



Institute of Informatics and Mechatronics

Bachelor's Thesis

in the field of study of Computer Engineering and Mechatronics

Cloud Managed Unmanned Aerial System

Bruno Axel Kamere

Student's transcript number **030756**

Seminar conducted by

Szychta Elżbieta, prof. dr hab. inż.

Thesis supervisor

Ocetkiewicz Tomasz, mgr inż.

Bydgoszcz, Poland 2022

This work is dedicated to my loving and very supportive parents . . .

Acknowledgements

Looking back at my journey that led to this point, I would like to take some time and thank everyone that has helped me throughout my studies, and design, development, and testing of this final engineering work. The moral support received throughout this period is unprecedented.

I would like to first and foremost thank my lovely parents. They have been there for me in each single step that led to where I am and what I have achieved today. They have encouraged me, believed in me, and supported me in various ways. Thank you.

Next, I would like to also thank a lot my thesis supervisor, Ocetkiewicz Tomasz, mgr inż.; first, for believing in my ability to conduct my research towards the execution of my project and second, for the unprecedented advice and support I have received throughout my studies and throughout my final engineering research and project execution. Thank you.

I want to also take a moment to thank the WSG university's institute of informatics and mechatronics professors and staff for the knowledge and skills they have provided me throughout my studies. They have inspired me to like more my field of studies, due to the challenges provided to me along the way. Thank you.

Finally, I would not finish without thanking my classmates, colleagues and friends who have been part of my support system and have given me ideas and support during throughout my studies, until this day. A big thanks to you.

Cloud Managed Unmanned Aerial System

Abstract. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Keywords: AWS, UAV, UAS

DECLARATION

I, the undersigned,
student of the University of Economy in Bydgoszcz, declare that the following
Bachelor's/Engineering/ Master's/ Thesis* entitled
.....
.....
has been written on my own. It means that:

- 1) during writing the thesis I have not used any help from other people (apart from necessary the consultations conducted within the didactic process), mainly I have not commissioned writing the thesis or it's part to other people and I have not rewritten it or its part from other people;
- 2) After the use of the literature and source materials I have mentioned the author, the title and the source according to the quotation rules.
- 3) The thesis and its fragments has not been used as the basis of other procedures aiming at conferring degrees and academic and professional titles;

I declare that I am aware that if my thesis infringes rights connected with copyright and diplomatic regulations, it may be invalidated by the University even after the degree examination.

I also declare that the media attached and the version in the Electronic System ISAPS contains all files connected with the following thesis, which have been used in a printed version.

I prove the authenticity of the above declaration with my handwritten signature

Bydgoszcz, y.
(signature)

DECLARATION OF AUTHOR'S WILL

Surname and name of the author.....
The title of the thesis
.....
Address of Residence

I agree to make above mentioned thesis of my authorship available for scientific and didactic purposes with the consent of the Supervisor. The form of sharing includes: sharing of a printed copy and sharing of its electronic version.

Bydgoszcz, y.
(signature)

*delete as appropriate

Contents

| | |
|---|-------------|
| List of Figures | ix |
| List of Tables | xi |
| List of Source Codes | xiii |
| Nomenclature | xv |
| 1 Introduction | 1 |
| 1.1 Related work | 2 |
| 1.2 Problem definition | 3 |
| 1.3 Use case | 3 |
| 1.4 About HelloSky group | 5 |
| 2 Theory | 7 |
| 2.1 Unmanned Aerial System | 7 |
| 2.1.1 Unmanned Aerial Vehicle | 8 |
| 2.2 Amazon Web Services | 8 |
| 2.2.1 Infrastructure as code | 9 |
| 2.3 Simulation | 12 |
| 2.4 Tools used | 12 |
| 2.4.1 Microsoft Visual studio code | 13 |
| 2.4.2 PyCharm by JetBrains | 13 |
| 2.4.3 PhpStorm by JetBrains | 13 |
| 2.4.4 Affinity Designer | 14 |
| 2.4.5 GitHub | 15 |
| 2.4.6 Microsoft Visio | 15 |
| 3 Methodology and setup | 17 |
| 3.1 Approach | 17 |
| 3.2 Solution description | 19 |
| 3.2.1 AWS cloud development kit setup | 19 |
| 3.3 AWS Network access and security | 26 |
| 3.3.1 Public subnet | 26 |

| | | |
|----------|--|-----------|
| 3.3.2 | Private subnet | 27 |
| 3.3.3 | Isolated-private subnet | 29 |
| 3.4 | Software in the loop UAV simulator | 29 |
| 3.5 | UAV Communication | 29 |
| 4 | Discussion | 31 |
| 5 | Conclusion | 33 |
| 5.1 | Future Work | 33 |
| 5.1.1 | Low latency communication | 33 |
| 5.1.2 | Collision avoidance navigation | 33 |

List of Figures

| | |
|--|----|
| 1.0.1 Proposed system high-high-level design. | 2 |
| 1.4.1 HelloSky group logos. | 5 |
| 2.2.1 AWS console home. | 9 |
| 2.4.1 PyCharm editor. | 13 |
| 2.4.2 PhpStorm editor. | 14 |
| 2.4.3 Affinity Designer editor. | 14 |
| 2.4.4 Github repository for the AWS infrastructure code. | 15 |
| 2.4.5 Microsoft Visio. | 16 |
| 3.1.1 Initial draft architecture designs. | 18 |
| 3.2.1 AWS CDK app structure. | 20 |
| 3.2.2 AWS configure CLI output. | 20 |
| 3.2.3 AWS STS and configure commands output. | 21 |
| 3.2.4 Proposed solution AWS stacks. | 22 |
| 3.2.5 AWS CDK app lifecycle. | 23 |

List of Tables

| | |
|--|----|
| 3.3.1 Public subnet NACL inbound traffic rules | 26 |
| 3.3.2 Public subnet NACL outbound traffic rules | 27 |
| 3.3.3 Private subnet NACL inbound traffic rules | 28 |
| 3.3.4 Private subnet NACL outbound traffic rules | 28 |

List of Source Codes

| | | |
|---|--|----|
| 1 | helloskygroup.com AWS CDK Python Route 53 snippet. | 12 |
| 2 | AWS CDK app.py snippet. | 26 |

Nomenclature

Acronyms / Abbreviations

| | |
|-------|-------------------------------------|
| ALB | Application Load Balancer |
| AWS | Amazon Web Services |
| AZ | Availability Zone |
| CDK | Cloud Development Kit |
| CLI | Command Line Interface |
| DJI | Da-Jiang Innovations |
| EC2 | Elastic Cloud Compute |
| ECR | Elastic Container Registry |
| ECS | Elastic Container Service |
| GCS | Ground Control Station |
| GPS | Global Positioning System |
| HHL D | High-High-Level Design |
| HLD | High-Level Design |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secured |
| IaaS | Infrastructure as a Service |
| IaC | Infrastructure as Code |
| IAM | Identity Access Management |
| IANA | Internet Assigned Number Authority |

IETF Internet Engineering Task Force

LTE Long Term Evoluton (Telecommunication)

ML Machine Learning

NACL Network Access Control List

NAT Network Address Translation

PaaS Platform as a Service

PCB Printed Circuit Board

RDS Relational Database Service

RFC Request for Comments

S3 (as in AWS S3) Simple Storage Service

SD (as in SD card) Secure Digital

TCP Transmisison Control Protocol

UAS Unmanned Aerial System

UAV Unmanned Aerial Vehicle

VPC Virtual Private Cloud

Chapter 1

Introduction

Unmanned aerial vehicles (UAVs) also known as remotely piloted vehicles (RPVs) or Drones, although hardly a new technology, with the first used UAV recorded in history dating back to 1849 [1], have recently gained a lot of attention from various sectors ranging from entertainment to military. This is going to have an impact that cannot be overseen over the coming years as more and more people find uses of UAVs in various applications. UAVs were initially developed to be used for military operations, mainly surveillance, but they were later armed to also enable them to perform long-distance military operations without putting humans at risk. The United States of America has used these types of UAVs mainly in the wars in the Middle East, where UAVs like the General Atomics MQ-9 Reaper also known as Predator B and Northrop Grumman RQ-4 Global Hawk have been widely deployed [2].

