City, University of London

BSc Computer Science with Placement year

Auditing tool for AWS machine instances and images

A web app made to audit the age of AWS resources. It colour codes the age of a resources, lets you filter by IDs, and sort by date. The web app operates entirely on AWS cloud services, making it scalable and highly available.

By Irina-Maria Fratila irina-maria.fratila@city.ac.uk

Consultant name: Dr. Laure Daviaud

Table of Contents

Link to the auditing tool: ***4***

Chapter 1: Introduction ***4***

1.1 Problem solved **4**

1.2 Project objectives **4**

1.2.1 Main objective: 5

1.2.2 Sub objectives: 5

1.3 Anticipated beneficiaries **5**

1.4 Work conducted to meet objectives **5**

Chapter 2: Output summary ***6***

Chapter 3: Literature review ***9***

3.1. Cloud computing benefits and knock-on-effects on enterprise operations **9**

3.2. Technology and services to be used **10**

3.3. Similar applications on the market **11**

Chapter 4: Method ***12***

4.1 Software development methodology **12**

4.2 Work plan **12**

4.2.1 Initial plan 12

4.2.2 Application of work plan and adjustments made to project/plan 14

4.4 Design decisions and documentation **14**

4.5 Implementation of system components and version control **16**

4.5.1 Website client-side files and hosting 16

4.5.2 API gateway and request handler 17

4.5.3 Data processing job 17

4.5.4 Data storage 17

4.5.5 Security: Identity and access management 18

4.6 Testing **18**

Chapter 5: Results ***19***

5.1 Architecture diagram **19**

5.2 Website static files **20**

5.3 Publishing the website to the public domain **22**

5.4 API gateway **23**

5.5 Request handler **24**

5.6 Data processing job and data generated **26**

5.6.1 Python script in AWS Lambda 26
5.6.2 Data generated 28

5.7 Data storage 30

5.8 Security: Identity and access management 30

5.9 Testing 32

Chapter 6: Conclusions and discussions 34

6.1 Objective review 34
6.1.1 Main objective 34
6.1.2 Sub objectives 34

6.2 Reflections and what I learnt 35

6.4 Conclusion 35

Glossary 36

Reference list 37

Appendices 39

Appendix A 39

APPENDIX B: Reuse summary 48
Appendix B1: index.html 48
Appendix B2: script.js 48
Appendix B3: auditingtool-updateInstancesAndImagesBuckets.py 48
Appendix B4: website-request-handler.py 48

APPENDIX C: Source code 49
APPENDIX C1: web-app/ index.html 49
APPENDIX C2: web-app/ script.js 55
APPENDIX C3: python-scripts-from-awsLambda/auditingtool-updateInstancesAndImagesBuckets.py 59
APPENDIX C4: python-scripts-from-awsLambda/ website-request-handler.py 62
APPENDIX C5: Data generated 63

APPENDIX D: Infrastructure diagram 65

Link to the auditing tool:

http://auditing-report-tool.s3-website-us-east-1.amazonaws.com/

Please bear in mind there is a known bug. When opening the website for the first time no data will populate the table. Refreshing the page after 30-45 seconds should fix it, alternatively, trying a different browser. I will go into this error in Chapter 5.3.

Chapter 1: Introduction

Before exploring the project's purpose, development, and decisions, it is important to clarify the "client" I will mention throughout. The relationship between myself and the "client" is informal and any interactions with them were nonofficial. In Chapter 1.2 where I discuss project objectives, there is mention of client requirements for the final product. The client did give me a set of suggestions and loose targets, but they merely provided me with the idea, they were not involved in any final decisions or formal feedback meetings. I had no access to confidential data, tools, or resources, only to their expertise for two feedback meetings along the project timeline. It is worth mentioning the informal client, which is NaturalMotion Ltd., they are the company I had my industrial placement with, and sometimes I refer to it as the "placement company" in the report.

1.1 Problem solved

This project explores the development of an auditing tool for AWS (Amazon Web Services) resources based on a set of given technical requirements. Various information about the currently running machine instances and images are displayed on a web application, easy to read and manipulate for maximum efficiency and ease of use.

This tool provides granular access to the instances and AMI (Amazon Machine Image) data, while being highly available and with minimal maintenance required by the user. This project's requirements were given to me by the client, and one of the gaps this filled in real time data. Their existing reporting tool were generated once a day, while the auditing tool I built will always give the latest data to the millisecond.

1.2 Project objectives

Before exploring the given technical requirements in detail, it is important to mention changes from the Project Proposal Document. In the document's section "Project objectives", I describe a specific methodology for a particular feature because that was the instruction given by the client. After having a meeting with the client at the beginning of March it was then clarified to me that the instructions given were a suggestion, and the requirements could be delivered in whichever manner

I found best suitable. With that, I will go through the objectives I followed and delivered after that meeting.

1.2.1 Main objective:

Create a website that will display information about currently running AWS instances and images. The data displayed must be up to date, be able to filter by ID or name, and clearly indicate the age of a resource with colour coding. For the main objective to be complete, the sub objectives below must be satisfied.

1.2.2 Sub objectives:

1. A data processing job must collect the latest data about running EC2 instances and user made AMIs (Amazon Machine Images) and store it.
2. The information extracted by the job above must be displayed on the website clearly with indicative colour coding by the date launched, and filtering options for all the data tables.
3. The application must be easy to maintain, scalable and as cheap as possible to run.

The requirements given are quite broad, which gave me the flexibility to explore different solutions and optimise as much as possible. The difference between the goals above and the ones in my PDD is that there is no mention of particular software to be used, but the final goal is unchanged.

Fortunately, all the objectives above, and subsequently the main objective, was met. In chapters 4 and 5 I will elaborate on the decisions taken to satisfy those requirements.

1.3 Anticipated beneficiaries

The client’s server team will benefit from this tool by being able to closely monitor the age of their AWS resources. One of the reasons for keeping machine instances running for less than 30 days is security. Keeping them and their images fresh reduces the possibility of any security vulnerabilities to cause any damage. Being able to see clearly marked with red any resources older than 30 days also helps eliminate the number of forgotten instances that have been running for longer than intended.

1.4 Work conducted to meet objectives

All the objectives mentioned above have been successfully completed, along with access and identity management on the application’s infrastructure. To understand AWS’s resources and pricing to build this application, extensive reading has been performed on the official AWS documentation.

Chapter 2: Output summary

2.1. Website client-side scripts	
Description:	The client-side HTML & JavaScript scripts which display the website interface and implement its functionality. The respective files are index.html and script.js.
Type:	Source code located in the web-app folder, files index.html and script.js Index.html: 185 lines (with comments), where 23 lines are adapted from an external source (referenced in both the source code and following chapters). It uses the libraries jQuery, moment.js and Bootstrap. Script.js: 133 lines (with comments), where 69 lines are adapted from an external source (referenced in both the source code and following chapters). It uses the library imported in index.html moment.js and jQuery.
Use:	The two static website files are uploaded to an AWS S3 bucket, where AWS hosts the website and makes it available on the public domain. In index.html, the call to the API is made and the data to be displayed on the page is requested and displayed.
Recipient:	Author, other researchers, and developers.
Link:	4.5.1. 5.2

2.2. Data processing job	
Description:	A python script running on AWS's Lambda service. The script is called auditingtool-updateInstancesAndImagesBuckets.py, it gathers all the instance and images data of the current AWS account, and it stores it in separate S3(AWS's Simple Storage Service) buckets. The script also updates the data currently stored to keep it up to date.
Type:	Source code using Python 3.7 and the library Boto3. Located in the folderpython-scripts-from-awsLambda/ auditingtool-updateInstancesAndImagesBuckets.py. The script is 104 lines (with comments), where 11 lines are reused. Their sources are credited in the source code itself and following report.

Use:	The Python script auditingtool-updateInstancesAndImagesBuckets.py is attached to a Lambda function of the same name. This function is invoked every time there is a GET request to the API between the website and the request handler script.
Recipient:	Author, other researchers, and developers.
Link:	4.5.3, 5.6.1

2.3. API and request handler	
Description:	The API is the communication medium between the website front end and the back-end scripting. A GET request will come through the API and it will be forwarded to the Lambda function “website-request-handler”. The Lambda function has a Python script attached to it which updates the data stored by calling the “auditingtool-updateInstancesAndImagesBuckets” function mentioned above. After that it retrieves the instance and image data from their respective S3 buckets and sends it back through the API to the website.
Type:	The API was built using AWS’s service “API Gateway” and the script attached to the Lambda function “website-request-handler” is named website-request-handler.py. The script is built using Python 3.7 and the library Boto3. It can be found in python-scripts-from-awsLambda website-request-handler.py. It is 46 lines long with comments. s
Use:	The API is used to retrieve the most up to date data from storage and pass it to the website scripts to be displayed. The Lambda function which processes the request will in turn update the data in store, get it, and package it to be displayed on the website.
Recipient:	Author, other researchers, and developers.
Link:	4.5.2, 5.4, 5.5

2.4. Datasets generated	
Description:	There are two data sets generated in the form on JSON files, each located in their respective S3 buckets. They are both generated and updated in the Lambda function “auditingtool-updateInstancesAndImagesBuckets”. The file “custom-amis-data.json” stores various information about user made machine images, and “running-ec2-instances.json” contains important data about the currently running instances on the AWS account.

Type:	Two JSON files automatically generated by the Lambda function auditingtool-updateInstancesAndImages, each stored in their respective buckets: “running-instances-data-if1” and “custom-amis-data-if1”.
Use:	The two files are accessed to display their contents on the website. They are also updated before they are to be displayed on the website to ensure the data is up to date.
Recipient:	Author, other researchers, and developers.
Link:	5.6.2

2.5. Data storage & Website hosting	
Description:	The data storage and website hosting are using the same AWS service, that is S3(Simple Storage Service). The buckets “custom-amis-data-if1” and “running-instances-data-if1” store the respective data about machine images and running instances in JSON format. The bucket “auditing-report-tool” is available to the public and is hosting the website.
Type:	AWS’s S3 storage buckets created manually through the AWS web management console. Regarding the website hosting, S3 is able to host static websites in a server-less manner. That is, the service will maintain and host the software and hardware needed for the website hosting, only using my uploaded client-side scripts to create the pages.
Use:	The two storage buckets are called when displaying the data on the website and updating their contents. The bucket which hosts the website is publicly available and can be accessed anytime by anyone.
Recipient:	Author, other researchers, and developers.
Link:	4.5.1, 4.5.4, 5.3, 5.7

Chapter 3: Literature review

The following chapter references various literature to understand the context in which the auditing tool will operate. It includes a thorough review of the cloud computing industry, what benefits does it offer to clients and its knock-on-effects on business operations. I also specifically explore Amazon Web Services, how have containers become key in the deployment of reliable services and why use a tool like Kubernetes to manage those containers. Finally, I briefly cover other existing alternatives to my tool and why they may not be feasible to my client.

3.1. Cloud computing benefits and knock-on-effects on enterprise operations

Cloud computing is the on-demand availability of computing power, data storage and various other services. It typically uses a “pay as you go” model where the user only pays for what resources they use, often reducing capital costs.

When discussing cloud computing benefits, I will look at the main elements organisations make in comparison to their existing IT resources. Venters and Whitley suggest that there are five categories vendors look at when considering cloud computing: security, availability and latency equivalence to current infrastructure, variety in services, abstraction of maintenance and scalability (Venters and Whitley, 2012).

Equivalence is the first criterion cloud computing has to satisfy to be considered by clients. Security of data, availability of their servers and response times must at least be as good as on-premise IT infrastructure. Variety refers to the wide range of services made available by a cloud provider and their flexibility, for example, Amazon Web Services offer 200 services across 31 regions, from California in the USA to South Korea’s Seoul. (AWS, n.d.).

Abstraction in cloud computing is to hide the complexities of managing IT infrastructure and software from the user. This idea lies on a spectrum, from AWS’s EC2 instances where the user has access to a virtual machine and its computing power to run any software, to AWS’s Lambda service. Labda is a serverless computing service where the user uploads the source code and names triggers, AWS takes care of all physical and software infrastructure, only running the code in response to the specified triggers. The abstraction of managing physical hardware, power and cooling, backups of data, redundancy, trained staff required and security of the building itself are significant factors for cost savings(Venters and Whitley, 2012).

The next category is scalability, which also ties into the “pay as you go” model. This feature is the heart of cloud computing, the ability to rapidly increase or decrease computing power in response to demand, only paying for what you use. A problem identified by Venters and Whitley in an interview with an IT service director was the over-provisioning of IT hardware, “they look at a project and implement enough IT infrastructure to support that project for up to three years. And typically you might find that only 50% of that capacity is used up until the third year of the project. It's spent two years sitting around doing nothing" (Venters and Whitley, 2012). Cloud computing, more specifically the elastic cloud eliminates this problem severely, making computing power easy to access and scale. The upfront cost of developing an application or service is significantly lower, allowing smaller enterprises and governments with limited capital to succeed and develop reliable products.

Overall, cloud computing has given businesses from start-ups to corporations a better way to manage and deploy their products. Scalable computing power, lack of hardware with associated power and cooling costs, and reduced number of IT staff are just some of the benefits that have completely changed the business world.

3.2. Technology and services to be used

A large variety of software is to be used during development of the auditing tool; therefore their relationship has to be carefully studied and implemented. While the relationship with my client is informal, there are technologies I must use during development so that their engineering team can develop this application further. In this subsection I will cover AWS services, containerisation, container, and infrastructure management.

The high-level goal of my web application is to easily observe which AWS resources are older than "x" number of days, therefore it is required that I have fundamental knowledge of AWS and their most popular services. I undertook the online course "AWS Cloud Technical Essentials" offered by AWS (AWS, n.d.). The course covered best practices regarding account security, identity and access management, different types of computing, storage and database services and various ways of monitoring your AWS resources. I will expand further on each section in "Chapter 4" as to what service I chose for a particular application requirement.

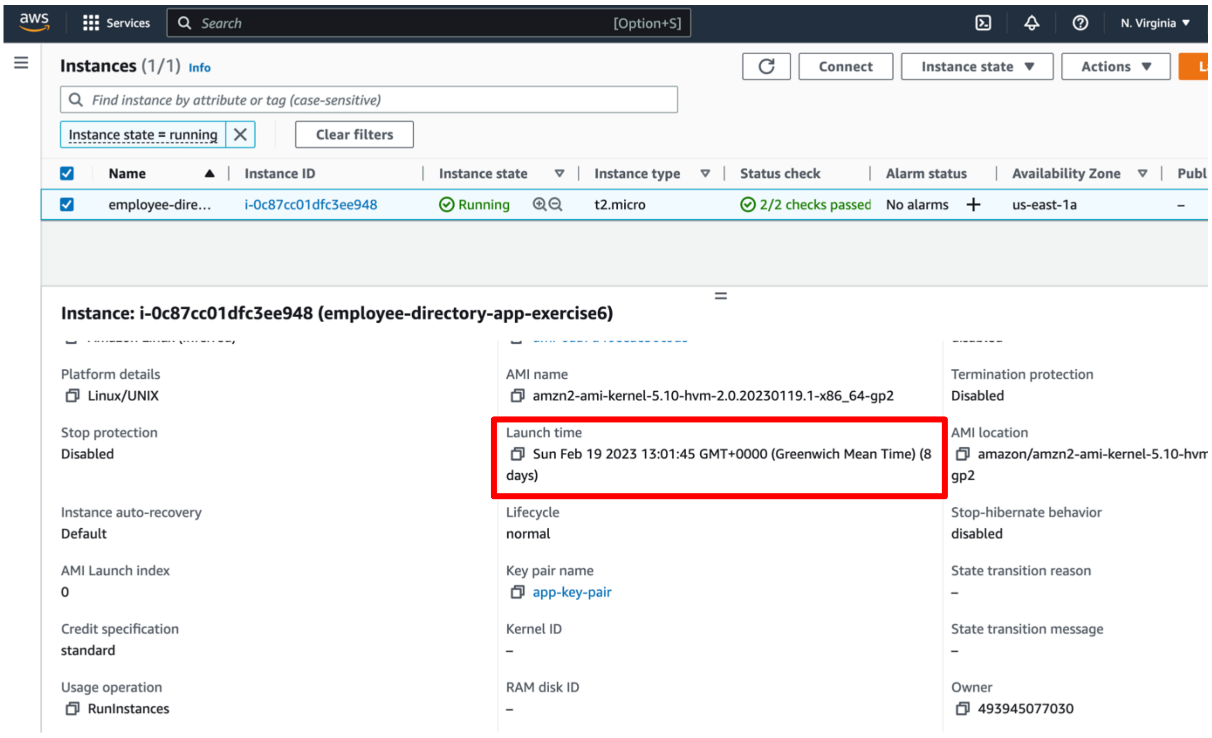
One of the required methodologies is deploying the web application to containers using Docker and managing those containers with Kubernetes. Containers are a packet of software which contain the source code and all their dependencies. It is a standalone unit which can be seamlessly run and deployed to different machines or environments (Docker, n.d.). Kubenetes is a platform for orchestrating containers, this includes automating services, scaling, and management of complex containerised workloads in the cloud (Kubernetes, n.d.). When developing the application, I will carefully study best practices about creating the relationship between the containers and Kubernetes in an AWS cloud environment. I plan on working through "Containerized applications on AWS" course offered by AWS themselves to study and learn good engineering practices (AWS, n.d.).

3.3. Similar applications on the market

When looking at similar products on the market there are two options, using the AWS console to perform checks manually, or implement a third-party application that manages all cloud resources under an account.

One can use the AWS management console web page to get all the information required about an EC2 instance and respective AMI, including the date it was launched. This manual task is not scalable as a large enterprise will likely have dozens of instances with various critical services running, as well as multiple IAM users under the root account. Many resources and users create room for human error, therefore not making this a viable solution for a large enterprise.

