, a job half done. This just means that the script can be further developed in the future. For this to succeed, two broad plans of development were conducted. This was fairly straightforward, since all the wanted results of the script were already established. The plans of development give only a guideline direction to what must be done for the wanted results to be achieved. They do not include the specifics; those are left for the developer to decide.

## Plan of development: Monthly cost change analysis

The current script gathers cost information from three different months, but doesn't include any kind of comparison aspect.

### Task:

Develop a comparison aspect by percent between the different time periods for each account. This could also include the top three most cost affecting services to the change. The script logic could also be changed to shorten the processing times and send an email notification of successful run.

### Steps:

- add the percentage change section
- list the most cost affecting services
- add an email notification section
- analyze if the script could be "cleaned up"

The further development of this script is needed in order to recognize any aberrancies in cost changes and to recognize any potential rising costs.

*Figure 29. Plan of script development*

## Plan of development: Generating an automatic cost report

For each wanted script to be used in cost analysis, an automatic report creation has to be conducted.

### Task:

Create a new Lambda function to run the existing Python scripts for cost calculations. Trigger the lambda function each month by using EventBridge.

### Steps:

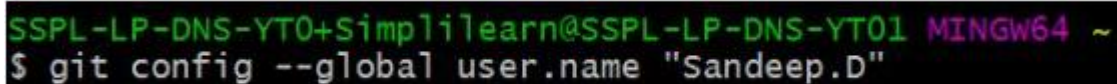
- Create a new Lambda function, add the existing code.
- Set the necessary IAM-permissions for the function to gain access to different API's.
- Create a Lambda trigger using EventBridge to generate the report each month.
- Confirm that solution works as intended.

*Figure 30. Plan of automation development*

## 4.7 Final implementations

As the last step for the scripting process, the script, and the report it generates, need to be stored somewhere. Since the whole scripting was done in CloudShell via a root account (an account with all the privileges and accesses) there was no need to set up any permissions or rules in AWS, storing the report to S3 was concluded straight from the script itself. As for the script, it was pushed via Git to a company owned repository. The repository can be accessed by team or personnel when the script needs to be used later or developed further using the plan of development points.

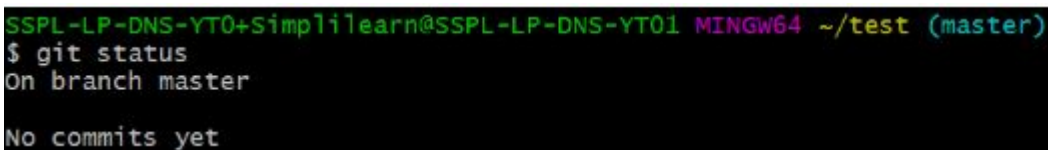
The repository was created by personnel in the company as a new, empty repository. Using the command 'git clone (address to the remote repository)' the repository is then cloned to the local system. The user configuration was done in advance since Git has been used at work for other projects. Figure 27, which from Sayeda. H. P's tutorial (Perveez, 2023), displays the example configuration.



```
SSPL-LP-DNS-YT0+SimpliLearn@SSPL-LP-DNS-YT01 MINGW64 ~  
$ git config --global user.name "Sandeep.D"
```

Figure 31. Git config (Perveez, 2023)

After every step it is good practice to type the 'git status' command. This keeps the user on track of all the current changes (Perveez, 2023).



```
SSPL-LP-DNS-YT0+SimpliLearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)  
$ git status  
On branch master  
  
No commits yet
```

Figure 32. Git status (Perveez, 2023)

After the repository has been set up, it is time to add the files the user wants to push, in this case the created python script. The below figure displays the syntax for the command. (Perveez, 2023).

```
git add <file>...
```

*Figure 33. Git add (Perveez, 2023)*

For the repository's files' changes to be kept track of, using 'git commit' to add a message for push is good practice. Using a different message, maybe why the push was made or what is the file for gives a good indication of its existence. (Perveez, 2023)

```
$ git commit -m "committing a text file"
[master (root-commit) 1c0673b] committing a text file
1 file changed, 2 insertions(+)
create mode 100644 info.txt
```

*Figure 34. Git commit (Perveez, 2023)*

Before pushing the file to the remote repository, the local repository needs to be connected to the remote one. After this is done, typing 'git push origin master' pushes the file in the origin (the local repository clone) to the master (the remote repository). (Perveez, 2023)

```
$ git remote add origin https://github.com/simplilearn-github/test.git
```

*Figure 35. Git add origin to remote (Perveez, 2023)*

```
$ git push origin master
```

*Figure 36. Git push (Perveez, 2023)*

After the push, the repository page should display the just pushed file after the page has been re-freshed. Sometimes, the repositories might include a file called 'README.md'. This file is created to be containing user manual of the repository's contents.