Despite their use in the military sector, UAVs have also been employed in other sectors such as commercial and entertainment sectors, where UAVs are being used in things like land geography mapping, industrial surveillance, photography and many more. Companies like SZ DJI Technology Co., Ltd. or Shenzhen DJI Sciences and Technologies Ltd. in full, more popularly known as its trade name DJI have had a lot of success in this area, where as of March 2021 DJI was covering itself covers <RESEARCH ON THE MARKET PERCENTAGE OF DRONES THAT DJI MAKES>. UAVs have also seen great use in the healthcare sector, where companies like Zipline [3] are implementing an end-to-end supply chain system that employs UAVs to supply and deliver medical supplies to hospitals in rural areas in Rwanda that are hard to reach or inefficient to reach by other means of delivery.

Rwanda has also seen great use of UAVs during the COVID-19 pandemic where UAVs were widely used by the Rwanda's Ministry of Health and the Rwanda National Police to spread COVID-19 awareness in Kigali communities [4].

As UAVs gain the market, the need to have robust unmanned aerial systems (UASs) becomes inevitable. Currently there are not a lot of UASs that are deployed on the cloud. Therefore, in this thesis, focus was put in designing and building a robust, scalable, highly available unmanned aerial system deployed on the cloud service platform, Amazon

1. Introduction

web services (AWS), to provide a solution where UAV pilots can control UAVs from virtually anywhere in the world from web applications deployed on AWS. The proposed solution comprises of a UAV simulated on AWS with a datalink to a ground control system (GCS), telemetry dashboards and a command-and-control center application running in a highly available and fault tolerant AWS cloud infrastructure. The focus of this thesis is to therefore assess the possibilities of implementing such a solution in an efficient, resilient, reliable, and highly available manner and discuss on the pros and cons of the solution.

The proposed solution, as seen in the high high level design in figure 1.0.1, was developed following the best industry standards in software development and architecture as is going to be described in detail in the next chapters. This thesis is also going to discuss the developments that have already been made in this area as well as areas that need further research and development.

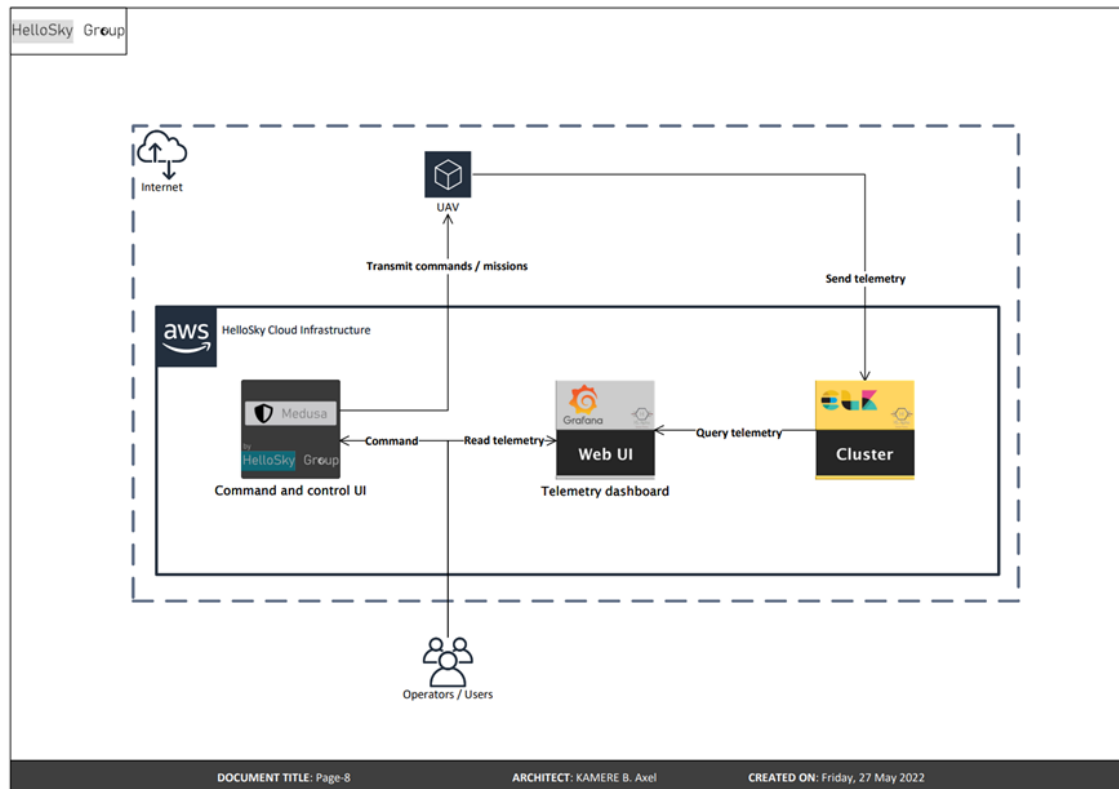


Figure 1.0.1. Proposed system high-high-level design.

Source: Own work. Designed with Microsoft Visio. Refer to 2.4.6.

1.1 Related work

UAVs and UASs in general is a field that has undergone substantial development through various researches done by scientists, engineers and academicians.

One of the challenges still faced by UAVs, especially in the capability of being able to deploy them in urban areas, is the safety of their operations. Being able to build a UAV with highly effective collision avoidance algorithms is still a field under active research. And this is one of the main challenges that need to be solved for the world to see robust autonomous UAVs employed widely in communities for various use cases. Pedro et. al have studied on how UAVs can be made more resilient and safe with the help of artificial intelligence, machine learning and the likes. In their article "Framework for fully autonomous UAVs" [5], they reviewed the current collision avoidance algorithms for both static and dynamic objects and proposed a conceptual framework to improve more the safety and reliability of UAVs.

<ADD ONE MORE USECASE, CLOUD RELATED??>

1.2 Problem definition

The cloud technology is an evolving area nowadays due to how agile, efficient, scalable and cost effective it is to deploy resources on the cloud. Many companies all over the world have seen multiple success stories in using cloud services where, for example, General Electric Renewable Energy has managed to achieve a 99.9% data availability through its move to the cloud[6].

The use of cloud services is yet to expand even more to other industries like the aerospace industry, especially in the management of unmanned aerial systems (UAS). As the world sees great use of unmanned aerial vehicles (UAVs), more efficient, reliable, scalable and highly available ways of deploying and operating components of the UAS will need to be developed. Building UASs where the UAV compute power can be on the cloud, as well as other components like the command and control centre would play a big part in advancing the unmanned aerial mobility sector. This can help in increasing the range in which UAVs that rely on battery power operate for example, because the UAV would not need to carry heavy compute power to process its generated data like imagery, as everything would be sent to the cloud to be processed. Therefore, the aim of this thesis is to expand on the question below:

- How can one take advantage of cloud computing to deploy an unmanned aerial system in a more efficient, scalable and highly available manner?
- What advantage is therein running applications on the cloud?