Third-party applications like CloudCheckr, CloudHealth, DataDog and many more, offer a cloud management service that covers cost optimisation, automation, performance, and security (CloudZero, 2023). These services cover many features, including my tool, getting the age and health of EC2 instances under an account. While a service like this may be useful for small to medium enterprises with smaller capital, my client is in need of a specific auditing tool, suggesting that a "do it all" product like this is not necessary.



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Publ
employee-dire...	i-0c87cc01dfc3ee948	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	–

Instance: i-0c87cc01dfc3ee948 (employee-directory-app-exercise6)

Platform details Linux/UNIX	AMI name amzn2-ami-kernel-5.10-hvm-2.0.20230119.1-x86_64-gp2	Termination protection Disabled
Stop protection Disabled	Launch time Sun Feb 19 2023 13:01:45 GMT+0000 (Greenwich Mean Time) (8 days)	AMI location amazon/amzn2-ami-kernel-5.10-hvm gp2
Instance auto-recovery Default	Lifecycle normal	Stop-hibernate behavior disabled
AMI Launch index 0	Key pair name app-key-pair	State transition reason –
Credit specification standard	Kernel ID –	State transition message –
Usage operation RunInstances	RAM disk ID –	Owner 493945077030

Figure 1 Screenshot of the AWS management console when looking at an EC2 instance's information in thorough detail. In red is boxed the date of the instance's launch. The instance "employee-directory" has been running for approximately 8 days.

Chapter 4: Method

4.1 Software development methodology

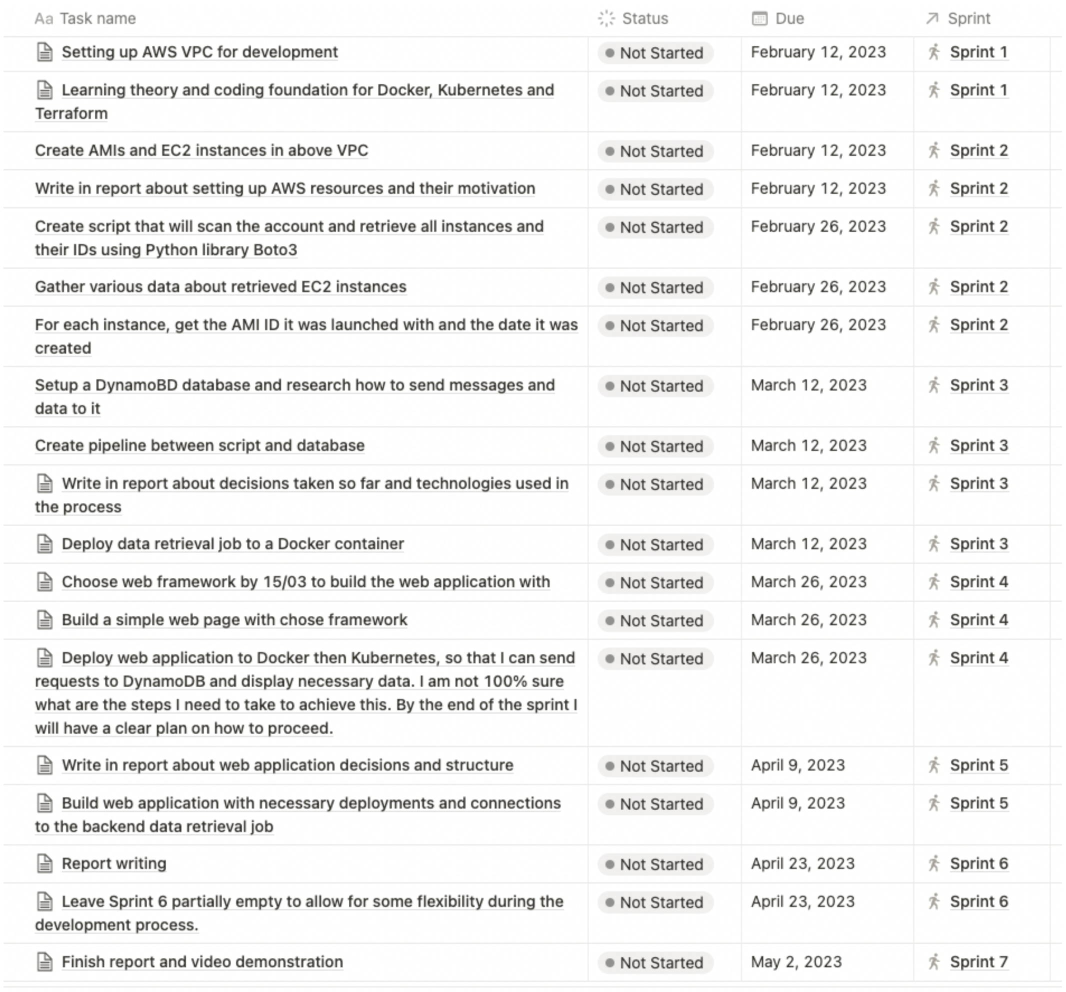
During the software development of the auditing tool the code was tested as it was being developed. With each feature added to the data processing job, I manually checked the data in storage with the information available in the AWS Management Console, while also checking the JSON formatting was correct. The API request handler and website were also constantly checked and verified while developing. The website was manually checked whether the response from the API was displayed as intended on the website, or if any piece of information is missing.

In terms of time management, I will scrum sprints of two weeks in length as discussed in the Project Development Plan. The project requirements are potentially split to smaller tasks and assigned a sprint, where at the end of each sprint there is a team meeting (sprint retrospect) to reflect on the progress of the tasks on that specific sprint. I am working alone, so at the end of each sprint I will assess my progress on the tasks and adjust them as needed. I believe this methodology is suitable for my project as it will allow for consistent output and reflection on my development progress, while it also being widely adopted in the industry and improving my professional skills. In the next section, I will go in greater detail about the work plan, how I stayed on track, and any adjustments I made during development.

4.2 Work plan

4.2.1 Initial plan

As discussed above and in the project proposal, I will use an agile scrum methodology to manage and deliver my project. In general, each sprint is typically 2 weeks long and is made up of four stages: sprint planning, daily scrum, sprint review and sprint retrospective. The sprint plan I created initially with the application Notion is shown in Figure 1, where the project requirements are split into smaller tasks and added to sprints. I used my best judgement to estimate how long a task will take, taking into consideration what skills I already know. There are exactly seven sprints, where the final sprint is just a weeklong, coinciding with the week between the code and report submission.



Task name	Status	Due	Sprint
Setting up AWS VPC for development	Not Started	February 12, 2023	Sprint 1
Learning theory and coding foundation for Docker, Kubernetes and Terraform	Not Started	February 12, 2023	Sprint 1
Create AMIs and EC2 instances in above VPC	Not Started	February 12, 2023	Sprint 2
Write in report about setting up AWS resources and their motivation	Not Started	February 12, 2023	Sprint 2
Create script that will scan the account and retrieve all instances and their IDs using Python library Boto3	Not Started	February 26, 2023	Sprint 2
Gather various data about retrieved EC2 instances	Not Started	February 26, 2023	Sprint 2
For each instance, get the AMI ID it was launched with and the date it was created	Not Started	February 26, 2023	Sprint 2
Setup a DynamoBD database and research how to send messages and data to it	Not Started	March 12, 2023	Sprint 3
Create pipeline between script and database	Not Started	March 12, 2023	Sprint 3
Write in report about decisions taken so far and technologies used in the process	Not Started	March 12, 2023	Sprint 3
Deploy data retrieval job to a Docker container	Not Started	March 12, 2023	Sprint 3
Choose web framework by 15/03 to build the web application with	Not Started	March 26, 2023	Sprint 4
Build a simple web page with chose framework	Not Started	March 26, 2023	Sprint 4
Deploy web application to Docker then Kubernetes, so that I can send requests to DynamoDB and display necessary data. I am not 100% sure what are the steps I need to take to achieve this. By the end of the sprint I will have a clear plan on how to proceed.	Not Started	March 26, 2023	Sprint 4
Write in report about web application decisions and structure	Not Started	April 9, 2023	Sprint 5
Build web application with necessary deployments and connections to the backend data retrieval job	Not Started	April 9, 2023	Sprint 5
Report writing	Not Started	April 23, 2023	Sprint 6
Leave Sprint 6 partially empty to allow for some flexibility during the development process.	Not Started	April 23, 2023	Sprint 6
Finish report and video demonstration	Not Started	May 2, 2023	Sprint 7

Figure 2 All tasks created for the project as of 5 Feb 2023, organised in sprints

In terms of the daily scrum, sprint review and sprint retrospect, those must be adjusted slightly to suit a one-person team like myself. The daily scrum is a daily meeting where a team briefly goes through their progress in the context of the whole sprint, and for myself, at the start of each working day I reflected on the plan and decided what is to be completed that day. Sprint review and retrospect is combined to one activity for me, and it was an opportunity for me to reflect on my pace of work or make any adjustments to the plan. In the next subsection I will go through each sprint of the project, with their reflections and potential changes to the project/plan.

Touching on the length of a sprint, I chose two weeks, but the official guide suggests any amount between one and four weeks (scrumguides.org, 2020). Based on my previous work experience and knowledge of my workflow, two weeks seemed to be the most optimal length. It allowed me to frequently reflect on the previous sprint, what I could improve and continue doing. The reflections of each sprint are discussed below.

4.2.2 Application of work plan and adjustments made to project/plan

In this subsection I will present the results of each sprint, how well I was able to stay on track, and any adjustments made to the plan.

Sprint 1 & 2: When creating the initial plan, I did not consider studying for literature review, so that was added in Sprint 1, and the development setup was pushed to Sprint 2. As a result, in Sprint 2 I successfully created the AWS Virtual Private Cloud network needed for the development of the auditing tool, as well as creating computing instances and images needed for testing in the network. I stayed on track with the workload and finished tasks within the expected timeframe.

Sprint 3: Created a Python script using the library Boto3 to scan the AWS account of instances and images, extracting all the data necessary. Created a DynamoDB database and pushed the data extracted to it in JSON format.

Sprint 4: Upon meeting with the client for an informal feedback meeting, I found out that the technical requirements I got were suggestions only, and that I need to find the most optimal way of building this tool for the team. The functional requirements are unchanged, only that now I can think outside the box. I started looking at server-less services AWS offers. At the end of sprint 4, I had a website running on the public domain using AWS's S3 service, with the necessary server-side scripts to retrieve and format the data accordingly, but the front and backend were not yet communicating. During Sprint 4, I tried a few different approaches to hosting the website, but I will discuss those in greater detail in the next chapter. That led to a rather loosely structured sprint, as a few of my tasks planned at the start of the project were not applicable anymore. The changes to the methodology were frequent, making a "To do" list more useful in this experimentation stage.

Sprint 5 & 6: By the end of Sprint 6, the front end of the web application and the back end were communicating via an API, and the server-side Python scripts developed in Sprint 3 were now executed serverless-ly. I had tried two other approaches to executing the server scrips for the web application, that being using the web framework Django and AWS's SDK for JavaScript. These attempts are elaborated further in this chapter and the next. In this sprint the product code was also finalised, by improving readability with comments and whitespace, and adding the necessary sources in the source code. For this time period of 4 weeks, a "To do" list was more helpful due to the always changing methods, and the sprint plan initially created has become less and less useful.

Sprint 7: Report writing as planned.

4.4 Design decisions and documentation

The web application for the tool is hosted on the cloud by AWS's S3 service, where AWS does all the work in making that website available in the public domain. In an S3 bucket, I uploaded the static website files (index.html and script.js) and the website was live. In terms of server-side scripting, I chose to use AWS's Lambda service for the data processing job, which executes any script you provide serverless-ly. This is a very powerful service as I do not have to maintain any hardware or update/ manage any libraries needed for server-side scripting. The front end will send a request through an API gateway to the Lambda service, update the data in storage for real time accuracy, retrieve it and display it on the page. In the section 4.5 below I will go in more detail about each component.

With the architecture detailed above, sub objective 3 from Chapter 1 is satisfied, as all cloud services listed do not require any software or hardware maintenance compared to a classic on-site server. I will elaborate further how each component of my design satisfies the objectives in the next section, but for now, I will briefly introduce other solutions I tried or considered prior to this final design.

Before the clarification I was given during the feedback meeting, I had started to develop the approach detailed in the PDD. This meant creating the static web files and data processing script as usual, only that they would run in a Docker container on an EC2 (Elastic Compute Cloud) instance and store the data in a DynamoDB database. As seen in the work plan, I only got as far as creating the data processing scripts and storing the data, but this design would not have satisfied sub objective number 3. While the solution would be scalable because those AWS services are designed to increase and decrease with demand, it would add a significant amount of complexity to the maintenance of the web application. EC2 is a computing solution, a computer on the cloud requires nearly as much software and libraries maintenance as a computer on-site with an application deployed on it. Pricing would not have been the cheapest possible, as EC2 services are paid per hour of running and you pay for all the hardware components of this cloud computer (CPU, GPU, storage, etc.). A web application like this would not take advantage of all the resources an instance would have to offer, in turn paying for something you are not using.

At this point in the project, I learnt the possibility of using S3 to host the website, only that I made the mistake of not understanding that S3 will only host static web files. This led to a few days wasted trying to upload a Python Django website on S3, but in the end, I did realise S3 will not process server-side scripts, like a .py file.

The final discarded solution was to use AWS official JavaScript SDK on the client side scripts. Once I was clear on what files S3 can and cannot process for a website, I came across the JS SDK, and I intended to get rid of the server-side scripts entirely and move that scripting to the client side using the SDK in a JS file, while it being hosted using S3. While this would not have significantly offended sub objective 3 in terms of maintainability or scalability, a much bigger problem arose once I started developing. For calls to AWS to be answered successfully, the source must have the appropriate permissions and credentials available, meaning that I would have to upload those credentials to the JS file. This JS file would be on the public domain, and so would the secret keys. Therefore, this solution was immediately withdrawn as it is extremely poor practice and would expose my AWS account to the world, and in turn my credit card.

With all that in mind, it is important to mention documentation of the currently implemented solution. The whole application is hosted on the cloud, even the server scripts, and the operation of the services the tool is using are largely documented by AWS themselves. I did create a diagram using Visual Paradigm of the tool's architecture, which will clarify how each of those AWS services communicates with each other.

There was no need for a detailed visual diagram for the Python scripts of the data processing job or the API request handler. There is no use of Object-Oriented programming, and the scripts have an intuitive structure. It is a main function with a list of operations, neither of them passes 100 lines, and there are helpful and descriptive comments throughout. As for data storage, the data generated by the data processing job (i.e. a Lambda function) is stored in JSON files with the typical JSON format and is stored in S3 buckets. Storing data to the buckets is a "put_object" line of code, it is not a structured database with attributes and entries. Therefore, an entity-relationship diagram for the data storage solution is not necessary.

4.5 Implementation of system components and version control

4.5.1 Website client-side files and hosting

To make the web application available on the public domain I used AWS's service S3. S3 can host a static website with static web files and any client-side scripts, but it is not capable of any server-side processing. The static web files which build my website are uploaded to an S3 bucket within the same account the data is extracted from. Both HTML and JavaScript files build the website (tables, search bar, filtering, colour coding, etc.), while also sending a request to the API to get the data necessary. This data coming in is then processed, ordered, and displayed by the JavaScript script, after which HTML and the Bootstrap library make it look appealing. The library jQuery is also used to implement the search bar feature and the ability to click on the date column header to sort by ascending /descending date. I used the library moment.js to calculate whether a given instance is older or younger than 30 days.

I chose to use S3 to host the website because of its low maintenance. All the hardware and software necessary to host a website locally or on another cloud computer is eliminated. Uploading the static files and client-side scripts in an S3 bucket is easy to maintain and significantly cheaper than those alternatives. In the US East region, the most expensive type of requests (PUT, COPY, POST, etc.) cost just $0.005/ 1,000 requests. This solution satisfies subobjective 2 and 3 brilliantly.

4.5.2 API gateway and request handler

The API was created using AWS's service Amazon API Gateway using the web Management Console. Any requests coming through the API are forwarded to the request handler which is a Lambda function within the same AWS account. The Lambda function which processes the request is a Python script utilising the library Boto3. This script will trigger another Lambda function (see 4.5.3) to update the data in storage, get the instances and images data from their respective S3 buckets (more on that in 4.5.4), and build a JSON response ready to be sent back through the API.

Lambda is a server-less service provided by AWS that will execute any script you provide and can be effortlessly connected to an API. One of the advantages of using Lambda for server-side execution of scripts is the extremely low maintenance and high resilience simply by eliminating the local machine. Hooking up a Lambda function with the API also allow for scalability as both services can adapt based on traffic on the website, and they are highly secure preventing unauthorised access. This server-less setup for the back end of my website satisfies subobjective 3 very well.

4.5.3 Data processing job

The data processing job is a Lambda function which consists of making calls to AWS to get the desired data on instances and images, formatting it to a JSON object, and pushing it to two separate buckets for each. This is completed using Python and the library Boto3 within the Lambda function, and its execution is triggered when a request comes through the API. Because the script is executed every time the website is loaded, the data stored is in real-time and most up-to-date, satisfying subobjective 1. Another advantage of using Lambda for server scripting is having the flexibility to develop without any physical infrastructure constraints, as well as only paying for the milliseconds it is running.

4.5.4 Data storage

Once the data processing Lambda function discussed above processes the required data in a JSON object, it is pushed to two separate S3 buckets. S3 (Simple Storage Service) is a cloud service for storing and retrieving data via the web, making the data easy and fast to access. S3 buckets also provide virtually unlimited capacity and have built-in security features to protect access to the data. The data about running instances and images are stored in two separate buckets, each in a JSON data format to make it easy to read and manipulate in the website static files.