In this case, the python script written was pushed to the remote master repository, and a 'README.md' file was created. This file contains instructions on what the script does and how to use it. This is very important if the personnel accessing the repository do not have the script instructions from any external source at hand.

## 5 Results and notifications

The developed script focuses on correcting the existing miscalculations of all the existing accounts' total costs. As was mentioned in the goals of this thesis, the saving plans coverage was not integrated to the right accounts, therefore generating wrong cost data each month which was then corrected manually by a company employee. This script achieves the right information by negating the manual calculations after the report has been generated. As the data generated with the script contains client cost data, it will not be displayed here. The report contents were verified as correct by first manual checking by the thesis writer and afterwards the assigner employee from the company. To avoid the script miscalculating cost values, a manual check was conducted several times from several different reports. This ensured that if the values match with each other and to those from Cost Explorer, the script has successfully achieved its task.

The working script was saved for the company's internal use and the cost report was set in the script to be saved in the new S3-bucket. The S3-bucket was set to be accessed only by the root account by the thesis assigner's request. This ensures, that the reports holding sensitive information can be accessed by only certain personnel when needed.

The script generated report is displayed in a CSV-file with two columns: Account and Cost. The Account column, suggesting by its name, makes a list of all accounts listed in the script. The script then connects the corrected costs to their right places and sets them in the Cost-column next to their corresponding account. The cost also includes marking "USD" clarifying to the reader that the values are indeed in U.S dollars. To guarantee an easy reading experience for the report, it was also tested in Excel, not just straight from CloudShell. The column separation works, setting the columns to their individual cells and the column contents directly under them.

## 6 Discussion

I was set with two goals: fixing the wrong cost data and conducting an automatic change comparison between monthly costs to examine any potential risks in costs increasing too much. The first goal was met, the other was not. As mentioned earlier in the scripting process explanations, the second script needed for the change examination could not be conducted for numerous of reasons. These could be the ones explained in the scripting process: incomplete file type changes, script format and layout being unnecessarily long and so forth.

The automation of the process was not entirely achieved either since the script running was not automated with Lambda or timed by EventBridge. This would have required extra steps to be taken after the scripting process. The documentation of Lambda and EventBridge was however done in an easy-to-understand way in the Terminology-section, which could prove to be helpful when the time comes to further develop the existing implementation. The implementation development is given a broad direction with the second plan of development. As for the trend analysis aspect, it was left unfinished as well. Following the visual presentation trends, a pie chart or diagram of sorts can be conducted for extra presentation clarity. Those can easily be done for example from the cost data in Excel.

Even though not all the goals were met, the final product may serve still as a good way to demonstrate to clients their costs with accuracy and reliability. What really matters is the big picture, in this case not so much the underlying statistics.

As for the aspects left not achieved, those can be further developed in the future. The documentation in its entirety should guarantee a clear understanding quite well of the whole process and may potentially serve as a suitable subject for someone else's thesis assignment.

## List of references

Access Permissions. N.d. unixintro. Accessed on 26 April 2023. Retrieved from [https://www.cis.rit.edu/class/simg211/unixintro/Access\\_Permissions.html](https://www.cis.rit.edu/class/simg211/unixintro/Access_Permissions.html)

AWS Benefits. N.d. AWS. Accessed on 25 January 2023. Retrieved from <https://aws.amazon.com/application-hosting/benefits/>

AWS CLI. N.d. AWS Official. Accessed on 29 April 2023. Retrieved from <https://aws.amazon.com/cli/>

AWS CloudShell. 2023. AWS. Accessed on 29 April 2023. Retrieved from <https://docs.aws.amazon.com/cloudshell/latest/userguide/welcome.html>

AWS CloudShell FAQs. 2023. AWS. Accessed on 12 May 2023. Retrieved from <https://aws.amazon.com/cloudshell/faqs/>

AWS Free Tier. N.d. AWS. Accessed on 16 April 2023. Retrieved from [https://aws.amazon.com/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=\\*all&awsf.Free%20Tier%20Categories=\\*all](https://aws.amazon.com/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all)

AWS SDK for Python (Boto3). N.d. AWS. Accessed on 11 May 2023. Retrieved from <https://aws.amazon.com/sdk-for-python/>

AWS S3 bucket. N.d. AWS Docs. Accessed on 29 April 2023. Retrieved from <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html>