1.3 Use case

As UAVs emerge, there will be a need to be able to centrally manage a fleet of UAVs. Depending on the UAV use case, operators might need to also control them at a long

distance beyond eyesight. A UAV operates as part of a system comprised of multiple other components that support the operation of a UAV. The main components are a Ground Control System, <RESEARCH ON THE MAIN COMPONENT OF A UAV>. UAVs can either be fully autonomous, fully manual, or semi-autonomous. UAVs can also be employed in various use cases, below are various scenarios in which UAVs can be used <CITE SOURCES OF THE BELOW USE CASES>

- Agriculture.
- Facility inspection.
- Terrain mapping.
- Shipping and delivery.
- Search and rescue.
- Law enforcement.
- Military reconnaissance / Surveillance.

For a UAV to perform any of the above, it needs to meet certain criteria, a UAV should:

- Have onboard computer to process mission commands on the fly.
- Have onboard key components like,
 - Sensors, depending on the mission.
 - Cameras.
 - Battery.
 - LTE modules or Satnav modules to allow communication with ground control.
- Have LTE or Satellite communication to enable the UAV to set up a datalink with the ground control. The UAV would have to send data such as
 - Ground speed.
 - Altitude.
 - Battery levels.
 - Yaw.
 - Location.
 - Direction.
 - Sensor data.
 - Send the data frequently for real-time or near real-time communication.
 - Be able to react and if necessary, take evasive maneuvers when:

- * On collision course.
- * The batteries are low on power.
- * Out of connectivity range.

1.4 About HelloSky group

Across this thesis, there will be mentions of the name "HelloSky group". Several designs built for the project as well as source codes all have mentions of HelloSky group or hsg in abbreviations.

HelloSky group is a name that I came up with to label my work done and future developments that will be made on this project and many other related projects that will be built in the future. It is mainly done like so to improve motivation to keep working on the project, be it now or in the future. Figure 1.4.1 shows the HelloSky group logos that will be seen across various figures and source codes throughout the thesis project.



(a) Colored 500 x 200.



(b) Black and white 500 x 200.

Figure 1.4.1. HelloSky group logos.

Source: Own work. Designed with Affinity Designer. Refer to 2.4.4.

Chapter 2

Theory

In this chapter, key background concepts and methodologies used in the thesis are going to be discussed. The chapter is going to explain what is meant by unmanned aerial system and its components.

The chapter is also going to discuss on the cloud provider, Amazon web services (AWS), used to host various components of developed system, simulation and software development tools used, as well as laws and regulations around unmanned aerial systems.

2.1 Unmanned Aerial System

An unmanned aerial system commonly referred to as UAS is a set of an unmanned aerial vehicle and components that support its operations. They do not carry human pilots but are piloted remotely or autonomously. A UAS is usually comprised of four main components namely,

- An unmanned aerial vehicle also known as a UAV.
- A ground control station also known as a GCS, from where human pilots can remotely pilot a UAV or upload mission payload for the UAV to execute autonomously.
- Sensors and devices specific to the aerial vehicles' intended mission. These can be cameras or other various sensors.
- A datalink between a UAV and a GCS.

A typical UAS also includes some ways to collect telemetry from the UAV to some sort of datalake for further analysis. The collected data can be used for machine learning to improve the UAV operational efficiency.

<ADD AN IMAGE THAT SHOWS A TYPICAL UAS>

2.1.1 Unmanned Aerial Vehicle

Unmanned aerial vehicle is a term used to refer to an aerial vehicle that has no human onboard pilot but is rather piloted remotely by humans from a remote ground control station or is autonomously piloted using onboard flight algorithms. Unmanned aerial vehicles are commonly known as UAVs, drones or remotely piloted vehicles (RPV). <ADD MORE EXPLANATION>

Classification of Unmanned Aerial Vehicles

UAVs come in different classifications depending on the frame structure, size, operational range and/or payload capacity.

Table <REFERENCE TABLE> shows how the United States Department of Defense classifies UAVs.

2.2 Amazon Web Services

Amazon Web Services, commonly known as AWS, is a cloud platform provided by Amazon that provides various service offerings such as platform as a service, PaaS, and infrastructure as a service, IaaS[7]. AWS makes it easy for developers, engineers and businesses to deploy scalable, resilient, agile and highly available infrastructures for databases, servers, applications, storage, analytics, *et cetera*. AWS offers attractive and cost saving payment strategies of which there are pay-as-you-go, save when you commit, and pay less by using more[8].

AWS has a concept of regions, which refers to a physical location around the world, where multiple datacenters are deployed in a cluster. Each cluster of datacenter is called an availability zone[9]. AWS set this up like this to guarantee high availability and reliability of deployed resources.

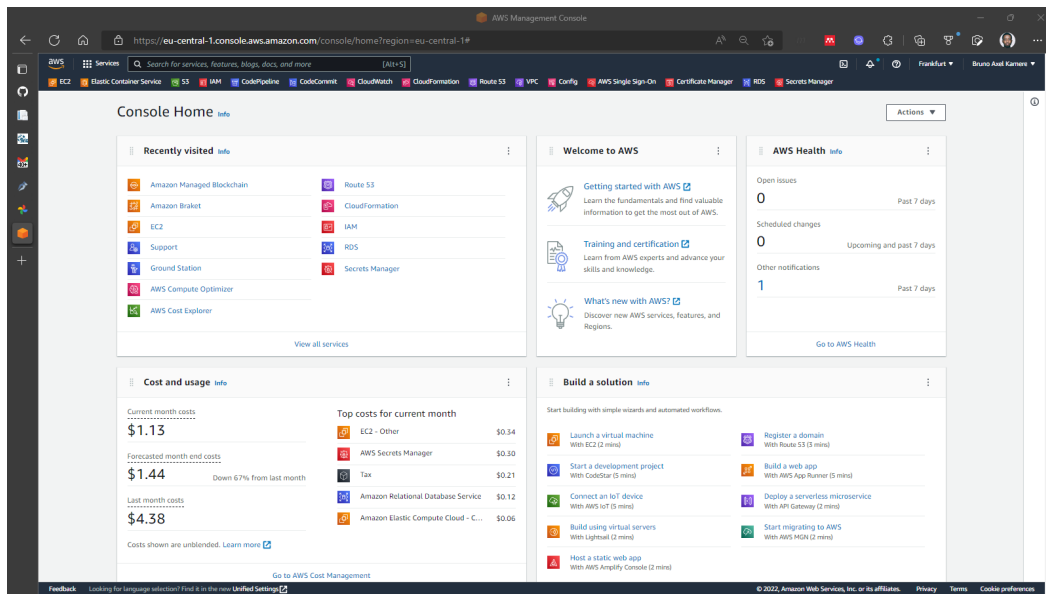


Figure 2.2.1. AWS console home.

Source: Own work.

Cloud computing is an emerging technology that has revolutionized how businesses go online. Cloud computing has been and still is of great use in various industries, including the aerospace and energy industries. Burak et al developed a cloud and edge solution running on AWS that aimed at increasing turbine maintenance inspections' efficiency through automation and a serverless AWS architecture while reducing operations cost[10]. The solution proposed by Burak et al was comprised of drones, machine learning and internet of things processes running on cloud and edge.