4.5.5 Security: Identity and access management

Because the application operates entirely using AWS services, all those components need to have permission to interact with each other. IAM (AWS Identity and Access Management) is the service that provides granular control over what a resource can do, read, or not do. For example, the Lambda function which gets and stores data in S3 buckets follows the principle of least privilege (Amazon, 2022): it can only read and write only the buckets it needs. This concept of "least privilege" using IAM permissions was applied to all services used in this project to ensure best practices, tight security, and minimal reach in case of unauthorised access. In Chapter 5 I will elaborate in greater detail on how each component is handled.

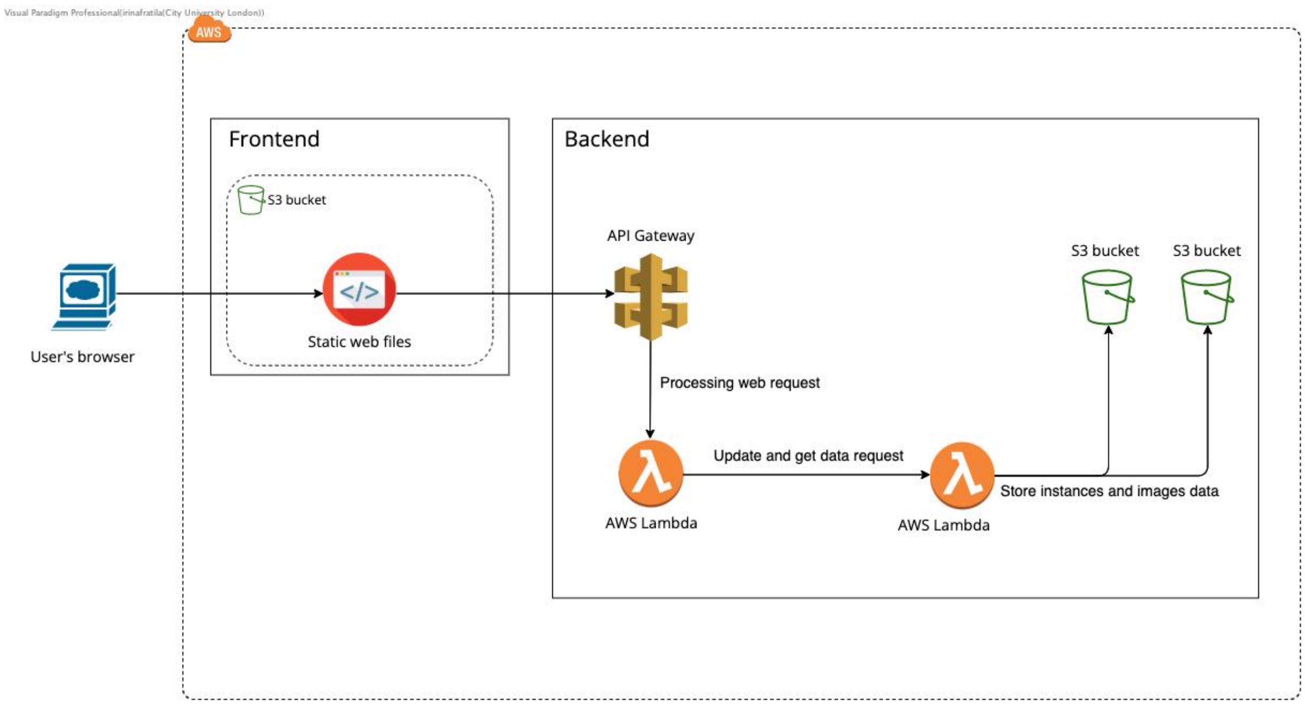
4.6 Testing

Testing whether my website is being hosted and displayed accordingly was straightforward due to AWS doing the heavy lifting to make it public. Every time a new feature was added to the static web files or server scripts, I constantly checked the output on the website for any missing data or formatting mishaps. I used Postman to test the API's response to the request, always checking for the appropriate JSON format of the response. That in turn also tests all the other Lambda functions' functionality from the backend, as an incorrect response meant one of them was misfunctioning. I also tested the data processing job by accessing the S3 buckets manually from the AWS Web Management Console against the main instances page (Figure1). In Chapter 5, this approach of testing, its advantages and disadvantages, are discussed in more detail.

Chapter 5: Results

In this chapter I discuss all the outputs produced in Chapter 2, and the work done to complete them. I will also elaborate on how each component satisfies the objectives from Chapter 1, the decisions taken, and what other alternatives/ solutions have been discarded.

5.1 Architecture diagram



After many iterations and decisions, the final infrastructure of the application is entirely on the cloud, and the diagram above describes how each component works together. In this subsection, I will briefly discuss how the application works as a whole, in the following sections each component will be isolated and elaborated on how it works and why it satisfies the project's objectives.

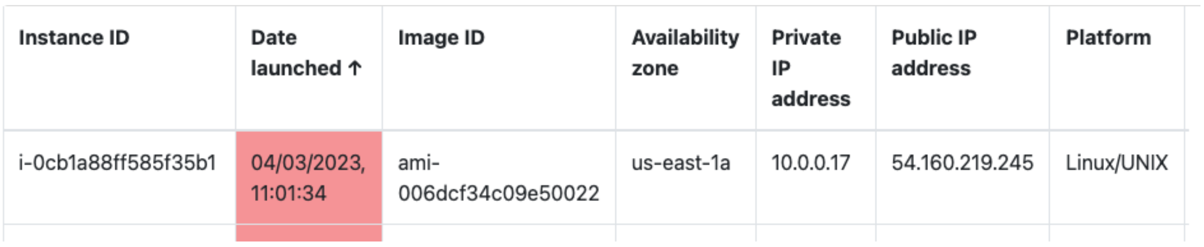
Referring to the diagram above, the first point of contact once accessing the website is an S3 bucket. This S3 holds the website's static files, and its contents are public, therefore creating a fully functional website. Once the website is accessed, a GET request is sent to the API gateway, and in turn makes its way to the request handler Lambda function. To satisfy subobjective number 1, the second Lambda function is then invoked which updates the data in storage to the latest figures. Once that is complete, the data stored in each S3 bucket is retrieved and brought forward to be displayed on the website. In the end, the web application displays all the latest data clearly, with an infrastructure that is easy to maintain and scale up, in turn satisfying all the project objectives.

5.2 Website static files

There are two files that build the front end of the website: index.html and script.js. Both files are uploaded to an S3 bucket to make it a functioning website (more on that in section 5.3). They can be found in the "web-app" folder on the Moodle submission and is the output outlined in section 2.1.

Starting with the HTML file, the very first thing built is the request to the API Gateway. The request is built with the method GET, no body necessary, and posted to the API Gateway link. The response is then forwarded to a function `response_handler()` within the HTML file for data processing. The response received from the API is in JSON format, it is parsed and separated into instances and AMI data. Once separated and turned into JS dictionaries, the data is sorted by date in descending order. The sorted data is ready to be displayed on the website and each put in their respective tables. The functions `buildInstancesTable(instanceData)` and `buildCustomImagesTable(AMIData)` are written in the file script.js, and they are built on the skeleton provided by Dennis Ivy on YouTube (Ivy, 2019).

In the video, he showcases the basic technique for displaying data in a dictionary in an HTML table by creating rows and changing the `innerHTML` of the table. I have added an extra layer on top of his technique, and that is colour coding by the age of the resource. This is a project requirement I successfully satisfied (subobjective 2), and it was achieved with the use of the moment.js library. Right before each data point in the dictionary is to be transformed to an HTML row (i.e. `<tr>`), the function `isOlderThanXDays(ageLimit, date)` is called with each data point to compute whether a resource is older than 30 days. If so, the date attribute for that resource will be filled in with red, green otherwise. The same computation and logic applies to both the "Instances" and "Images" tables.



Instance ID	Date launched ↑	Image ID	Availability zone	Private IP address	Public IP address	Platform
i-0cb1a88ff585f35b1	🟥 04/03/2023, 11:01:34	ami-006dcf34c09e50022	us-east-1a	10.0.0.17	54.160.219.245	Linux/UNIX

Figure 3 Example of an instance in the "Instances" table where it has been launched more than 30 days ago.

The function `isOlderThanXDays(ageLimit, date)` takes a date and the number of days to set as the age limit. I used the moment.js library to convert the date to a moment.js date object and subtracted the `ageLimit` value from that. I chose to use that library so that I can accurately compute x days before the date given.

Once the tables are populated with data, each is given a search text box and the ability to click the "Date launched" column for sorting by the earliest or latest date. The functionality of those features was achieved using the jQuery library and Ivy's videos "Sortable columns with Javascript" and "Search/filter table data with Javascript" (Ivy, 2019)((Ivy, 2019). I used his methodology as the skeleton for my implementation of the search bar and the sortable "Date" column". With the library jQuery, on every character in the search bar typed, it is taken and used to search if any instance ID or image ID contains the value from the search bar. The function `serchInstancesTable(searchValue, data)` or `searchImagesTable(searchValue, data)` is called depending on which table you are searching on, and it returns a dictionary with any data points that match in the search bar input.

In terms of sorting by ascending or descending date jQuery was used to detect a click on the "Date launched" header, and as mentioned above, the data is initially sorted in descending order, with every click the order switches between ascending and descending and calling the custom function `sortByDate(a, b, descOrAsc)`.

```
instancesDict = instancesDict.sort((a,b) =>
sortByDate(a['date_launched'],b['date_launched'], 'asc'))
```

The line above can be found in index.html on line 107 and it's the point where each data point from the dataset is taken and compared to one another. My custom function `sortByDate` takes two dates and how to sort them and returns 1 or -1 based on what date is supposed to be before the other. The function is found in the script.js file.

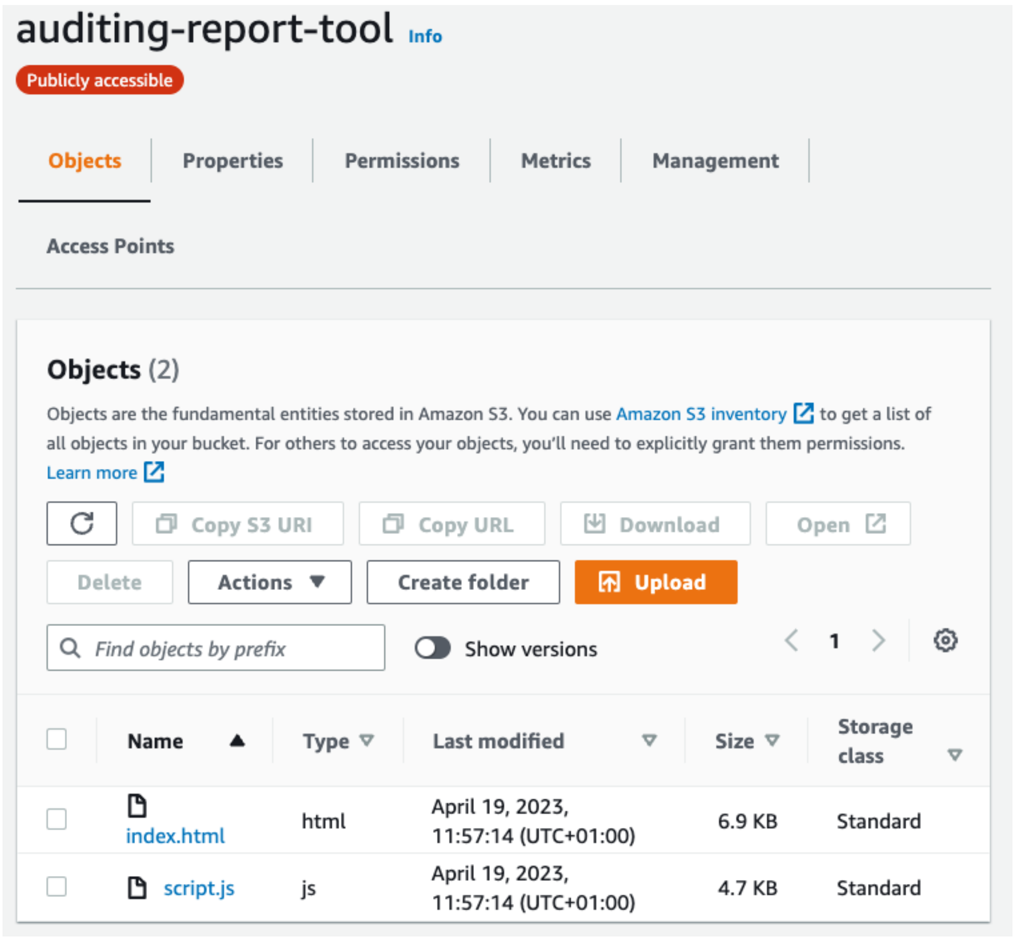
```
function sortByDate(a,b, descOrAsc) {
    if (descOrAsc == 'desc') {
        if (a > b) {
            return 1
        } else {
            return -1
        }
    } else if (descOrAsc == 'asc') {
        if (a < b) {
            return 1
        } else {
            return -1
        }
    } else {
        throw "Invalid option for descending or ascending"
    }
}
```

This line of logic is applied to both tables on the page, each one having its own "search" function, successfully satisfying subobjective 2. The two static web files successfully clearly display the data, provide means for filtering data and colour coding for resources based on time passed since launch/ creation.

5.3 Publishing the website to the public domain

As mentioned before, the website is hosted using AWS's S3 service. S3 in itself is a storage service, with virtually no storage limit, and can be accessed via the web from anywhere. It is not a structured database, more like a classic folder on your computer where data and files can be dumped. With S3, that folder is called a bucket.

The bucket which hosts the website is named "auditing-report-tool" and as you can see in Figure 4, the static files discussed in section 5.1 are uploaded to the bucket.



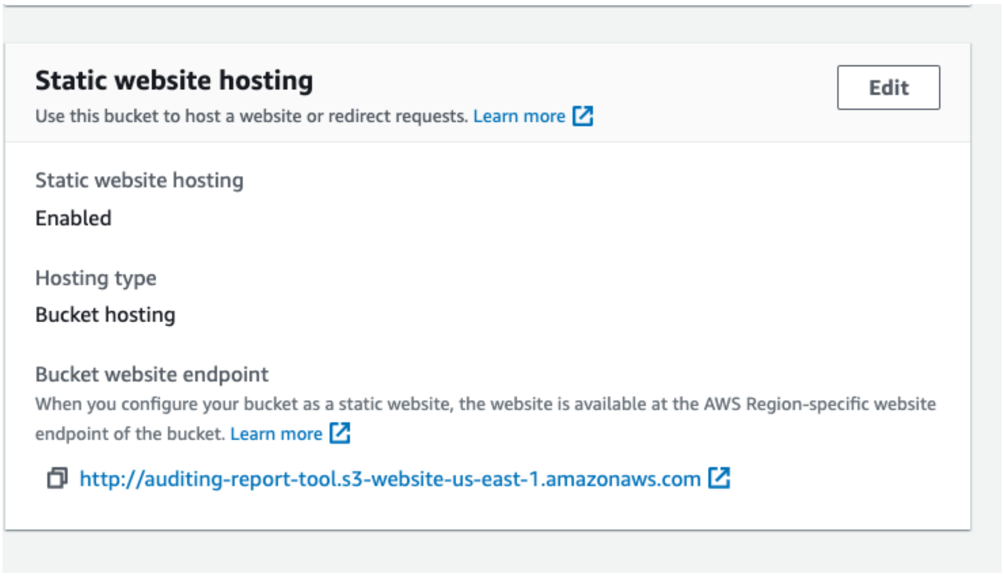
Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
index.html	html	April 19, 2023, 11:57:14 (UTC+01:00)	6.9 KB	Standard
script.js	js	April 19, 2023, 11:57:14 (UTC+01:00)	4.7 KB	Standard

Figure 4 The bucket which hosts the application website

To turn the uploaded files into a functional website in the public domain, S3 makes it frictionless by turning a toggle on. Changing "Static website hosting" to Enabled will get the job done (Figure 5).