Availability Zones. N.d. Regions and Zones. Accessed on 30 January 2023. Retrieved from <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html#concepts-availability-zones>

Billing Console. N.d. AWS. Accessed on 25 February 2023. Retrieved from <https://aws.amazon.com/aws-cost-management/aws-billing/>

Bolovan, R. 2012. Python, Boto3, and AWS S3: Demystified. Real Python. Accessed on 12 May 2023. Retrieved from <https://realpython.com/python-boto3-aws-s3>

Carty, D. 2023. AWS Command Line Interface. TechTarget. Accessed on 29 April 2023. Retrieved from <https://www.techtarget.com/searchaws/definition/AWS-Command-Line-Interface>

C City. 2020. 3 Business Benefits of Technical Documentation. Emazzanti Technologies. Accessed on 14 May 2023. Retrieved from <https://www.emazzanti.net/technical-documentation/>

Coursera. 2023. What Is Python Used For? A Beginners Guide. Coursera. Accessed on 11 May 2023. Retrieved from <https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python>

Gillespie-Lord, E. 2023. What is Scripting and what is it used for? Bootcamps by BestColleges. Accessed on 19 April 2023. Retrieved from <https://www.bestcolleges.com/bootcamps/guides/what-is-scripting/>

get\_cost\_and\_usage. 2023. Boto3 documentation. Accessed on 10 May 2023. Retrieved from [https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/ce/client/get\\_cost\\_and\\_usage.html](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/ce/client/get_cost_and_usage.html)

How are Amazon EC2 instance-hours billed? N.d. AWS re:Post. Accessed on 16 April 2023. Retrieved from <https://repost.aws/knowledge-center/ec2-instance-hour-billing>

Jain, A. 2023. Amazon Simple Notification Service. K21Academy. Accessed on 29 April 2023. Retrieved from <https://k21academy.com/amazon-web-services/amazon-sns/>

Lambda. 2023. Amazon Web Services. Accessed on 12 April 2023. Retrieved from <https://aws.amazon.com/lambda/>

Local Zones. N.d. Regions and Zones. Accessed on 30 January 2023. Retrieved from <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html#concepts-local-zones>

Outposts. N.d. Regions And Zones. Accessed on 30 January 2023. Retrieved from <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html#concepts-outposts>

Perveez, S. H. 2023. What is Git: Features, Command and Workflow in Git. simplilearn. Accessed on 10 May 2023. Retrieved from <https://www.simplilearn.com/tutorials/git-tutorial/what-is-git>

Regions and Zones. N.d. Amazon Elastic Compute Cloud. Accessed on 30 January 2023. Retrieved from <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html>

Regions. N.d. Regions and Zones. Accessed on 30 January 2023. Retrieved from <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html#concepts-regions>

REST API. N.d. What is a REST API? Red Hat. Accessed on 29 April 2023. Retrieved from <https://www.redhat.com/en/topics/api/what-is-a-rest-api>

Saving Plans Billing. N.d. AWS Official. Accessed on 16 April 2023. Retrieved from <https://repost.aws/knowledge-center/savings-plans-billing>

Saving Plans. N.d. AWS. Accessed on 28 January 2023. Retrieved from <https://aws.amazon.com/savingsplans/>

Sontam, V. 2022. Cost Optimization Practices on AWS for Telco BSS Workloads. AWS for Industries. Accessed on 15 May 2023. Retrieved from <https://aws.amazon.com/blogs/industries/cost-optimization-practices-on-aws-for-telco-bss-workloads/>

Tags. N.d. AWS Docs. Accessed on 28 January 2023. Retrieved from <https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/cost-alloc-tags.html>

What is Amazon EC2. N.d. AWS Docs. Accessed on 16 April 2023. Retrieved from <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>

What is Amazon EventBridge? 2023. AWS Docs. Accessed on 12 April 2023. Retrieved from <https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-what-is.html>

What is AWS. N.d. Amazon Web Services. Accessed on 25 January 2023. Retrieved from <https://aws.amazon.com/what-is-aws/>

What is Cost Management. N.d. AWS Docs. Accessed on 25 January 2023. Retrieved from <https://docs.aws.amazon.com/cost-management/latest/userguide/what-is-costmanagement.html>

What is Python? N.d. Executive Summary, Python. Accessed on 30 January 2023. Retrieved from <https://www.python.org/doc/essays/blurb/>

Zhang, M. 2023. Top 10 Cloud Service Providers Globally in 2023. Dgtl Infra. Accessed on 19 April 2023. Retrieved from <https://dgtlinfra.com/top-10-cloud-service-providers-2022/>