The proposed solution in this thesis also takes advantage of what AWS and cloud computing offers. Several components, like the ground control system, of the proposed solution are running on AWS. See the high-high-level design in figure 1.0.1.

2.2.1 Infrastructure as code

Infrastructure as Code also known as IaC, a technique very often used in the DevOps and automation community, is an infrastructure that is provisioned through code and scripts written in familiar programming languages like Python, PHP, Node.JS, C# *et cetera*. The infrastructure deployed through code can be servers, databases, firewalls, data centers *et cetera*. The main advantages of defining an infrastructure as code are:

- Improved efficiency and consistency.
- Reduced human error.
- Infrastructure agility. An infrastructure defined as code can be deployed as many times as needed, which reduces the effort invested by developers in case a replica of

an environment is needed elsewhere.

- It allows developers to take advantage of programming languages features like loops, variables *et cetera* to build more agile infrastructures.
- The infrastructure can be versioned and tightly controlled. Since the infrastructure is basically standard code, it can be versioned with various versioning tools like Git or Subversion. This facilitates maintenance and makes the infrastructure easy to be rolled back, in case of issues.
- It helps with cost savings. Since the whole infrastructure is basically deployed automatically through code, engineers can then shift their focus to work on other important tasks.

In this thesis, Infrastructure as Code is used to its outmost potential. The AWS infrastructure is deployed as code using the AWS proprietary software development framework called AWS Cloud Development Kit or AWS CDK. AWS CDK is an open source kit provided by AWS that allows engineers to define IT infrastructures on AWS using familiar programming languages. In the source code 1 is an example snippet from the AWS CDK app developed for the proposed solution in this thesis. The snippet represents a part that adds DNS records to the AWS Route 53 service using standard Python code.

```
1  #!/usr/bin/env
2
3
4  # Import needed libraries
5  import aws_cdk as cdk
6  from aws_cdk import (
7      aws_route53 as route53,
8      aws_certificatemanager as certificate_manager,
9      aws_route53_targets as targets,
10     Stack
11 )
12 from constructs import Construct
13
14
15 class Route53RecordsStack(Stack):
16     def __init__(self, scope: Construct, construct_id: str, props: dict,
17         ↪ internet_facing_alb, hosted_zone,
18         ↪ **kwargs) -> None:
19         super().__init__(scope, construct_id, **kwargs)
20
21         # TODO: #61 Apply removal policy of the hosted zone.
22
23         # Create A record 'helloskygroup.com' pointing to the internet facing
24         ↪ ALB alias.
```

```

23     route53.ARecord(self,
24         f"{props['company_abbreviation']}-medusa-{props['environ_']
25         ↪ ment']}-alias-a-record",
26         target=route53.RecordTarget(
27             alias_target=targets.LoadBalancerTarget(internet_fac_
28             ↪ ing_alb)),
29             zone=hosted_zone,
30             comment="A record for root helloskygroup.com pointing to
31             ↪ the internet facing ALB",
32             ttl=cdk.Duration.hours(2)
33         )
34
35     # Create A record 'www.helloskygroup.com' pointing to the internet
36     ↪ facing ALB alias.
37     route53.ARecord(self,
38         f"{props['company_abbreviation']}-medusa-www-{props['env_']
39         ↪ ironment']}-alias-a-record",
40         target=route53.RecordTarget(
41             alias_target=targets.LoadBalancerTarget(internet_fac_
42             ↪ ing_alb)),
43             zone=hosted_zone,
44             record_name="www",
45             comment="A record for www pointing to the internet
46             ↪ facing ALB",
47             ttl=cdk.Duration.hours(2)
48         )
49
50     # Create A record 'dashboard.helloskygroup.com' pointing to the internet
51     ↪ facing ALB alias.
52     route53.ARecord(self,
53         f"{props['company_abbreviation']}-grafana-{props['enviro_']
54         ↪ nment']}-alias-a-record",
55         target=route53.RecordTarget(
56             alias_target=targets.LoadBalancerTarget(internet_fac_
57             ↪ ing_alb)),
58             zone=hosted_zone,
59             record_name="dashboard",
60             comment="A record for Grafana
61             ↪ (dashboard.helloskygroup.com) pointing to the
62             ↪ internet "
63             ↪ "facing ALB",
64             ttl=cdk.Duration.hours(2)
65         )
66
67     # Create A record 'logs.helloskygroup.com' pointing to the internet
68     ↪ facing ALB alias.
69     route53.ARecord(self,

```

```
57         f"{props['company_abbreviation']}-kibana-{props['environ_}
58         ↪ ment']}-alias-a-record",
59         target=route53.RecordTarget(
60             alias_target=targets.LoadBalancerTarget(internet_fac_
61             ↪ ing_alb)),
62             zone=hosted_zone,
63             record_name="logs",
64             comment="A record for Kibana (logs.helloskygroup.com)
65             ↪ pointing to the internet facing ALB",
66             ttl=cdk.Duration.hours(2)
67         )
68
69     # Create A record 'api.helloskygroup.com' pointing to the internet
70     ↪ facing ALB alias.
71     route53.ARecord(self,
72         f"{props['company_abbreviation']}-node-red-{props['envir_}
73         ↪ onment']}-alias-a-record",
74         target=route53.RecordTarget(
75             alias_target=targets.LoadBalancerTarget(internet_fac_
76             ↪ ing_alb)),
77             zone=hosted_zone,
78             record_name="api",
79             comment="A record for Node-Red (api.helloskygroup.com)
80             ↪ pointing to the internet facing ALB",
81             ttl=cdk.Duration.hours(2)
82         )
```

Listing 1. helloskygroup.com AWS CDK Python Route 53 snippet.

2.3 Simulation

The initial project plan was to develop a whole UAS from scratch with an actual physical UAV and components, but this was deemed to be time consuming, and expensive to develop. Therefore, all the components that make a UAV were simulated using a simulation technique called Software in the loop or SITL. This technique allows developers to develop UAV flight logics using software and no hardware involved, eventhough that is possible as well. Chapter 3 elaborates more on how SITL was used in this project to simulate an actual UAV with telemetry.

2.4 Tools used

The solution proposed in this thesis was built using various software development, and design tools. The choice of tools is really key to an organised and well managed project

development, therefore it was important to choose the right tools for the right tasks to help get the expected outcome from them.

In the next subsections, various tools during the solution development are going to be listed and discussed.

2.4.1 Microsoft Visual studio code

Visual studio code or VS code is a source code editor provided by Microsoft. It was used in this thesis as a code editor to write various codes and scripts. <ADD SCREENSHOT>

2.4.2 PyCharm by JetBrains

The AWS infrastructure was built as code using Python programming language. Due to its robustness and great Python support, Pycharm by JetBrains integrated development devolpment (IDE) was the go to choice.