Static website hosting

Use this bucket to host a website or redirect requests. Learn more

Static website hosting
Enabled

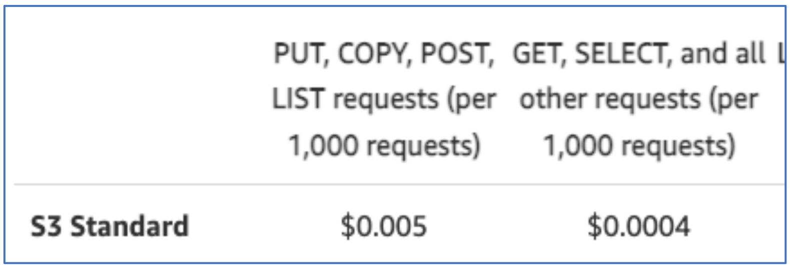
Hosting type
Bucket hosting

Bucket website endpoint
When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket. Learn more

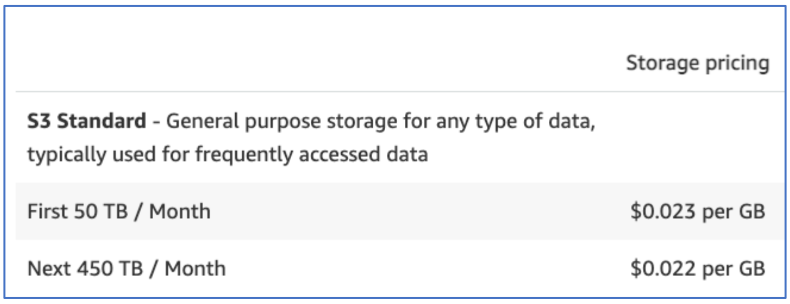
http://auditing-report-tool.s3-website-us-east-1.amazonaws.com

Figure 5 Turning "Static website hosting" toggle to Enabled

This solution is extremely cost-effective, with S3 you're only charged for the storage and data transfer used, there is no cost of maintaining physical hardware like an on-site server. To keep the website in the public domain with minimal usage, accessed about 15-20 times a day, we are looking at less than $1 per month. In terms of scalability, S3 is highly adaptive and can handle large amounts of traffic without performance compromise. An S3 bucket requires minimal maintenance as objects can be easily deleted or edited using the



	PUT, COPY, POST, LIST requests (per 1,000 requests)	GET, SELECT, and all other requests (per 1,000 requests)
S3 Standard	$0.005	$0.0004



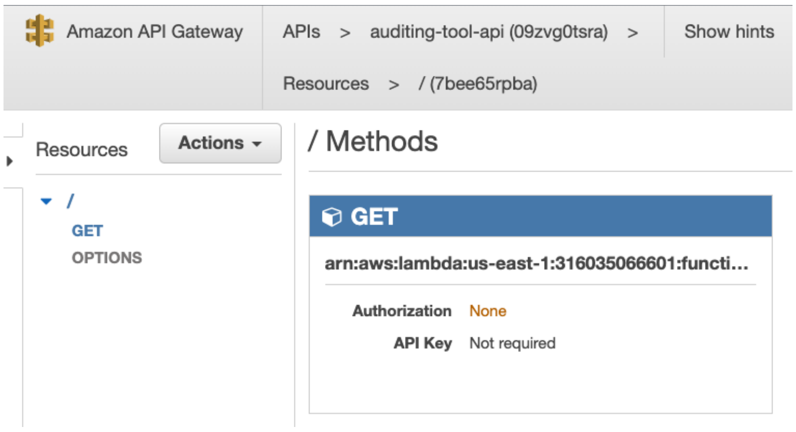
	Storage pricing
S3 Standard - General purpose storage for any type of data, typically used for frequently accessed data	
First 50 TB / Month	$0.023 per GB
Next 450 TB / Month	$0.022 per GB

web interface of AWS CLI. And of course, the lack of physical hardware and software to maintain is one of the biggest advantages.

If I had followed the Docker infrastructure outlined in section 4.4, it would have introduced a lot more variables and maintenance to the system. By using S3 to host the website in the public domain, subobjective 3 is satisfied on all fronts, making this solution optimal. This is also part of the output in section 2.5.

5.4 API gateway

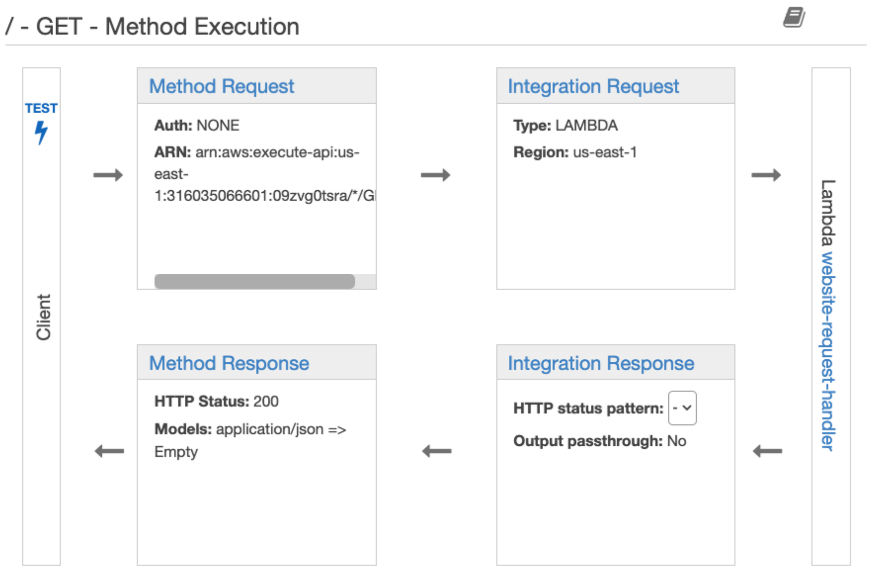
To create a communication medium between the website and the Lambda server-side script I chose to use AWS's API Gateway service. It is a RESTful API, named "auditing-tool-api", that was created within AWS's web interface, Management Console. Within the web interface, I hooked up the GET requests coming in with the Lambda function "website-request-handler".



arn:aws:lambda:us-east-1:316035066601:functi...

Authorization None
API Key Not required

Figure 6 Screengrab of the API "auditing-tool-api" used in the project to connect the front and back end.



Auth: NONE
ARN: arn:aws:execute-api:us-east-1:316035066601:09zvg0tsra/*/G

Type: LAMBDA
Region: us-east-1

HTTP Status: 200
Models: application/json => Empty

HTTP status pattern: -
Output passthrough: No

Figure 7 Connection between GET request incoming and the Lambda function to handle the request.

AWS's API can scale up or down depending on demand, is easy to use/ create within a few clicks on the Management Console, and seamlessly integrates with other AWS services. The effortless setup and interaction between the API and a Lambda function is its biggest strength as it significantly reduces maintenance and set-up time. It is by far the strongest candidate to satisfy subobjective 3 in terms of manageability, scale, and price, coming in at $3.50 for the first 333 million requests/month (AWS, n.d.). The API is part of the output in section 2.3.

5.5 Request handler

The script which processes requests coming in through the API is a Lambda function named "website-request-handler". It relies on the SDK Boto3 officially offered by AWS to get the data in storage and return it back through the API. This script is part of the output in section 2.3, and can also be found in the folder “python-scripts-from-awsLambda” in the Moodle submission under the filename “website-request-handler.py”.

The process is split inti two parts: getting the data from storage and formatting it to be sent back through the API. The function `lambda_handler(event, context)` is the equivalent of a regular Python script’s “if __name__ == __main__” in Lambda. Once in here, the first step is to update the data stored in S3 buckets to real-time by calling another Lambda function “auditingtool-updateInstancesAndImagesBuckets” asynchronously. After this lambda function does all the necessary steps to update the data in storage the response code is checked to be “202”, the data is extracted from their respective buckets. A call to AWS using `get_object` from both “running-instances-data-if1” and “custom-amis-data-if1” will return the contents, which is a JSON file. Both JSON responses are put together to form the body of what will be sent through the API. The the code snippet below builds the response and returns it, ready to be displayed on the website.

```
response = {
            'instances': instances_json_to_send,
            'amis': ami_json_to_send
            }

return {
        'statusCode': 200,
        'body': json.dumps(response)
        }
```

Lambda is a highly scalable service, so it can scale up with increasing calls to the functions, but it is also easy to manage over time. With no hardware/ software to update, or development environments, using Lambda to execute server-side scripting is the most effective solution for my website. With Lambda you are only charged for the requests coming in and the time taken to execute, where the first 1M requests cost $0.20 and 0.000016 for every GB/sec of runtime. Making the cost of running this server script well below $1, therefore satisfying subobjective 3 brilliantly.

Regarding the bug motioned on the very first line of the report, that is, the data not populating the tables when opening the web page for the first time ever. I believe the issue lies with how the Lambda function which updates the data in storage is called. Line 16 in website-request-handler.py:

```
update_buckets_response = lambda_client.invoke
      (FunctionName='auditingtool-updateInstancesAndImagesBuckets',
      InvocationType='Event')
```

The function is called with “Event” type, meaning that is called asynchronously so the script does not wait for the function to return a response, it will carry on execution. In turn not having a response and the website trying to display an empty response. This could be fixed by changing the invocation type to “RequestResponse”, where the script explicitly waits for the Lambda function invoked to return a response. An error like this could’ve been avoided but thankfully is an easy fix.

5.6 Data processing job and data generated

The data processing job is a Python script deployed to a Lambda function, and it gets all the data necessary by sending requests to AWS and stores it in S3 buckets with a JSON format. It is named “auditingtool-updateInstncesAndImages” and it is the output named in 2.1. The data stored in S3 buckets will be broken down in 5.6.2 and it refers to the output in 2.4.

5.6.1 Python script in AWS Lambda

The script is built using Python and the library Boto3 to make the necessary calls to AWS. The two resources I am collecting data on are running EC2 (Elastic Compute Cloud) instances and user-made machine images (referred to them as AMIs).

The first line of execution in the main function (that is `lambda_handler()`), a call to the EC2 service is made to return all instances, with a filter for the ones that are currently running. The filter was created by John Rotenstein (Rotenstein, 2019) and used in my script. Once presented with a list of instances that are running, with all their metadata, I now extract the data needed and most important.

```
    instance_data = []
    for r in all_instances['Reservations']:
        for instance in r['Instances']:
            instance_id = instance['InstanceId']
            ami_id = instance['ImageId']
            date_launched = instance['Launch Time'].strftime("%d/%m/%Y,
%H:%M:%S")
            availability_zone =
instance['Placement']['AvailabilityZone']
            private_ip_address = instance['PrivateIpAddress']
            public_ip_address = instance['PublicIpAddress']
            platform = instance['PlatformDetails']
            list_of_tags = instance['Tags']
            instance_data.append({
                'instance_id': instance_id,
                'ami_id': ami_id,
                'date_launched': date_launched,
                'availability_zone': availability_zone,
                'private_ip_address': private_ip_address,
                'public_ip_address': public_ip_address,
                'platform': platform,
                'tags': list_of_tags
            })
```

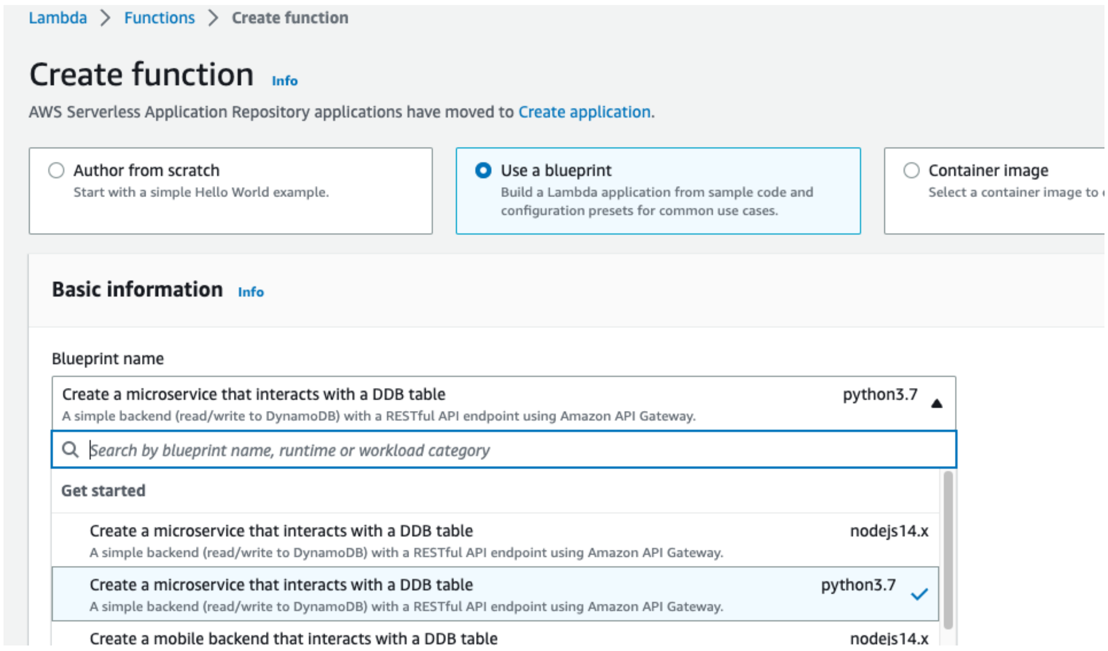
Looking at the code snippet above, the array `instance_data` will hold dictionaries of the data extracted about currently running EC2 instances. The data of importance is: instance ID, AMI ID, date it was launched, availability zone, private and public IP address, operating system, and tags attached. A similar array is created for custom AMI data, with its ID, name, and creation date. Each instances and images array are stored in their own buckets: “running-instances-data-if1” and “custom-amis-data”. But before the newly created arrays are pushed to the buckets as JSON objects, the existing data is deleted from the buckets and rewritten. This is to satisfy subjective 2’s point about real-time data, every time the page is loaded, this data extraction process occurs and the data is most up to date.

To push the data into their respective buckets, two calls to S3’s service `put_object()` are made. These are what the calls look like, where the “Body=json.dumps(array) was used from Santos on StackOverflow (Santos, 2021).

```
instances_response = s3.put_object(Body=json.dumps(instance_data),
Bucket='running-instances-data-if1', Key='running-ec2-instances.json')

amis_response = s3.put_object(Body=json.dumps(images_data),
Bucket='custom-amis-data-if1', Key='custom-amis-data.json')
```

To properly format the response of this Lambda function, the function `respond(err, res=None)` within the file is called. This function was automatically generated by AWS when I created this Lambda function because I used a template to do so. The template I used can be seen in Figure 8, “Create a microservice that interacts with a DDB table”, and the `respond()` function is the only thing kept from the template generated. Only one modification was made to the function, and that is changing the success status code from “200” to “202”.


Lambda > Functions > Create function

Create function Info

AWS Serverless Application Repository applications have moved to Create application.

Author from scratch
Start with a simple Hello World example.
Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.
Container image
Select a container image to

Basic information Info

Blueprint name

Create a microservice that interacts with a DDB table — python3.7
A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway.

Search by blueprint name, runtime or workload category

Get started

Create a microservice that interacts with a DDB table — nodejs14.x
A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway.

Create a microservice that interacts with a DDB table — python3.7 ✓
A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway.

Create a mobile backend that interacts with a DDB table — nodejs14.x

Figure 8 Template used to create the data processing Lambda function

As discussed in 5.5, the benefits of using Lambda for server-side scripting greatly satisfy subobjective 3, while this particular script ticks off subjective 1 by refreshing the data in storage with every call.

5.6.2 Data generated

This is what the currently running instances data looks like in the “running-instances-data-if1” bucket with two running instances, one with the “NeverDelete” tag attached. The file where this data is stored is named “running-ec2-instances.json”.

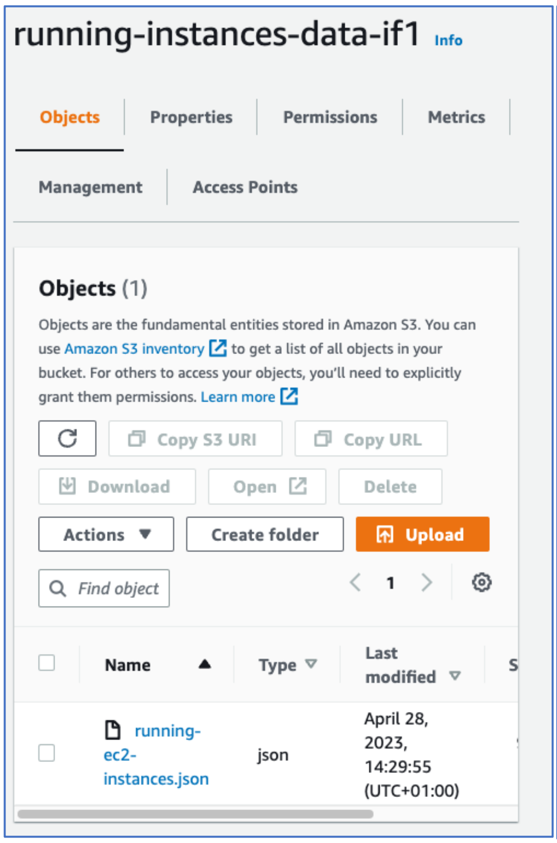
```
[
   {
      "instance_id":"i-0cb1a88ff585f35b1",
      "ami_id":"ami-006dcf34c09e50022",
      "date_launched":"04/03/2023, 11:01:34",
      "availability_zone":"us-east-1a",
      "private_ip_address":"10.0.0.17",
      "public_ip_address":"54.160.219.245",
      "platform":"Linux/UNIX",
      "tags":[
         {
            "Key":"Name",
            "Value":"instance1"
         }
      ]
   },
   {
      "instance_id":"i-04a257efb240061ea",
      "ami_id":"ami-0cb3484c1a6f84067",
      "date_launched":"04/03/2023, 12:43:41",
      "availability_zone":"us-east-1a",
      "private_ip_address":"10.0.6.72",
      "public_ip_address":"54.160.138.127",
      "platform":"Linux/UNIX",
      "tags":[
         {
            "Key":"Name",
            "Value":"instance2"
         },
         {
            "Key":"NeverDelete",
            "Value":""
         }
      ]
   }
]
```

And this is how data is stored about custom AMIs in the bucket “custom-amis-data-if1” in the file named “custom-amis-data.json”.

```
[
   {
      "image_id":"ami-0cb3484c1a6f84067",
      "image_name":"CustomAMIRecipe 2023-03-04T11-20-20.189Z",
      "creation_date":"04/03/2023, 11:35:22"
   },
   {
      "image_id":"ami-026356055b38dcfc7",
      "image_name":"CustomAMIRecipe2",
      "creation_date":"27/03/2023, 19:18:11"
   },
   {
      "image_id":"ami-0e6b71a8147764d2c",
      "image_name":"CustomAMIRecipe3",
      "creation_date":"27/03/2023, 19:44:26"
   }
]
```

5.7 Data storage

There are two S3 buckets as mentioned above, one to store the running instances data, and the other for custom AMIs. Below is a screengrab of them in the S3 management console. This is part of the output detailed in 2.5.

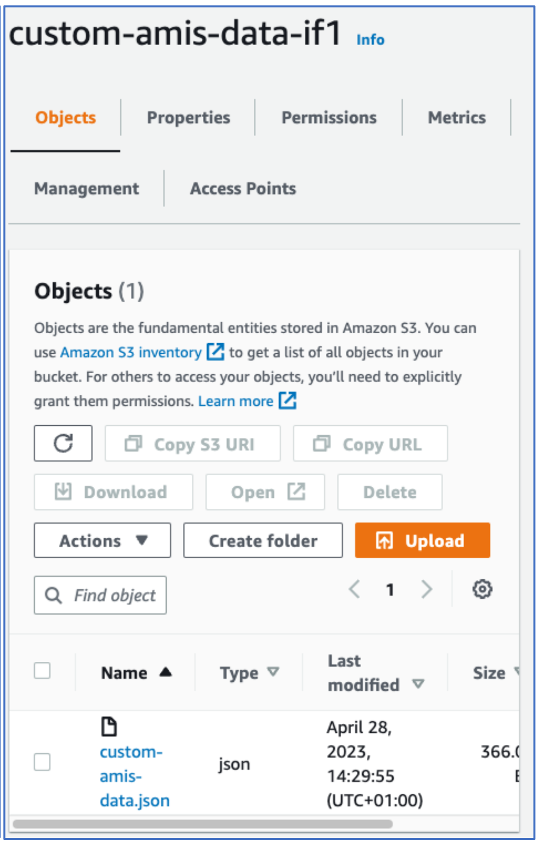


running-instances-data-if1

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

	Name	Type	Last modified	S
	running-ec2-instances.json	json	April 28, 2023, 14:29:55 (UTC+01:00)	



custom-amis-data-if1

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

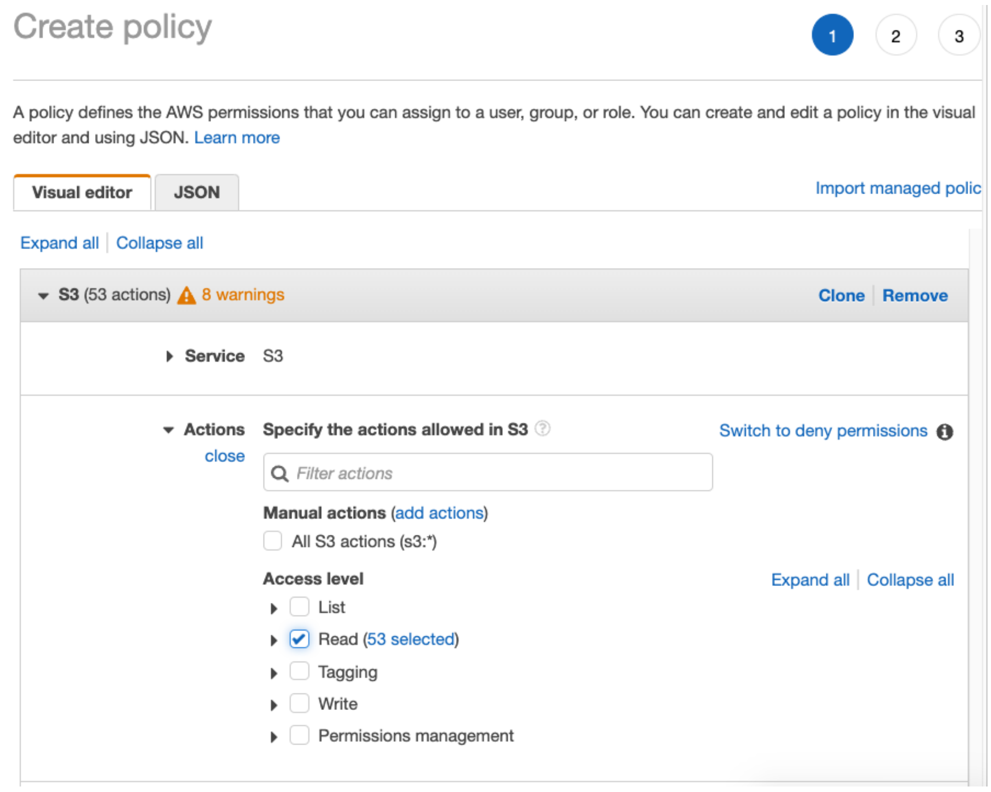
	Name	Type	Last modified	Size
	custom-amis-data.json	json	April 28, 2023, 14:29:55 (UTC+01:00)	366.(E

The advantage of using S3 buckets for simple JSON data is their accessibility and durability. With buckets being accessible via the web, a lot of friction is removed when trying to extract or store data, compared to a traditional local database. S3 buckets are also designed for durability, with built-in data redundancy and back-up options, offered and handled by AWS (AWS, 2023). This solution for storing the data needed for the website provides minimal maintenance and great scalability, making it a great candidate for subobjective 3.

5.8 Security: Identity and access management

As AWS themselves mention in their documentation (AWS, 2023), it is best practice to create an admin use for your AWS instead of using the root user. When creating an AWS account, the root user is the identity created which also has access to all AWS services and resources. It is important to add Multi Factor Authentication to the root account to restrict unauthorised access, it is a high-value target because it has access to all resources. By creating an admin user with more restricted access for daily activities it minimises the reach an unauthorised person might have. The account the project is operating within is set up in this manner, where the Admin user not being able to see billing information, change security measures like MFA or keys, terminate the account, and many more high risk operations.

In terms of managing permissions between AWS services themselves, I used the Identity and Access Management (IAM) service to follow the principle of least permission. For the Lambda function which responds to API requests "website-request-handler", it is assigned a role named "website-request-handler-role-gw8lr1y6" which gives it permission to read and write only what is strictly necessary. It has two sets of permissions: to read the two buckets which contain the data to be displayed on the website, and to invoke the Lambda function which updates data in storage. The sets of permissions were created via a visual interface in the Management console like in Figure 9.


Create policy

1 2 3

A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. Learn more

Visual editor | JSON | Import managed polic

Expand all | Collapse all

S3 (53 actions) ⚠ 8 warnings | Clone | Remove

Service S3

Actions close — Specify the actions allowed in S3 | Switch to deny permissions

Filter actions

Manual actions (add actions)
☐ All S3 actions (s3:*)

Access level | Expand all | Collapse all
☐ List
✓ Read (53 selected)
☐ Tagging
☐ Write
☐ Permissions management

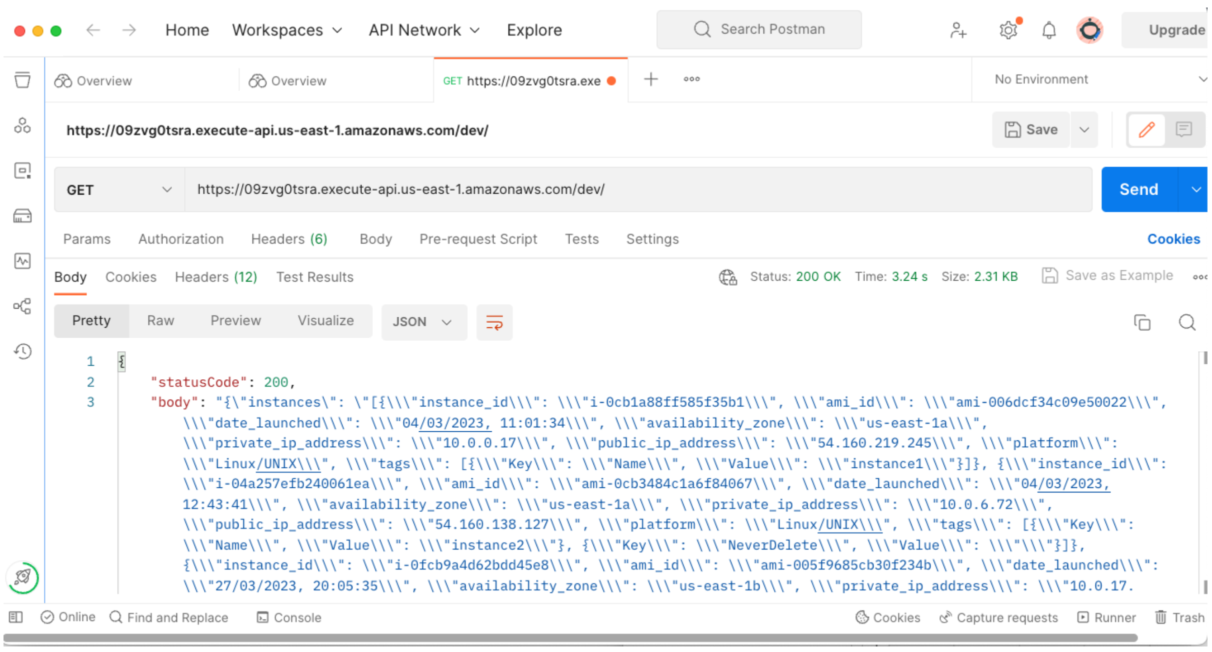
Figure 9 Example of the UI that creates a new policy.

Once a policy is created manually, IAM will turn the permissions to JSON, and below you will find the two sets of permissions mentioned above for the function "website-request-handler", the first one restricting only a red operation on the two buckets.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": [
                "arn:aws:s3:::running-instances-data-if1/*",
                "arn:aws:s3:::custom-amis-data-if1/*"
            ]
        }
    ]
}
```
And this policy here will allow the Lambda function to call the other Lambda function "auditingtool-updateInstancesAndImagesBuckets".
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-east-
1:316035066601:function:auditingtool-updateInstancesAndImagesBuckets"
        }
    ]
}
```
For the second Lambda function which updates the data in storage, the permission policy looks similar, only allowing a limited number of operations of the S3 buckets.
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeImages",
                "s3:PutObject",
                "ec2:DescribeInstances",
                "s3:ListBucket",
                "s3:DeleteObject",
                "ec2:StopInstances"
            ],
            "Resource": "*"
        }
    ]
}
```

5.9 Testing

This project was primarily tested manually due to its relatively small size and minimal moving parts involved. To test the API Gateway and the request handler Lambda function I used Postman, an application that sends custom requests to any link provided. I sent a GET request and check for any error returned or mistake in the data formatting.



GET https://09zvg0tsra.execute-api.us-east-1.amazonaws.com/dev/

Status: 200 OK Time: 3.24 s Size: 2.31 KB