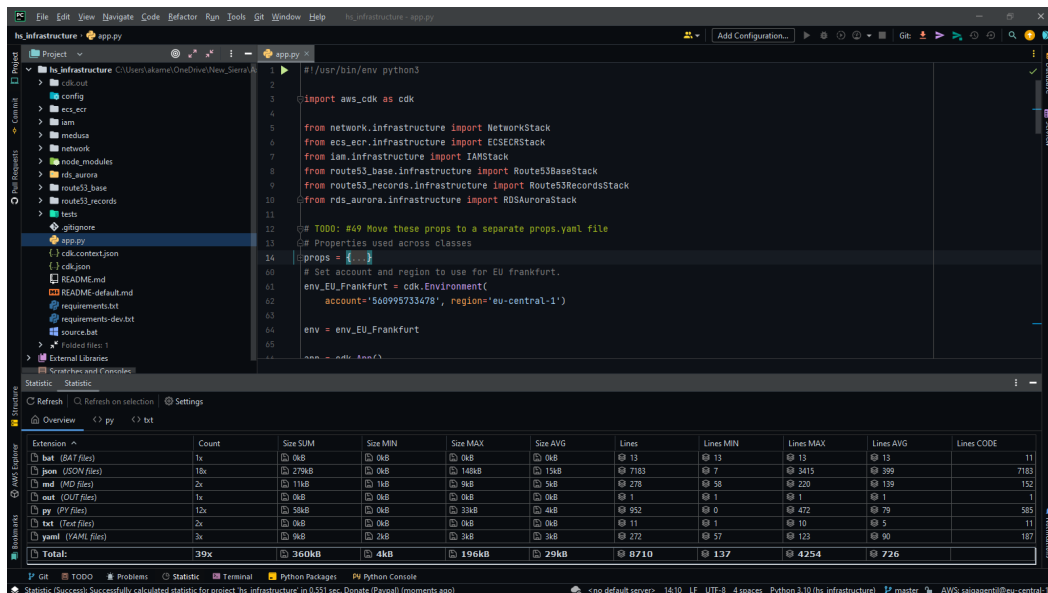


Figure 2.4.1. PyCharm editor.

Source: Own work.

2.4.3 PhpStorm by JetBrains

PhpStorm is an integrated development environment created by JetBrains specifically for php programming language. The web interface from which the UAV is controlled from is built with php's Laravel framework, and PhpStorm is perfectly optimised for development of php web applications.

2. Theory

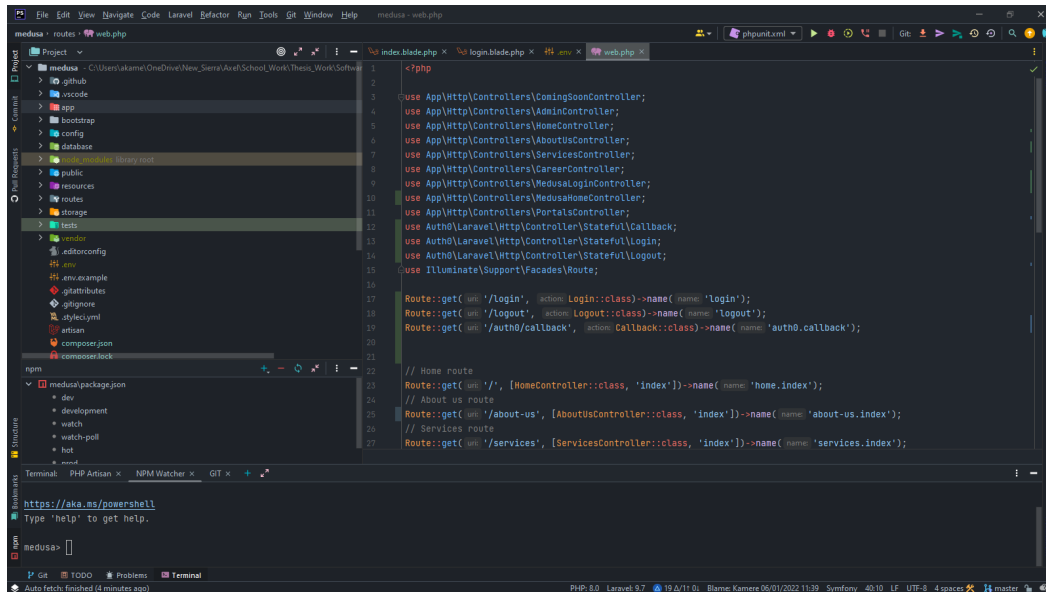


Figure 2.4.2. PhpStorm editor.

Source: Own work.

2.4.4 Affinity Designer

Several user interface components used across the project like icons and logos, like in figure 1.4.1 for example, were designed using Affinity Designer. Affinity Designer is a graphics tool used to design and create logos, icons, concept arts, UI designs *et cetera*.

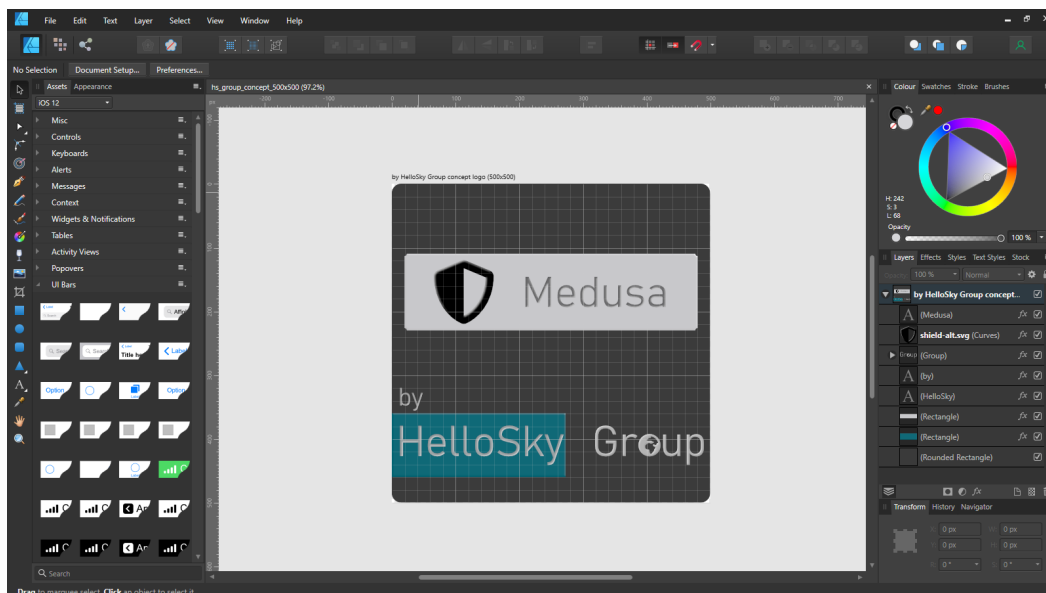


Figure 2.4.3. Affinity Designer editor.

Source: Own work.