```
{
    "statusCode": 200,
    "body": "{\"instances\": \"[{\\\"instance_id\\\": \\\"i-0cb1a88ff585f35b1\\\", \\\"ami_id\\\": \\\"ami-006dcf34c09e50022\\\",
        \\\"date_launched\\\": \\\"04/03/2023, 11:01:34\\\", \\\"availability_zone\\\": \\\"us-east-1a\\\",
        \\\"private_ip_address\\\": \\\"10.0.0.17\\\", \\\"public_ip_address\\\": \\\"54.160.219.245\\\", \\\"platform\\\":
        \\\"Linux/UNIX\\\", \\\"tags\\\": [{\\\"Key\\\": \\\"Name\\\", \\\"Value\\\": \\\"instance1\\\"}]}, {\\\"instance_id\\\":
        \\\"i-04a257efb240061ea\\\", \\\"ami_id\\\": \\\"ami-0cb3484c1a6f84067\\\", \\\"date_launched\\\": \\\"04/03/2023,
        12:43:41\\\", \\\"availability_zone\\\": \\\"us-east-1a\\\", \\\"private_ip_address\\\": \\\"10.0.6.72\\\",
        \\\"public_ip_address\\\": \\\"54.160.138.127\\\", \\\"platform\\\": \\\"Linux/UNIX\\\", \\\"tags\\\": [{\\\"Key\\\":
        \\\"Name\\\", \\\"Value\\\": \\\"instance2\\\"}, {\\\"Key\\\": \\\"NeverDelete\\\", \\\"Value\\\": \\\"\\\"}]},
        {\\\"instance_id\\\": \\\"i-0fcb9a4d62bdd45e8\\\", \\\"ami_id\\\": \\\"ami-005f9685cb30f234b\\\", \\\"date_launched\\\":
        \\\"27/03/2023, 20:05:35\\\", \\\"availability_zone\\\": \\\"us-east-1b\\\", \\\"private_ip_address\\\": \\\"10.0.17.
```

Figure 10 Using Postman to send a GET request to the API

Any issue with the response, in turn, means that there is unintended behaviour from the request handler. These two elements were constantly run and tested as I developed, fixing any errors and bugs as I went. I manually tested the data processing job and storage by running the script and checking the bucket's content, but also checked the data displayed on the website for any anomalies. While I may not have followed a structured approach for testing, the approach proved sufficient as all project objectives have been met successfully and within the time provided.

Overall, with the infrastructure constructed for this project and solution built, all the project objectives set in Chapter 1 have been met successfully. The web application does display all information necessary about currently running instances and AMIs, it is capable of searching by ID and sorting my date, as well as colour coding by the age of a resource.

Chapter 6: Conclusions and discussions

6.1 Objective review

In this section I will review the objectives set in Chapter 1 and to what extent they have been fulfilled.

6.1.1 Main objective

The main objective in 1.2.1 describes a website that will display various useful data about running AWS instances and custom images. The user must be able to filter by ID or name and observe the age of a resource with colour coding. The main object has been successfully fulfilled as the web application and the back-end processing produce the desired output.

6.1.2 Sub objectives

This section will go through each sub objective outlined in 1.2.2.

Sub objective	Reflection
1.A data processing job must collect the latest data about running EC2 instances and user-made AMIs (Amazon Machine Images) and store it.	This objective has been successfully met as the data collected by the data processing job is accurate and in real time. The data processing job is a Lambda function that is called every time the web application is loaded. It was achieved with the use of Python and the library Boto3 and has been elaborated in sections 4.5.1 and 5.2 in great detail.
2.The information extracted by the job above must be displayed on the website clearly with indicative colour coding by the date launched, and filtering options for all the data tables.	This objective has been successfully met apart from the bug I noted after submission. This bug will display the webpage and tables but with no data in the tables, but it has been discussed in section 5.3. Apart from that, the project outcome ticks all the requirements: colour coding, search functionality, and sorting descending/ ascending by date. The website static files which satisfy this sub objective are discussed in great detail in sections 4.5.1 and 5.2.
3.The application must be easy to maintain, scalable, and as cheap as possible to run.	This has been successfully met by all the elements of the system. The entire architecture is hosted on the cloud by AWS, in turn, satisfying all requirements of minimal maintenance, scalability, and price efficiency. In each

	system component In Chapter 5, it is related to this sub objective and how it satisfies it.

6.2 Reflections and what I learnt

The biggest learning curve was AWS and its services. By interacting with so many of their services: S3, Lambda, DynamoDB for the first draft, and API Gateway I was forced to study the documentation as well as their pricing. To build a convenient system that is also price efficient I had to look at multiple AWS services and weigh out how much maintenance and scalability each offers. With how frequently this website will be accessed, what is the most efficient architecture? That's just one of the questions I asked during the design stage. Overall, I am pleased with the architecture I used for the web application, and I am also glad to have learnt so much about so many AWS services, which will undoubtedly become useful in my career.

In terms of time management, I used my sprint planning up to sprint 4 where it was clarified to me that I was using their suggestions as strict requirements. Up until that point, I was pleased with my work pace and how well I stuck to the planning. After that, there was a lot of trial and error, so the sprint planning proved pointless, therefore I used a classic "To-do" list in Notion. I found it incredibly useful, and I was producing outputs at the same rate as sprint planning, so I stuck to using the "To-do" list until the end of the development process.

If I was to redo this project, I would introduce formal testing for the data processing job. While the scale I am working on is quite small, and the number of instances has been below 5 at all times during development, in a real-life situation where this is deployed to be used by a team, that would not be the case. Introducing formal tests would allow the system's source code to be scaled up and easily tested by another developer, not just the infrastructure it is running on.

6.4 Conclusion

I have gained a lot of knowledge and experience through developing this project on infrastructure design and development, but also time management. I was already familiar with Python but interacting with cloud services through code was new to me, and I am pleased to have added that to my skill list. Very importantly I have learnt to manage my time effectively and to persevere through technical challenges. Overall, I am pleased with the quantity and quality of the work produced and happy to add more things to my software engineer tool kit.

Glossary

AWS: “is a comprehensive, evolving cloud computing platform provided by Amazon that includes a mixture of infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS) and packaged-software-as-a-service (SaaS) offerings. AWS services can offer an organization tools such as compute power, database storage and content delivery services.” (Barney, n.d.)

AWS Lambda: “is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers.” (AWS, n.d.)

AWS S3: “is an object storage service offering industry-leading scalability, data availability, security, and performance. “ (AWS, n.d.)

AWS DynamoDB: “is a fully managed, serverless, key-value NoSQL database designed to run high-performance applications at any scale.” (AWS, n.d.)

AWS EC2: “is a part of Amazon’s cloud-computing platform, Amazon Web Services (AWS), that allows users to rent on which to run their own computer applications. “ (wikipedia, n.d.)

Availability zone: “is one or more discrete data centres with redundant power, networking, and connectivity in an AWS Region.” (AWS, n.d.)

Amazon Machine Image (AMI): “is a supported and maintained image provided by AWS that provides the information required to launch an instance.” (AWS, n.d.)

Docker: “ is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers.” (wikipedia, n.d.) d

Reference list

Amazon, 2022. *Security best practices in IAM.* [Online]
Available at: https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html
AWS, 2023. *Data protection in Amazon S3.* [Online]
Available at:
https://docs.aws.amazon.com/AmazonS3/latest/userguide/DataDurability.html
AWS, 2023. *Set Up an AWS Account and Create an Administrator User.* [Online]
Available at: https://docs.aws.amazon.com/streams/latest/dev/setting-up.html
AWS, n.d. *Amazon API Gateway pricing.* [Online]
Available at: https://aws.amazon.com/api-gateway/pricing/
AWS, n.d. *AWS Cloud Technical Essentials.* [Online]
Available at: https://www.edx.org/course/aws-cloud-technical-essentials
AWS, n.d. *AWS Global Infrastructure.* [Online]
Available at: https://aws.amazon.com/about-aws/global-infrastructure/#:~:text=The%20AWS%20Cloud%20spans%2099,%2C%20New%20Zealand%2C%20and%20Thailand.
AWS, n.d. *Containerized Applications on AWS.* [Online]
Available at: edx.org/course/containerized-applications-on-aws?index=product&queryID=1ce4f8f86ecb2bb04ef89e4abfd5fad0&position=4
CloudZero, 2023. *CloudHealth Vs. CloudCheckr (And Is There A Better Option?).* [Online]
Available at: https://www.cloudzero.com/blog/cloudhealth-vs-cloudcheckr
Docker, n.d. *Use containers to Build, Share and Run your applications.* [Online]
Available at: https://www.docker.com/resources/what-container/
Ivy, D., 2019. *Sortable Table Columns with Javascript.* [Online]
Available at: https://www.youtube.com/watch?v=Q9aYU1Ufkpk&list=PL-51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=6&t=568s
Ivy, D., 2019. *JSON Array to HTML Table with Javascript.* [Online]
Available at: https://www.youtube.com/watch?v=XmdOZ5NSqb8&list=PL-51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=1
Ivy, D., 2019. *Search/Filter Table Data with Javascript.* [Online]
Available at: https://www.youtube.com/watch?v=DzXmAKdEYIs&list=PL-51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=4
Kubernetes, n.d. *Overview.* [Online]
Available at: https://kubernetes.io/docs/concepts/overview/
Rotenstein, J., 2019. *Filtering method.* [Online]
Available at: https://stackoverflow.com/a/69147625
Santos, A., 2021. *Body=json.dumps().* [Online]
Available at: https://stackoverflow.com/a/62839445
scrumguides.org, 2020. *The 2020 Scrum Guide.* [Online]
Available at: https://scrumguides.org/scrum-guide.html#the-sprint
Venters, W., 2012. *A Critical Review of Cloud Computing: Researching Desires and Realities.* [Online]
Available at: https://doi.org/10.1057/jit.2012.17
Wesley, A., 2002. *Test-Driven Development by Example.* s.l.:s.n.

Whitley, E. A., 2012. *A critical review of cloud computing: researching desires and realities.* [Online]
Available at: https://doi.org/10.1057/jit.2012.17

Appendices

Appendix A

PROJECT PROPOSAL

Problem to be solved

The project covers the lack of a software system that audits NaturalMotion's Amazon Web Services (AWS) estate of machine instances and images. This is to help the engineering team monitor the age of an instance along with various useful data about it (availability zone, platform, private IP address etc), as well as the respective image and date it was created.

Project objectives

This project will consist of a web application that will display various data about the instances (EC2) and AMIs (Amazon Machine Image) extracted by scanning an AWS (Amazon, n.d.) account. The web interface will allow for filtering the instances and images shown by day and their age, and various other applicable filters like system or availability zones.

Other data to be extracted upon scanning an account include: the hostname, status, account, availability zone, profile which created the instance, instance ID, platform, date instance was launched, public and private IP. Among this data, the respective AMI used to launch the instance is also extracted and date it was created. This data will be displayed when the user will choose to see more details about a particular item via a button or dropdown. Another objective is to prepare the above data daily in batch mode and written to a DynamoDB database, another service offered by AWS. And because the data is to be extracted and processed daily, I will use Kubernetes (Kubernetes, n.d.) to schedule the job accordingly.

The web application will run in Kubernetes which will expose the necessary ports for sending and receiving requests. All elements of the system, Kubernetes service, Docker (Inc., n.d.) containers, the network where they will all exist and all AWS services will be managed with Terraform (HashiCorp, n.d.).

Project beneficiaries

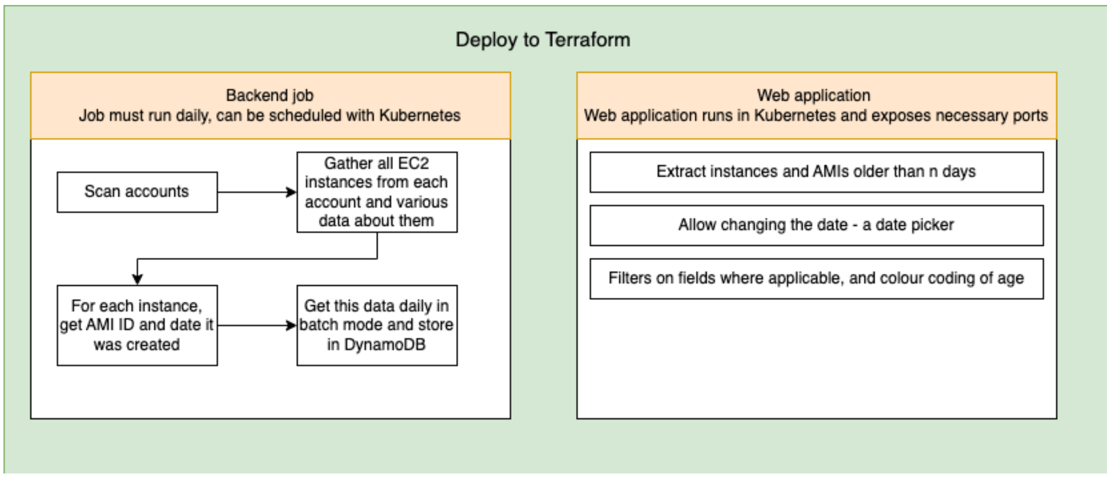
The NaturalMotion server team will benefit from this tool by being able to keep a close eye on the resources currently is use in the AWS estate, to terminate machine instances and images older than a certain number of days. The team will be able to keep instances fresh, patched and running for no longer than 30 days, as well ensure EC2 instances are not launched using an image created over 60 days ago.

Stakeholders and senior leadership would also directly benefit from the server team optimising their AWS resources by observing a potentially smaller AWS bill at the end of the month, therefore increasing the team's performance and success.

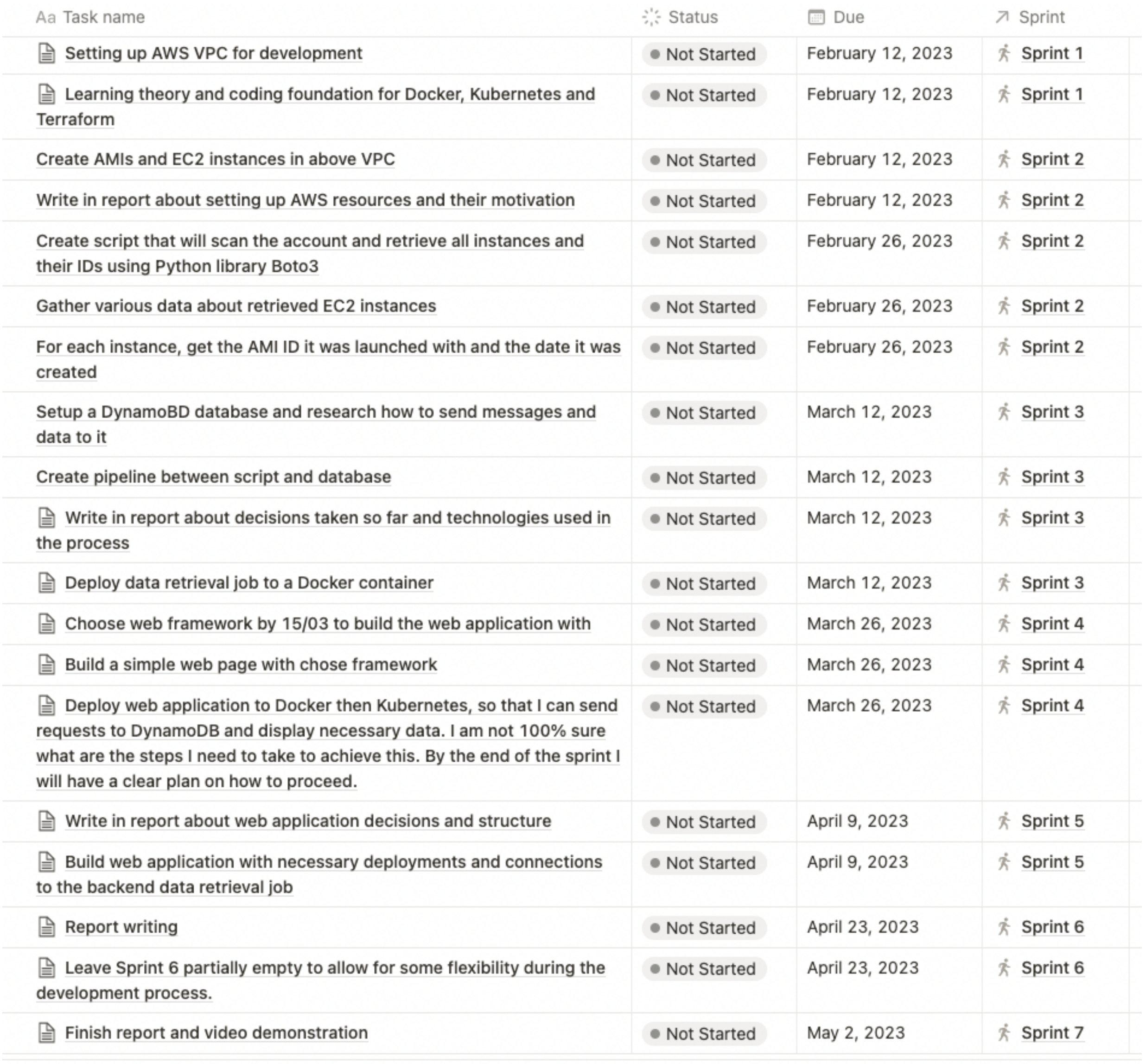
As the server team's productivity will improve with the help of this tool, NaturalMotion customers will indirectly benefit from it too. Optimisations and improvements to a process will undoubtedly bring a better product to customers, making their experience as intended by the product owner.

Work plan

I will use an agile methodology to manage my project, more specifically using scrum sprints. This framework will help me keep track of tasks and deliver an output frequently, while also allowing for some flexibility if necessary. The scrum framework is also the most widely used in the computer software industry (Santo, 2022), making it a great learning opportunity for my career. I will use the application Notion to apply this methodology to my project planning, as well as the wiki from Atlassian (Atlassian, n.d.), the developers of Jira, to learn and follow the best practices. Looking at the cycle of a sprint there are four steps: sprint planning, daily scrum, sprint review and sprint retro. I did sprint planning for the whole length of the project with the help of Notion, giving my best judgement as to how long it will take for me to learn and implement a feature. During the daily scrum a team will get together and discuss if the work is on track. I am working by myself but at the start of each working day I will compare my progress against the proposed sprint plan I made. A sprint review and retrospective will take place at the end of each sprint, and it will be an opportunity for me to reflect on the pace of my work, make any necessary adjustments and identify areas of improvement.


Deploy to Terraform
Backend job
Job must run daily, can be scheduled with Kubernetes
Scan accounts
Gather all EC2 instances from each account and various data about them
For each instance, get AMI ID and date it was created
Get this data daily in batch mode and store in DynamoDB
Web application
Web application runs in Kubernetes and exposes necessary ports
Extract instances and AMIs older than n days
Allow changing the date - a date picker
Filters on fields where applicable, and colour coding of age

The diagram above is the most high-level view of the project and what the requirements are. This is to help myself and the reader go through the sprint planning. Between 30TH of January 2023 and 25th of April 2023 when the source code is due to be submitted on Moodle there are 12 weeks and 2 days, which are split to 6 sprints of two weeks length each. Similarly, the project report is due a week later, creating sprint 7 which is just a week long. The length of a sprint must be between one week and at most four weeks (scrumguides.org, 2020), but for myself and this project I chose two. Two weeks is the right amount of time to have the flexibility of when to start and end a particular task, but also give me the opportunity to frequently reflect on the previous sprint, what could be improved and what to keep doing. With each sprint I will write in the report these observations, which will also help me avoid writing these important details last minute.



Aa Task name	Status	Due	Sprint
Setting up AWS VPC for development	Not Started	February 12, 2023	Sprint 1
Learning theory and coding foundation for Docker, Kubernetes and Terraform	Not Started	February 12, 2023	Sprint 1
Create AMIs and EC2 instances in above VPC	Not Started	February 12, 2023	Sprint 2
Write in report about setting up AWS resources and their motivation	Not Started	February 12, 2023	Sprint 2
Create script that will scan the account and retrieve all instances and their IDs using Python library Boto3	Not Started	February 26, 2023	Sprint 2
Gather various data about retrieved EC2 instances	Not Started	February 26, 2023	Sprint 2
For each instance, get the AMI ID it was launched with and the date it was created	Not Started	February 26, 2023	Sprint 2
Setup a DynamoBD database and research how to send messages and data to it	Not Started	March 12, 2023	Sprint 3
Create pipeline between script and database	Not Started	March 12, 2023	Sprint 3
Write in report about decisions taken so far and technologies used in the process	Not Started	March 12, 2023	Sprint 3
Deploy data retrieval job to a Docker container	Not Started	March 12, 2023	Sprint 3
Choose web framework by 15/03 to build the web application with	Not Started	March 26, 2023	Sprint 4
Build a simple web page with chose framework	Not Started	March 26, 2023	Sprint 4
Deploy web application to Docker then Kubernetes, so that I can send requests to DynamoDB and display necessary data. I am not 100% sure what are the steps I need to take to achieve this. By the end of the sprint I will have a clear plan on how to proceed.	Not Started	March 26, 2023	Sprint 4
Write in report about web application decisions and structure	Not Started	April 9, 2023	Sprint 5
Build web application with necessary deployments and connections to the backend data retrieval job	Not Started	April 9, 2023	Sprint 5
Report writing	Not Started	April 23, 2023	Sprint 6
Leave Sprint 6 partially empty to allow for some flexibility during the development process.	Not Started	April 23, 2023	Sprint 6
Finish report and video demonstration	Not Started	May 2, 2023	Sprint 7

Figure 11 All tasks created for the project as of 5 Feb 2023, organised in sprints

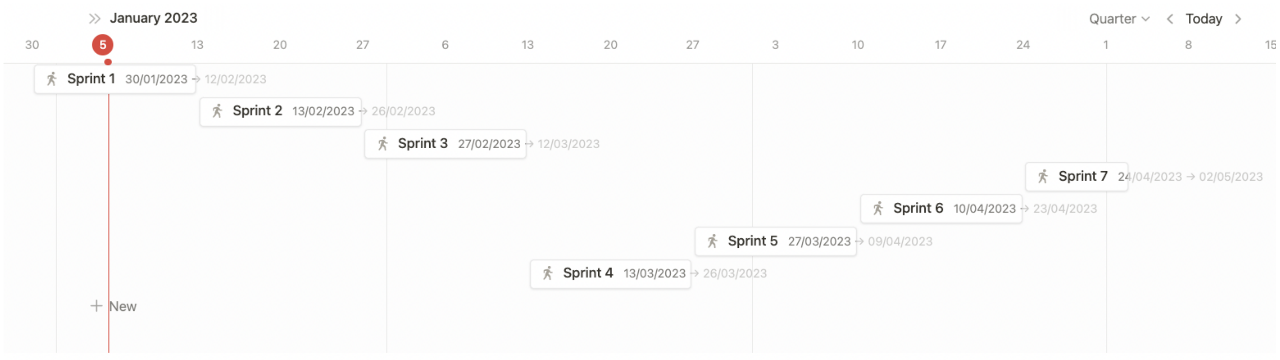

January 2023
Quarter
Today
30
5
13
20
27
6
13
20
27
3
10
17
24
1
8
15
Sprint 1 30/01/2023 → 12/02/2023
Sprint 2 13/02/2023 → 26/02/2023
Sprint 3 27/02/2023 → 12/03/2023
Sprint 7 24/04/2023 → 02/05/2023
Sprint 6 10/04/2023 → 23/04/2023
Sprint 5 27/03/2023 → 09/04/2023
Sprint 4 13/03/2023 → 26/03/2023
New

Figure 12 A timeline of all sprints

Looking at figure 1, it is a to-do list of necessary tasks to make the project successful in the end. There is some preparation work and learning before proceeding with a certain task. For example, in Sprint 1 I must learn the foundation of Docker, Kubernetes and Terraform both in theory and practice. The backend data processing job and the web application will be deployed in Docker containers, which are to be managed by Kubernetes, while all the AWS infrastructure like VPC and database will be managed by Terraform. I must learn the best practices and how-to of all these services to implement them well and up to good engineering standards. Similarly in Sprint 4, I am not yet familiar with the detailed steps necessary to deploy the web application to Docker and so on. So, by the end of the sprint, 26th of March, I will have researched and learnt from online sources, the client contact and literature, how to solve the problem and have a solid plan to follow. As I get more familiar with the software and services I am to use, at the start of each sprint each task will have more specific steps to follow and clear goals. This will of course be recorded in the report at the end of each sprint.

Project risks

There are three perspectives to look at in terms of the risks my project will face or cause. Risks to the project that may cause it to fail or reduce my marks, risks from the project that may harm myself or others and risks to individuals involved in the project development. There are no other people involved in the project, making that risk not applicable.

There are a few risks that could negatively impact my project's performance, one of them being Amazon Web Services. While Amazon is responsible with the upkeep and maintenance of their services, and the main selling point to their customers is that you don't have maintain your own servers, there are instances of such service failing. I can reduce the effect it has on my project by following the work plan. The work plan ensures I finish all tasks within the time given as well as a buffer of about a week in Sprint 6, therefore following that will allow for any AWS outages to occur without significant impact on my project's progression.

There is also the risk of taking longer than expected to develop the backend job and applying required services like Docker, Kubernetes or Terraform. In such a situation, I plan on scrapping the use of AWS's DynamoDB to store data and instead run a database locally. This will keep my project going and performing as intended, while not failing it entirely.

The risk of unauthorised access to the network is to be considered too. I will use AWS's Virtual Private Cloud to create a network where the project will exist and run, therefore I can utilise AWS's security features (security groups, access roles, security tokens) to limit access to the network. With the bonus of Amazon themselves keeping the physical architecture of my network secure. In case of a breach and source code being stolen, I will use GitHub to remotely and securely store the code to avoid my project failing entirely because of it.

Staying on the topic of version control, the risk of theft is high as I will be travelling on public transport during rush hour, so to avoid such a scenario flooring my project I will store everything using GitHub, report included. To try to mitigate the risk, my bag has a tracking device in it, and keep my laptop close whenever I'm not using it directly. In terms of the risk of disk failure, using version control will cover that risk, and holding my laptop well when moving around will prevent me dropping it.

One final risk to the project is failing to deliver the project to the client. By following the work plan and reflecting on my performance at the end of each sprint will ensure I deliver the project and keep the risk to a minimum.

In terms of risk to others that my project may cause, it is minimal because I will not be interacting with official Natural Motion systems or data. It will be built using my own AWS resources and open-source libraries, so there is no security risk regarding customer data or company intelligence. Once the tool is given to the client, they have the expertise necessary to apply specialised security to it.

Bibliography

Amazon, 2022. *Security best practices in IAM.* [Online]
Available at: https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html
Amazon, n.d. *Amazon Web Services.* [Online]
Available at: https://aws.amazon.com/
Atlassian, n.d. *A guide to scrum.* [Online]
Available at: https://www.atlassian.com/agile/scrum
AWS, 2023. *Data protection in Amazon S3.* [Online]
Available at: https://docs.aws.amazon.com/AmazonS3/latest/userguide/DataDurability.html
AWS, 2023. *Set Up an AWS Account and Create an Administrator User.* [Online]
Available at: https://docs.aws.amazon.com/streams/latest/dev/setting-up.html

AWS, n.d. *Amazon API Gateway pricing.* [Online]
Available at: https://aws.amazon.com/api-gateway/pricing/
AWS, n.d. *Amazon Machine Images (AMI).* [Online]
Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html
AWS, n.d. *AWS Cloud Technical Essentials.* [Online]
Available at: https://www.edx.org/course/aws-cloud-technical-essentials
AWS, n.d. *AWS DynamoDB.* [Online]
Available at: https://aws.amazon.com/dynamodb/
AWS, n.d. *AWS Global Infrastructure.* [Online]
Available at: https://aws.amazon.com/about-aws/global-infrastructure/#:~:text=The%20AWS%20Cloud%20spans%2099,%2C%20New%20Zealand%2C%20and%20Thailand.
AWS, n.d. *AWS Lambda.* [Online]
Available at: https://aws.amazon.com/lambda/
AWS, n.d. *AWS S3.* [Online]
Available at: https://aws.amazon.com/s3/
AWS, n.d. *Containerized Applications on AWS.* [Online]
Available at: edx.org/course/containerized-applications-on-aws?index=product&queryID=1ce4f8f86ecb2bb04ef89e4abfd5fad0&position=4
AWS, n.d. *Regions and Availability Zones.* [Online]
Available at: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/
Barney, N., n.d. *Amazon Web Services.* [Online]
Available at: https://www.techtarget.com/searchaws/definition/Amazon-Web-Services
CloudZero, 2023. *CloudHealth Vs. CloudCheckr (And Is There A Better Option?).* [Online]
Available at: https://www.cloudzero.com/blog/cloudhealth-vs-cloudcheckr
Docker, n.d. *Use containers to Build, Share and Run your applications.* [Online]
Available at: https://www.docker.com/resources/what-container/
HashiCorp, n.d. *Terraform.* [Online]
Available at: https://www.terraform.io/
Inc., D., n.d. *Docker.* [Online]
Available at: https://www.docker.com/
Ivy, D., 2019. *Sortable Table Columns with Javascript.* [Online]
Available at: https://www.youtube.com/watch?v=Q9aYU1Ufkpk&list=PL-51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=6&t=568s
Ivy, D., 2019. *JSON Array to HTML Table with Javascript.* [Online]
Available at: https://www.youtube.com/watch?v=XmdOZ5NSqb8&list=PL-51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=1
Ivy, D., 2019. *Search/Filter Table Data with Javascript.* [Online]
Available at: https://www.youtube.com/watch?v=DzXmAKdEYIs&list=PL-51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=4
Kubernetes, n.d. *Kubernetes.* [Online]
Available at: https://kubernetes.io/
Kubernetes, n.d. *Overview.* [Online]
Available at: https://kubernetes.io/docs/concepts/overview/
Rotenstein, J., 2019. *Filtering method.* [Online]
Available at: https://stackoverflow.com/a/69147625
Santo, D. E., 2022. *op 5 main Agile methodologies.* [Online]
Available at: https://www.xpand-it.com/blog/top-5-agile-methodologies/
Santos, A., 2021. *Body=json.dumps().* [Online]
Available at: https://stackoverflow.com/a/62839445

scrumguides.org, 2020. *The 2020 Scrum Guide.* [Online]
Available at: https://scrumguides.org/scrum-guide.html#the-sprint
Venters, W., 2012. *A Critical Review of Cloud Computing: Researching Desires and Realities.* [Online]
Available at: https://doi.org/10.1057/jit.2012.17
Wesley, A., 2002. *Test-Driven Development by Example.* s.l.:s.n.
Whitley, E. A., 2012. *A critical review of cloud computing: researching desires and realities.* [Online]
Available at: https://doi.org/10.1057/jit.2012.17
wikipedia, n.d. *Amazon Elastic Compute Cloud.* [Online]
Available at: https://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud
wikipedia, n.d. *Docker (software).* [Online]
Available at: https://en.wikipedia.org/wiki/Docker_(software)

ETHICS CHECKLIST

A.1 If you answer YES to any of the questions in this block, you must apply to an appropriate external ethics committee for approval and log this approval as an External Application through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
1.1	Does your research require approval from the National Research Ethics Service (NRES)? *e.g. because you are recruiting current NHS patients or staff?* *If you are unsure try - https://www.hra.nhs.uk/approvals-amendments/what-approvals-do-i-need/*	**NO**
1.2	Will you recruit participants who fall under the auspices of the Mental Capacity Act? *Such research needs to be approved by an external ethics committee such as NRES or the Social Care Research Ethics Committee - http://www.scie.org.uk/research/ethics-committee/*	**NO**
1.3	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation? *Such research needs to be authorised by the ethics approval system of the National Offender Management Service.*	**NO**
A.2 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee, you must apply for approval from the Senate Research Ethics Committee (SREC) through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
2.1	Does your research involve participants who are unable to give informed consent? *For example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf.*	**NO**
2.2	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	**NO**
2.3	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	**NO**
2.4	Does your project involve participants disclosing information about special category or sensitive subjects? *For example, but not limited to: racial or ethnic origin; political opinions; religious beliefs; trade union membership; physical or mental health; sexual life; criminal offences and proceedings*	**NO**

2.5	Does your research involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning that affects the area in which you will study? *Please check the latest guidance from the FCO - http://www.fco.gov.uk/en/*	**NO**
2.6	Does your research involve invasive or intrusive procedures? *These may include, but are not limited to, electrical stimulation, heat, cold or bruising.*	**NO**
2.7	Does your research involve animals?	**NO**
2.8	Does your research involve the administration of drugs, placebos or other substances to study participants?	**NO**
A.3 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee or the SREC, you must apply for approval from the Computer Science Research Ethics Committee (CSREC) through Research Ethics Online - https://ethics.city.ac.uk/ **Depending on the level of risk associated with your application, it may be referred to the Senate Research Ethics Committee.**		*Delete as appropriate*
3.1	Does your research involve participants who are under the age of 18?	**NO**
3.2	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? *This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.*	**NO**
3.3	Are participants recruited because they are staff or students of City, University of London? *For example, students studying on a particular course or module.* *If yes, then approval is also required from the Head of Department or Programme Director.*	**NO**
3.4	Does your research involve intentional deception of participants?	**NO**
3.5	Does your research involve participants taking part without their informed consent?	**NO**
3.5	Is the risk posed to participants greater than that in normal working life?	**NO**
3.7	Is the risk posed to you, the researcher(s), greater than that in normal working life?	**NO**
A.4 If you answer YES to the following question and your answers to all other questions in sections A1, A2 and A3 are NO, then your project is deemed to be of MINIMAL RISK. **If this is the case, then you can apply for approval through your supervisor under PROPORTIONATE REVIEW. You do so by completing PART B of this form.** **If you have answered NO to all questions on this form, then your project does not require ethical approval. You should submit and retain this form as evidence of this.**		*Delete as appropriate*
4	Does your project involve human participants or their identifiable personal data? *For example, as interviewees, respondents to a survey or participants in testing.*	**NO**

PART B: Ethics Proportionate Review Form

B.1 The following questions must be answered fully. **All grey instructions must be removed.**		*Delete as appropriate*
1.1.	Will you ensure that participants taking part in your project are fully informed about the purpose of the research?	**YES / NO**
1.2	Will you ensure that participants taking part in your project are fully informed about the procedures affecting them or affecting any information collected	**YES / NO**

	about them, including information about how the data will be used, to whom it will be disclosed, and how long it will be kept?	
1.3	When people agree to participate in your project, will it be made clear to them that they may withdraw (i.e. not participate) at any time without any penalty?	**YES / NO**
1.4	Will consent be obtained from the participants in your project? Consent from participants will be necessary if you plan to involve them in your project or if you plan to use identifiable personal data from existing records. "Identifiable personal data" means data relating to a living person who might be identifiable if the record includes their name, username, student id, DNA, fingerprint, address, etc. *If YES, you must attach drafts of the participant information sheet(s) and consent form(s) that you will use in section B.3 or, in the case of an existing dataset, provide details of how consent has been obtained.* *You must also retain the completed forms for subsequent inspection.* *Failure to provide the completed consent request forms will result in withdrawal of any earlier ethical approval of your project.*	**YES / NO**
1.5	Have you made arrangements to ensure that material and/or private information obtained from or about the participating individuals will remain confidential?	**YES / NO**

B.2 If the answer to the following question (B2) is YES, you must provide details		*Delete as appropriate*
2	Will the research be conducted in the participant's home or other non-University location? *If **YES**, you must provide details of how your safety will be ensured.*	**YES / NO**

B.3 Attachments **ALL of the following documents MUST be provided to supervisors if applicable.** **All must be considered prior to final approval by supervisors.** **A written record of final approval must be provided and retained.**	***YES***	***NO***	***Not Applicable***
Details on how safety will be assured in any non-University location, including risk assessment if required (see B2)			
Details of arrangements to ensure that material and/or private information obtained from or about the participating individuals will remain confidential (see B1.5) *Any personal data must be acquired, stored and made accessible in ways that are GDPR compliant.*			
Full protocol for any workshops or interviews**			
Participant information sheet(s)**			
Consent form(s)**			
Questionnaire(s)** *sharing a Qualtrics survey with your supervisor is recommended.*			
Topic guide(s) for interviews and focus groups**			
Permission from external organisations or Head of Department** *e.g. for recruitment of participants*			

APPENDIX B: Reuse summary

Appendix B1: index.html

Written by me with some re used parts.

Adapted from Dennis Ivy (2019) Search/Filter Table Data with Javascript. Available at: https://www.youtube.com/watch?v=DzXmAKdEYIs&list=PL51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=4 for lines: 87-91, 149-153

Adapted from Dennis Ivy (2019) Sortable Table Columns with Javascript. Available at: https://www.youtube.com/watch?v=Q9aYU1Ufkpk&list=PL51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=6&t=568s for lines: 98-117, 160-178

Libraries used: Bootstrap, jQuery

Appendix B2: script.js

Written by me with some re used parts.

Adapted from Dennis Ivy (2019) Search/Filter Table Data with Javascript. Available at: https://www.youtube.com/watch?v=DzXmAKdEYIs&list=PL51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=4 for lines: 105-119, 122-133

"for" loop adapted from Dennis Ivy (2019) JSON Array to HTML Table with Javascript. Available at https://www.youtube.com/watch?v=XmdOZ5NSqb8&list=PL51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=1 for lines 36-63, 76-96

Libraries used: moment.js

Appendix B3: auditingtool-updateInstancesAndImagesBuckets.py

Written by me with some re used parts.

Function was automatically generated when I initiated the Lambda function using a template offered by AWS. See figure 8 in 5.6.1. Lines 14 to 21.

Used filtering method by John Rotenstein (September 2019) Accessible at https://stackoverflow.com/a/69147625 . Lines 31-38.

Used the "Body=json.dumps(data)" snippet by Alexander Santos (July 2021). Accessible at https://stackoverflow.com/a/62839445 . Line 96.

Libraries used: Boto3, json and datetime

Appendix B4: website-request-handler.py

Entirely written by me.
Libraries used: json and boto3.

APPENDIX C: Source code

APPENDIX C1: web-app/ index.html

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.3/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.29.3/moment.min.js"></script>
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
<script src="script.js"></script>

<head>
  <meta charset="UTF-8">
  <title> Auditing tool web app</title>
  <script>
    var callAPI = ()=>{
      var myHeaders = new Headers()
      // add content type header to object
      myHeaders.append("Content-Type", "application/json")

      // create a JSON object with parameters for API call and store in a variable
      var GETRequest = {
        method : 'GET',
        headers : myHeaders,
        redirect: 'follow'
      }

      fetch("https://09zvg0tsra.execute-api.us-east-1.amazonaws.com/dev", GETRequest)
        .then(response => response.text())
        .then(result => response_handler(JSON.parse(result).body))
        .catch(error => console.log('error', error));
    }
    callAPI()
  </script>
```

```
</head>
<body>

<div class="jumbotron text-center">
  <h2> Auditing tool for AWS resources </h2>
  <p> You can find various data about currently running instances(EC2) and custom-made
    Amazon Machine Images(AMI). Search by IDs and sort chronologically
    by date.</p>
</div>

<div class="container">

  <h2> Currently running instances </h2>
  <div class="row">
    <div class="card card-body">
      <input id="searchInputInstances" class="form-control" type="text" placeholder="Type the
instance or image ID">
    </div>
  </div>

  <div class="row table-responsive">
    <table class="table table-bordered">
      <tr class="bg-info"> </tr>
      <th> Instance ID </th>
      <th id="instanceTableDateColumn" data-order="desc"> Date launched &#x2191;</th>
      <th> Image ID </th>
      <th> Availability zone </th>
      <th> Private IP address </th>
      <th> Public IP address</th>
      <th> Platform </th>
      <th> Tags </th>
      <tbody id="instanceTable">
      </tbody>
```

```
        <script>
          instancesDict = []
          AMIsDict = []
          function response_handler(response) {
            instancesDict = JSON.parse(response).instances
            AMIsDict = JSON.parse(response).amis
            instancesDict = JSON.parse(instancesDict)
            AMIsDict = JSON.parse(AMIsDict)

            // On start up, the data must be sorted desc by date.
            instancesDict = instancesDict.sort((a,b) =>
sortByDate(a['date_launched'],b['date_launched'], 'desc'))
            AMIsDict = AMIsDict.sort((a,b) => sortByDate(a['creation_date'],b['creation_date'],
'desc'))

            buildInstancesTable(instancesDict)
            buildCustomImagesTable(AMIsDict);
          }



          /**
           * Adapted from Dennis Ivy (2019) Search/Filter Table Data with Javascript
           * Available at: https://www.youtube.com/watch?v=DzXmAKdEYIs&list=PL-
51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=4
           * Used his methodology as skeleton for my implementation of the search bar feature
           */
          $('#searchInputInstances').on('keyup', function(){
            var inputVal = $(this).val()
            var filteredRows = searchInstancesTable(inputVal, instancesDict);
            buildInstancesTable(filteredRows);
          })

          /**
           * Adapted from Dennis Ivy (2019) Sortable Table Columns with Javascript
```

```
            * Available at https://www.youtube.com/watch?v=Q9aYU1Ufkpk&list=PL-
51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=6&t=568s
            * Used his methodology as skeleton for my implementation of the sorting date
            */
            $('#instanceTableDateColumn').on('click', function(){
              var column = $(this).data('column')
              var order = $(this).data('order')

              if (order == 'desc'){
                $(this).data('order', 'asc')
                dateHeader = document.getElementById('instanceTableDateColumn').innerHTML =
`Date launched &#x2193;`

                // Only sorting by the date launched
                instancesDict = instancesDict.sort((a,b) =>
sortByDate(a['date_launched'],b['date_launched'], 'asc'))
                buildInstancesTable(instancesDict)

              } else {
                $(this).data('order', 'desc')
                dateHeader = document.getElementById('instanceTableDateColumn').innerHTML =
`Date launched &#x2191;`

                instancesDict = instancesDict.sort((a,b) =>
sortByDate(a['date_launched'],b['date_launched'], 'desc'))
                buildInstancesTable(instancesDict)
              }
            })

          </script>
        </table>
      </div>
    </div>
```

```
<div class="container">
   <h2> Custom AMIs  </h2>
   <div class="row">
      <div class="card card-body">
         <input id="searchInputImages" class="form-control" type="text" placeholder="Type the
image ID">
      </div>
   </div>

   <div class="row">
      <table class="table table-bordered">
         <tr class="bg-info"> </tr>
         <th> Image ID </th>
         <th id="imagesTableDateColumn" data-order="desc"> Date created &#x2191;</th>
         <th> Image name </th>

         <tbody id="AMIsTable">
         </tbody>

         <script>
            /**
             * Adapted from Dennis Ivy (2019) Search/Filter Table Data with Javascript
             * Available at: https://www.youtube.com/watch?v=DzXmAKdEYIs&list=PL-
51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=4
             * Used his methodology as skeleton for my implementation of the search bar feature
             */
            $('#searchInputImages').on('keyup', function(){
               var inputVal = $(this).val()
               var filteredRows = searchImagesTable(inputVal, AMIsDict);
               buildCustomImagesTable(filteredRows);
            })

            /**
```

```
             * Adapted from Dennis Ivy (2019) Sortable Table Columns with Javascript
             * Available at https://www.youtube.com/watch?v=Q9aYU1Ufkpk&list=PL-
51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=6&t=568s
             * Used his methodology as skeleton for my implementation of the sorting date
             */
            $('#imagesTableDateColumn').on('click', function(){
               var column = $(this).data('column')
               var order = $(this).data('order')

               if (order == 'desc'){
                  $(this).data('order', 'asc')
                  dateHeader = document.getElementById('imagesTableDateColumn').innerHTML =
`Date created &#x2193;`

                  AMIsDict = AMIsDict.sort((a,b) => sortByDate(a['creation_date'],b['creation_date'],
'asc'))
                  buildCustomImagesTable(AMIsDict)

               } else {
                  $(this).data('order', 'desc')
                  dateHeader = document.getElementById('imagesTableDateColumn').innerHTML =
`Date created &#x2191;`

                  AMIsDict = AMIsDict.sort((a,b) => sortByDate(a['creation_date'],b['creation_date'],
'desc'))
                  buildCustomImagesTable(AMIsDict)
               }
            })

         </script>
      </table>
   </div>
</div>
</body>
```

```
</html>
```

APPENDIX C2: web-app/ script.js

```
function isOlderThanXDays(ageLimit, date) {
   date = date.split(',')[0]
   var dateOfInstance = moment(date, "DD/MM/YYYY")
   var dateXDaysAgo = moment().subtract(ageLimit, 'days')

   return dateOfInstance.isBefore(dateXDaysAgo)
}
function sortByDate(a,b, descOrAsc) {
   if (descOrAsc == 'desc') {
      if (a > b) {
         return 1
      } else {
         return -1
      }
   } else if (descOrAsc == 'asc') {
      if (a < b) {
         return 1
      } else {
         return -1
      }
   } else {
      throw "Invalid option for descending or ascending"
   }
}



function buildInstancesTable(instanceData) {
   var table = document.getElementById('instanceTable');
   table.innerHTML = '';

   /**
```

```
 * "for" loop adapted from Dennis Ivy (2019) JSON Array to HTML Table with Javascript
 * Available at https://www.youtube.com/watch?v=XmdOZ5NSqb8&list=PL-
51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=1
 * Used his methodology as skeleton for my implementation.
 */
for (var i = 0; i < instanceData.length; i++) {
  var olderThan30Days = isOlderThanXDays(30, instanceData[i].date_launched);
  if (olderThan30Days) {
    var row = `<tr>
            <td> ${instanceData[i].instance_id}</td>
            <td bgcolor="#f59396"> ${instanceData[i].date_launched}</td>
            <td> ${instanceData[i].ami_id}</td>
            <td> ${instanceData[i].availability_zone}</td>
            <td> ${instanceData[i].private_ip_address}</td>
            <td> ${instanceData[i].public_ip_address}</td>
            <td> ${instanceData[i].platform}</td>
            <td> ${JSON.stringify(instanceData[i].tags)}</td>
            </tr>`
    table.innerHTML += row;
  } else {
    var row = `<tr>
            <td> ${instanceData[i].instance_id}</td>
            <td  bgcolor="#a4dbb2"> ${instanceData[i].date_launched}</td>
            <td> ${instanceData[i].ami_id}</td>
            <td> ${instanceData[i].availability_zone}</td>
            <td> ${instanceData[i].private_ip_address}</td>
            <td> ${instanceData[i].public_ip_address}</td>
            <td> ${instanceData[i].platform}</td>
            <td> ${JSON.stringify(instanceData[i].tags)}</td>
            </tr>`
    table.innerHTML += row;
  }
}
```

```
}


function buildCustomImagesTable(AMIData) {
   var table = document.getElementById('AMIsTable');
   table.innerHTML = '';

   /**
    * "for" loop adapted from Dennis Ivy (2019) JSON Array to HTML Table with Javascript
    * Available at https://www.youtube.com/watch?v=XmdOZ5NSqb8&list=PL-
51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=1
    */
   for (var i = 0; i < AMIData.length; i++) {
      var olderThan30Days = isOlderThanXDays(30, AMIData[i].creation_date);

      if (olderThan30Days){
         var row = `<tr>
                     <td> ${AMIData[i].image_id}</td>
                     <td bgcolor="#f59396"> ${AMIData[i].creation_date}</td>
                     <td> ${AMIData[i].image_name}</td>
                     </tr>`
         table.innerHTML += row;
      } else {
         var row = `<tr>
                     <td> ${AMIData[i].image_id}</td>
                     <td bgcolor="#a4dbb2"> ${AMIData[i].creation_date}</td>
                     <td> ${AMIData[i].image_name}</td>
                     </tr>`
         table.innerHTML += row;
      }


   }
}
```

```
/**
 * Adapted from Dennis Ivy (2019) Search/Filter Table Data with Javascript
 * Available at: https://www.youtube.com/watch?v=DzXmAKdEYIs&list=PL-
51WBLyFTg1l3K0aTH0uX6PzgaLfzJBK&index=4
 * Used his methodology as skeleton for my implementation of the functions
 * serchInstancesTable and searchImagesTable.
 */
function searchInstancesTable(searchValue, data){
   var filteredData = []
   searchValue = searchValue.toLowerCase(); // No case sensitivity
   for (var i = 0; i < data.length; i++) {
      var instance_id = data[i].instance_id.toLowerCase()
      var ami_id = data[i].ami_id.toLowerCase()

      if (instance_id.includes(searchValue)){
         filteredData.push(data[i])
      }
      else if (ami_id.includes(searchValue)){
         filteredData.push(data[i])
      }
   }
   return filteredData
}

function searchImagesTable(searchValue, data){
   var filteredData = []
   searchValue = searchValue.toLowerCase(); // No case sensitivity
   for (var i = 0; i < data.length; i++) {
      var imageID = data[i].image_id.toLowerCase();

      if (imageID.includes(searchValue)){
         filteredData.push(data[i])
      }
```

```
    }
    return filteredData;
}
```