2.4.5 GitHub

One of the fundamentals of software development and coding projects generally is to version the code so that changes can be tracked overtime. Making sure that a project is versioned and maintained centrally in a repository is very important, especially where teams are working together on a similar project. Git, one of the softwares used for code versioning, was used in this project to track changes across various components of the overall project. In fact this thesis document itself is versioned using Git <ADD LINK TO THIS THESIS ON GITHUB>, alongside other components of the proposed solution. Github then comes into play to act as the single point of truth where multiple Git repositories can be pushed and managed from. <ADD IMAGE SHOWING A TYPICAL BASIC GIT - GITHUB FLOW>

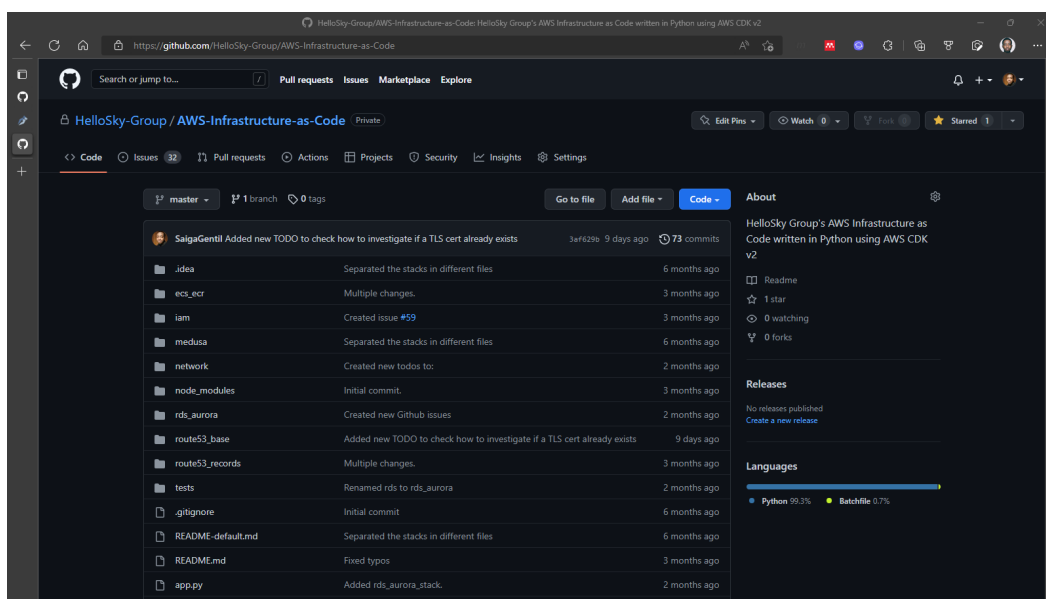


Figure 2.4.4. Github repository for the AWS infrastructure code.

Source: Own work.

2.4.6 Microsoft Visio

Microsoft Visio is an application part of the Microsoft office family that is used for digramming and graphics visualization. It is used to build architecture diagrams and many more. In this project, Microsoft Visio was used to design and create architecture design diagrams of the proposed solution.

2. Theory

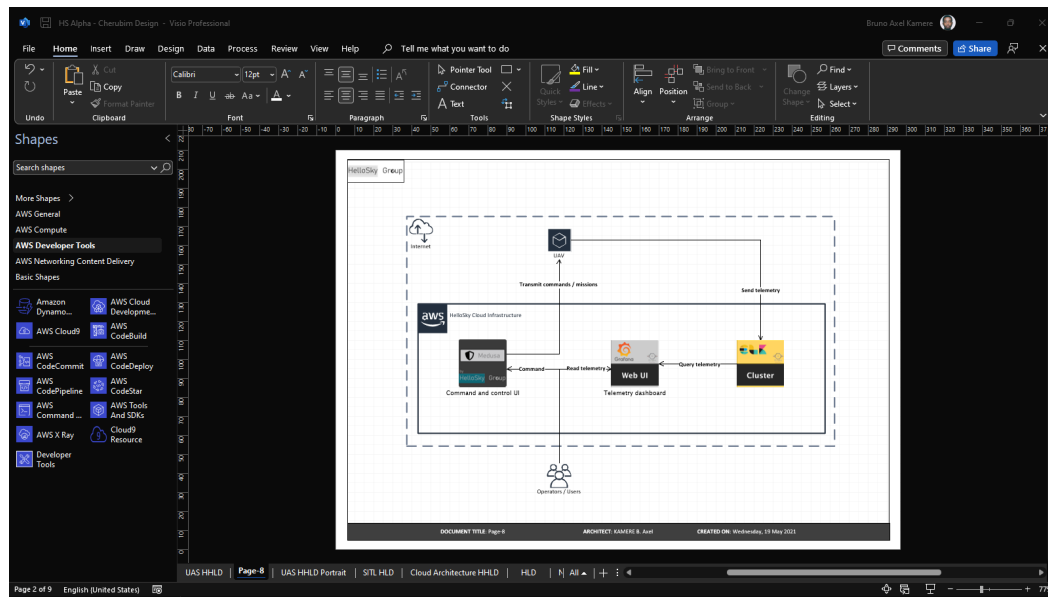


Figure 2.4.5. Microsoft Visio.

Source: Own work.

Chapter 3

Methodology and setup

In this chapter, the solution is going to be explained in details. The reason behind various design choices is going to be explained elaborately as well as the technical aspects of the solution.

3.1 Approach

In the early stages of the development, the main idea was that the solution had to be:

- well planned. Every step taken had to be thought of, usually drawn on a paper to assess its feasibility. Figure 3.1.1 shows the initial draft designs.
- well built. The project had to be well structured such that it is intuitive to navigate around source codes. A naming convention was followed throughout the source codes.
- easily maintainable. The project was broken down into segments of folders, and the code was built with inheritance.
- built with professional, software engineering industry standard toolset, as described in section 2.4.
- agile. The AWS infrastructure mainly had to be built such that it is easy to deploy and destroy without investing a lot of effort.

With the above points in mind, the solution design process started on paper. Figure 3.1.1 shows some of the initial designs of the proposed system. After several design iterations, the final design was produced as shown in the high level design on figure <REFERENCE THE HLD>. The next step was to bring the design to reality. And that was no easy task.

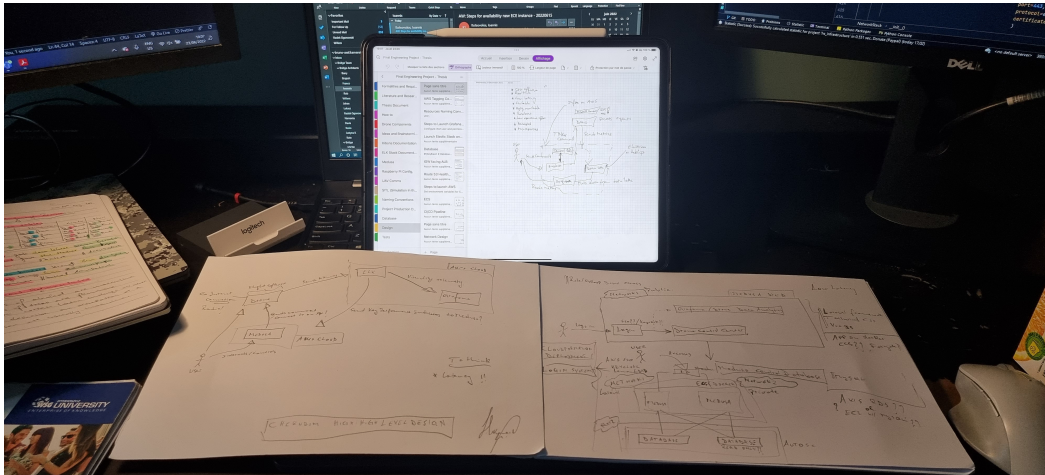


Figure 3.1.1. Initial draft architecture designs.

Source: Own work.

As the design started being built to reality, it was evident that deploying the whole AWS infrastructure manually from the AWS console, see figure <REFERENCE THE AWS CONSOLE>, was going to be a time consuming, inefficient activity. And this also contradicted the initial idea of building an agile infrastructure that can easily be redeployed in case changes are needed. That is where the thought to build the infrastructure as code using some sort of infrastructure as code (IaC) tool came in. Since the cloud provider of choice was AWS, the best tool for the task was with no doubt the AWS cloud development kit <CITE AWS CDK>.

After the AWS infrastructure was built, the next step was the UAV. The outstanding questions here were:

- How is the UAV going to be built?
- What is going to be the size?
- What is going to be the UAV type? Quadcopter? Fixed wing?
- How is the UAV going to be programmed? What programming language should be used?
- How is the UAV going to communicate with the rest of the system on AWS?

The above questions drove the next development stage. The drone type and size of choice were a small quadcopter. The initial idea was to build an actual quadcopter, therefore the below parts were purchased to build the quadcopter:

- Readytosky S500 quadcopter frame with built-in PCB.
- Pixhawk 2.4.8 flight controller with 4GB of onboard SD card storage.
- Raspberry Pi 4 model B with 4GB of onboard SD card storage.

- Readytosky M8N GPS module built-in compass with GPS antenna mount.
- 4 pieces of A2212 1000KV brushless motors.
- 4 pieces of 2-6S 30 amps Electronic speed controllers.
- 2 pairs of 1238 carbon fiber propellers.

<INSERT AN IMAGE OF THE ABOVE PARTS>

As the development of the quadcopter went on, it became obvious that this approach was not the best way to go, especially for a proof-of-concept (POC) solution mainly due to how costly it was becoming. At somepoint the wire connectors were even insufficient, and it would take too long to order new ones online. The idea to build the actual physical quadcopter was then put on-hold, and it was decided to rather use simulation tools to simulate the actual quadcopter. Section 3.4 elaborates more on how the simulation was set up.

<REVIEW THIS SECTION>

3.2 Solution description

<ATTACH THE HLD AND DESCRIBE EVERY CONNECTION>

3.2.1 AWS cloud development kit setup

As mentioned in the previous chapters, the designed AWS infrastructure was deployed as code using AWS cloud development kit (AWS CDK). AWS CDK allows developers to build AWS infrastructures in an agile manner by using normal programming languages. This allows developers to take advantage of programming idioms like loops, variables, conditionals and inheritance to build robust AWS infrastructures. This also means that software engineering practices like source control, testing and code reviews can be used since the infrastructure is basically standard code. Currently AWS CDK supports Python, Javascript, C#, Go, and Typescript programming languages.

A typical AWS CDK project is made up of 3 components, namely:

- an app. This is the overall set of everything. An AWS CDK project is basically called an AWS CDK app. This comprises of constructs.
- constructs. These represent components that make an AWS infrastructure. They can be an AWS simple storage service (S3), an elastic cloud compute (EC2), an application load balancer (ALB) *et cetera*. AWS has a comprehensive site that shows all the available AWS and community constructs[11].
- stacks. These are the unit of deployment in AWS CDK. A stack represents all AWS resources defined within the same scope.

Figure 3.2.1 shows how an AWS CDK app is structured.

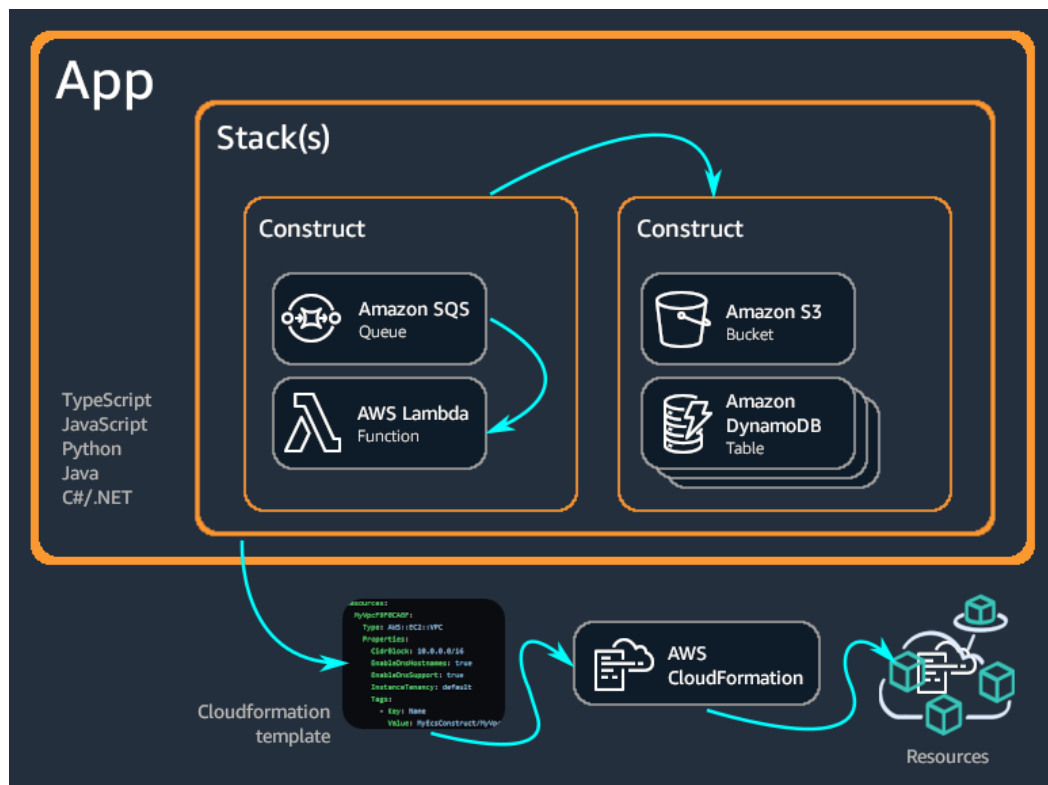


Figure 3.2.1. AWS CDK app structure.

Source: AWS CDK documentation[12].

Build

To start building the AWS infrastructure with AWS CDK, the AWS command line interface (CLI) needs to first be set up. To install the AWS CLI on Windows, run the following command `msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi`. This allows to interact with AWS via the CLI.

Once the AWS CLI is installed, type the command `aws configure` to configure credentials to AWS. The console will ask for the AWS access key ID, the AWS secret access key, the default region name and the default output format as shown in figure 3.2.2.

```
akame> aws configure
AWS Access Key ID [*****j375]:
AWS Secret Access Key [*****f/zk]:
Default region name [eu-central-1]:
Default output format [json]:
```

Figure 3.2.2. AWS configure CLI output.

Source: Own work.

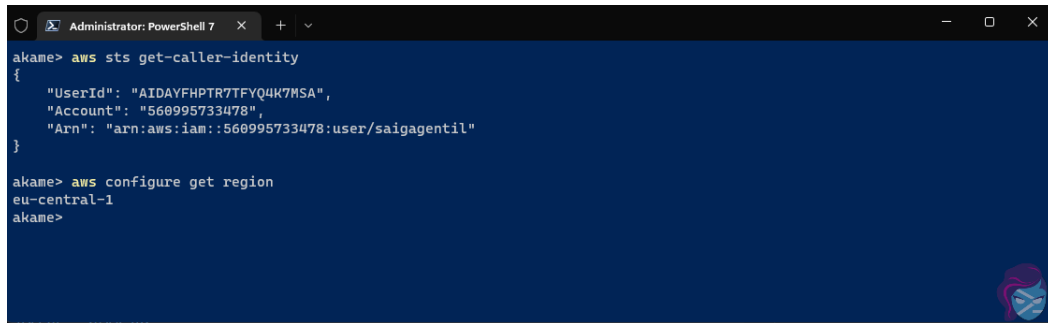
The next step is to then install AWS CDK from node package manager (NPM) with `npm`

`install -g aws-cdk`. This will download and install the latest aws-cdk version on the local workstation. The "-g" flag makes the package available globally on the workstation.