APPENDIX C3: python-scripts-from-awsLambda/auditingtool-updateInstancesAndImagesBuckets.py

```
import boto3
import json
from datetime import datetime

ec2 = boto3.client('ec2', region_name='us-east-1')
s3 = boto3.client('s3', region_name='us-east-1')
s3_resource = boto3.resource('s3')

# This function was automatically generated when I initiated
# the Lambda function using a template from them. The template
# is named "Create a microservice that interacts with a DDB table".
# This function is the only thing I kept from that template, I only changed the
# status code from '200' to '202'.
def respond(err, res=None):
    return {
        'statusCode': '400' if err else '202',
        'body': err.message if err else json.dumps(res),
        'headers': {
            'Content-Type': 'application/json',
        },
    }

# The main function executed when the Lambda function is invoked. It makes the
# necessary connections to AWS services, gets the instances and images data,
# and stores it in two separate S3 buckets.
def lambda_handler(event, context):
    # Get instances that are running
    # Used filtering method by John Rotenstein (September 2019)
    # Accessible at https://stackoverflow.com/a/69147625
```

```
all_instances = ec2.describe_instances(
    Filters=[
        {
            'Name': 'instance-state-name',
            'Values': [
                'running',
            ]
        },
    ]
)

instance_data = []
for r in all_instances['Reservations']:
    for instance in r['Instances']:
        instance_id = instance['InstanceId']
        ami_id = instance['ImageId']
        date_launched = instance['LaunchTime'].strftime("%d/%m/%Y, %H:%M:%S")
        availability_zone = instance['Placement']['AvailabilityZone']
        private_ip_address = instance['PrivateIpAddress']
        public_ip_address = instance['PublicIpAddress']
        platform = instance['PlatformDetails']
        list_of_tags = instance['Tags']
        instance_data.append({
            'instance_id': instance_id,
            'ami_id': ami_id,
            'date_launched': date_launched,
            'availability_zone': availability_zone,
            'private_ip_address': private_ip_address,
            'public_ip_address': public_ip_address,
            'platform': platform,
            'tags': list_of_tags
        })

# Check if running-ec2-instances.json already exists in the bucket, delete it
```

```
    # if so. This is to refresh the resources and delete entries that have been terminated.
    for file in s3_resource.Bucket('running-instances-data-if1').objects.all():
        if file.key == 'running-ec2-instances.json':
            s3.delete_object(Bucket='running-instances-data-if1',
                        Key='running-ec2-instances.json')

    # Get data of AMIs that are custom-made by the user
    all_images = ec2.describe_images(Owners=['self'])
    images_data = []
    for image in all_images['Images']:
        image_id = image['ImageId']
        image_name = image['Name']
        creation_date = datetime.strptime(image["CreationDate"], "%Y-%m-%dT%H:%M:%S.%fZ")
        creation_date = creation_date.strftime("%d/%m/%Y, %H:%M:%S")

        images_data.append({
            'image_id': image_id,
            'image_name': image_name,
            'creation_date': creation_date
        })

    # Check if custom-amis-data.json already exists in the bucket, delete it
    # if so. This is to refresh the resources and delete entries that have been terminated.
    for file in s3_resource.Bucket('custom-amis-data-if1').objects.all():
        if file.key == 'custom-amis-data.json':
            s3.delete_object(Bucket='custom-amis-data-if1',
                        Key='custom-amis-data.json')

    # Store data in S3 bucket
    try:
        # Used the "Body=json.dumps(data)" snippet by Alexander Santos (July 2021)
        # Accessible at https://stackoverflow.com/a/62839445
        instances_response = s3.put_object(Body=json.dumps(instance_data), Bucket='running-
instances-data-if1',
                                    Key='running-ec2-instances.json')
        amis_response = s3.put_object(Body=json.dumps(images_data), Bucket='custom-amis-data-if1',
                                Key='custom-amis-data.json')
        return respond(None, "Success")

    except:
        return respond(RuntimeError('Could not create S3 object of currently running instances or
AMis'))
```

APPENDIX C4: python-scripts-from-awsLambda/ website-request-handler.py

```
import json
import boto3

# When a GET request comes through the API, it will be directed to this function.
# This function will get the instances and images data stored in their respective
# S3 buckets, format it, and send it back to be displayed on the website.
def lambda_handler(event, context):
    s3 = boto3.client('s3', region_name='us-east-1')
    lambda_client = boto3.client('lambda', region_name='us-east-1')

    # Invoking the respective Lambda function asynchronously,
    # which will update the S3 buckets before getting the data
    # and sending it back here.
    # Adapted from AWS Boto3/Lambda/Client documentation
    # Accessible at
https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/lambda/client/invo
ke.html
    update_buckets_response = lambda_client.invoke(FunctionName='auditingtool-
updateInstancesAndImagesBuckets',
                                        InvocationType='Event')

    if (update_buckets_response['StatusCode'] == 202):
        # Get instance and image data from their respective S3 buckets.
        instances_data_response = s3.get_object(
            Bucket='running-instances-data-if1',
            Key='running-ec2-instances.json')
        instances_json_to_send = instances_data_response["Body"].read().decode()

        ami_data_response = s3.get_object(
            Bucket='custom-amis-data-if1',
            Key='custom-amis-data.json')
        ami_json_to_send = ami_data_response["Body"].read().decode()

        # Building the body of the response
        response = {
            'instances': instances_json_to_send,
            'amis': ami_json_to_send
        }

        return {
            'statusCode': 200,
            'body': json.dumps(response)
        }
    else:
        return {
            'statusCode': 400,
            'body': 'Bad Gateway'
        }
```

APPENDIX C5: Data generated

Indicating to the output outlined in 2.4.

running-ec2-instances.json:

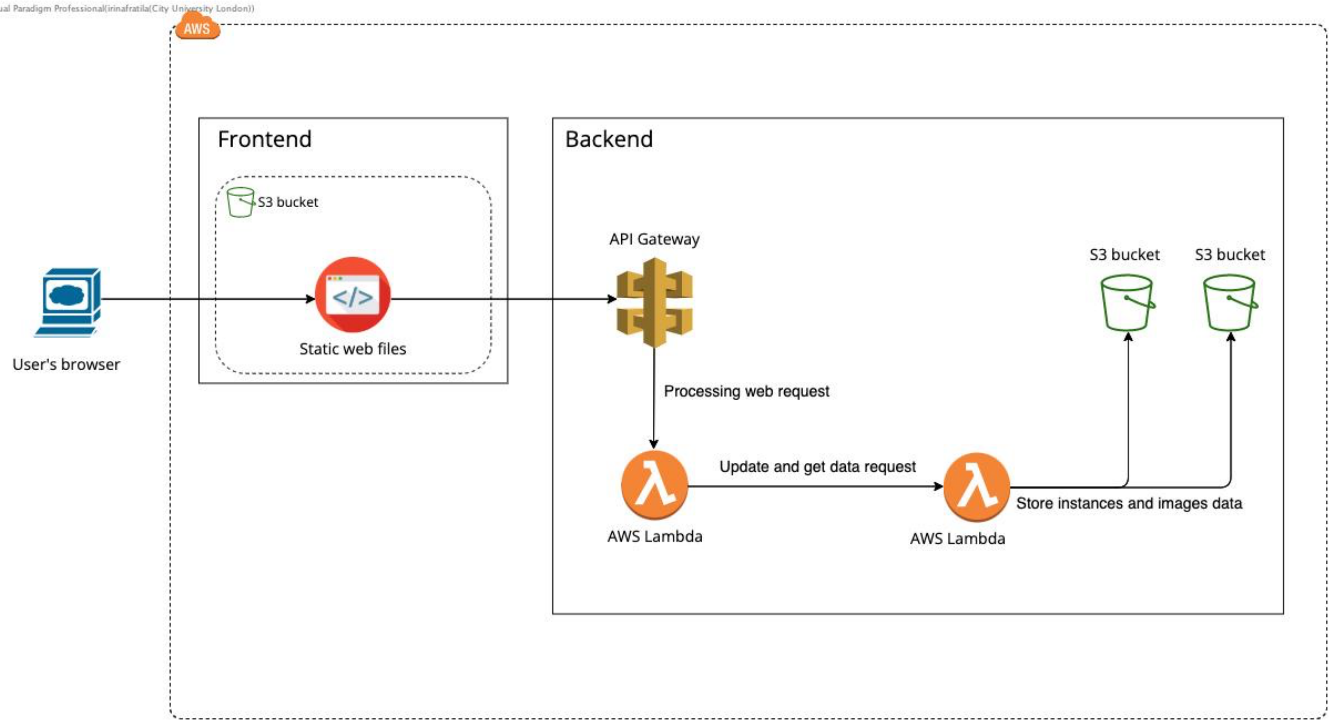
[
	{"instance_id": "i-0cb1a88ff585f35b1", "ami_id": "ami-006dcf34c09e50022", "date_launched": "04/03/2023, 11:01:34", "availability_zone": "us-east-1a", "private_ip_address": "10.0.0.17", "public_ip_address": "54.160.219.245", "platform": "Linux/UNIX", "tags": [{"Key": "Name", "Value": "instance1"}]},
	{"instance_id": "i-04a257efb240061ea", "ami_id": "ami-0cb3484c1a6f84067", "date_launched": "04/03/2023, 12:43:41", "availability_zone": "us-east-1a", "private_ip_address":

"10.0.6.72", "public_ip_address": "54.160.138.127", "platform": "Linux/UNIX", "tags": [{"Key": "Name", "Value": "instance2"}, {"Key": "NeverDelete", "Value": ""}]},
	{"instance_id": "i-0fcb9a4d62bdd45e8", "ami_id": "ami-005f9685cb30f234b", "date_launched": "27/03/2023, 20:05:35", "availability_zone": "us-east-1b", "private_ip_address": "10.0.17.74", "public_ip_address": "54.81.33.182", "platform": "Linux/UNIX", "tags": [{"Key": "Name", "Value": "instance3"}]},
 {"instance_id": "i-0ef56bd920dc76c00", "ami_id": "ami-02396cdd13e9a1257", "date_launched": "30/04/2023, 13:31:16", "availability_zone": "us-east-1a", "private_ip_address": "172.31.24.242", "public_ip_address": "3.90.183.104", "platform": "Linux/UNIX", "tags": [{"Key": "Name", "Value": "instance-demo"}]}]

custom-amis-data.json

[
	{"image_id": "ami-0cb3484c1a6f84067", "image_name": "CustomAMIRecipe 2023-03-04T11-20-20.189Z", "creation_date": "04/03/2023, 11:35:22"},
	{"image_id": "ami-0f7502a3b402fa336", "image_name": "CustomAMIRecipe4", "creation_date": "30/04/2023, 13:33:16"},
	{"image_id": "ami-0e6b71a8147764d2c", "image_name": "CustomAMIRecipe3", "creation_date": "27/03/2023, 19:44:26"},
	{"image_id": "ami-026356055b38dcfc7", "image_name": "CustomAMIRecipe2", "creation_date": "27/03/2023, 19:18:11"}]

APPENDIX D: Infrastructure diagram


AWS
Frontend
S3 bucket
Static web files
User's browser
Backend
API Gateway
Processing web request
AWS Lambda
Update and get data request
AWS Lambda
Store instances and images data
S3 bucket
S3 bucket