Next, get the account number from the AWS CLI using the AWS security token service (STS) and the region name from AWS CLI configuration by executing the commands below.

`aws sts get-caller-identity`, to get the account number.

`aws configure get region`, to get the region name.



```
Administrator: PowerShell 7
akame> aws sts get-caller-identity
{
  "UserId": "AIDAYFHPT7TFYQ4K7MSA",
  "Account": "560995733478",
  "Arn": "arn:aws:iam::560995733478:user/saigagentil"
}

akame> aws configure get region
eu-central-1
akame>
```

Figure 3.2.3. AWS STS and configure commands output.

Source: Own work.

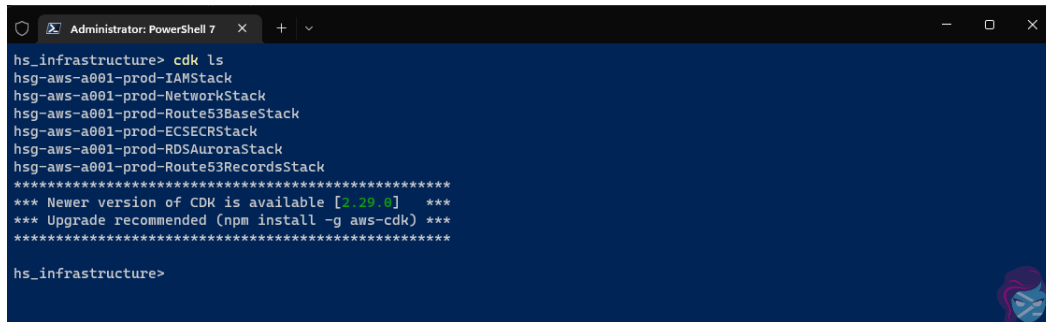
With the account number and the region name, AWS CDK can be bootstrapped now. Bootstrapping in AWS refers to the process of creating assets (local files, directories, docker image,...) that need to be deployed with the stack. These assets are deployed to an AWS S3 bucket for AWS CloudFormation to use them during stack deployment. Execute command `cdk bootstrap aws://560995733478/eu-central-1` to bootstrap AWS CDK.

Once all the above steps are completed, a CDK app can now be created. First create a directory for the app and move into the directory, `mkdir hs_infrastructure` then `cd hs_infrastructure`, then run the command `cdk init app --language python` to initialize a Python AWS CDK app. After that, activate the Python virtual environment with `source .venv/Scripts/activate` and install the core AWS CDK plugins with `python -m pip install -r requirements.txt`.

This will generate a couple of starting files in the app directory. Also if Git is installed on the workstation the app will be initialized as a Git repository that can be versioned and later be pushed to a Git remote repository like Github.

Deploy

After completing the previous steps, we have an app with a default stack but with no resources defined in it. The next step is to write code that defines stacks that make the AWS infrastructure. In the proposed solution, multiple stacks were created, figure 3.2.4 shows the stacks that make the proposed solution.



```
Administrator: PowerShell 7
hs_infrastructure> cdk ls
hsg-aws-a001-prod-IAMStack
hsg-aws-a001-prod-NetworkStack
hsg-aws-a001-prod-Route53BaseStack
hsg-aws-a001-prod-ECSECRStack
hsg-aws-a001-prod-RDSECRStack
hsg-aws-a001-prod-Route53RecordsStack
*****
*** Newer version of CDK is available [2.29.0] ***
*** Upgrade recommended (npm install -g aws-cdk) ***
*****
hs_infrastructure>
```

Figure 3.2.4. Proposed solution AWS stacks.

Source: Own work.

Each of the stacks defines several services needed to build the whole overall infrastructure.

- **hsg-aws-a001-prod-IAMStack:** Creates IAM user needed by the AWS elastic container service (ECS) and elastic container registry (ECR).
- **hsg-aws-a001-prod-NetworkStack:** Sets up the overall AWS infrastructure networking. Subnets, firewall rules, *et cetera*. This is the longest stack in terms of lines of code, because it is made up of approximately 480 lines of code.
- **hsg-aws-a001-prod-Route53BaseStack:** This creates a hosted zone for the domain `helloskygroup.com` as well as a wildcard SSL certificate for the domain.
- **hsg-aws-a001-prod-ECSECRStack:** This spins up docker containers hosting the web application in the AWS ECS Fargate service. Fargate was used since it is a serverless service. A serverless service is a service that offloads the responsibility from developers to managed the actual physical servers hosting the deployed resources.
- **hsg-aws-a001-prod-RDSECRStack:** This deploys a MySQL database in the AWS relational database service (RDS).
- **hsg-aws-a001-prod-Route53RecordsStack:** This adds DNS records in the hosted zone created in the 'hsg-aws-a001-prod-Route53BaseStack' stack.

Once each stack is developed, the next step is to deploy it to AWS. Snippet 2 shows the `main.py` code that basically imports all the created tasks in the AWS app and executes them. From there, the command `cdk deploy -all` needs to be executed from the root directory to deploy the resources and stacks defined in the app. Figure 3.2.5 shows the full lifecycle of an AWS CDK app.

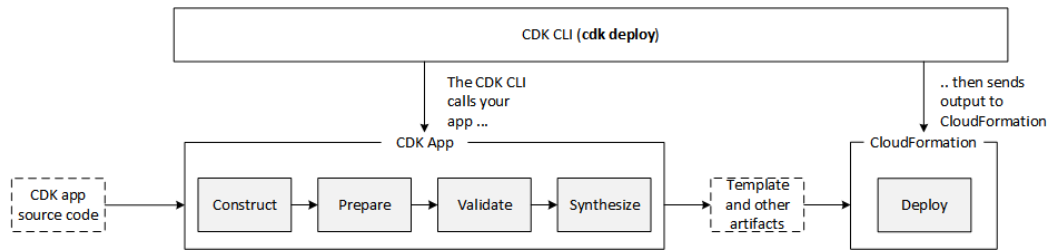


Figure 3.2.5. AWS CDK app lifecycle.

Source: AWS documentation[13].

```

1  #!/usr/bin/env python3
2
3  import aws_cdk as cdk
4
5  from network.infrastructure import NetworkStack
6  from ecs_ecr.infrastructure import ECSECRStack
7  from iam.infrastructure import IAMStack
8  from route53_base.infrastructure import Route53BaseStack
9  from route53_records.infrastructure import Route53RecordsStack
10 from rds_aurora.infrastructure import RDSAuroraStack
11
12 # TODO: #49 Move these props to a separate props.yaml file
13 # Properties used across classes
14 # <Ommited in the snippet due to how long it is>
15 props = {...}
16
17 # Set account and region to use for EU frankfurt.
18 env_EU_Frankfurt = cdk.Environment(
19     account='560995733478', region='eu-central-1')
20
21 env = env_EU_Frankfurt
22
23 app = cdk.App()
24
25 # Define and run the IAM stack.
26 iam_stack = IAMStack(app,
27     f"{props['company_abbreviation']}-{props['namespace']}-{pro}
28     ↪ ps['environment']}-IAMStack",
29     props,
30     description=f"This stack creates all the users, roles,
31     ↪ groups, policies and permissions for other "
32     f"{props['company_abbreviation']}-{props['names}
33     ↪ pace']}-{props['environment']} to use.",
34     env=env)
35
36 # Define and run the route 53 base stack. This stack defines the hosted zone and
37 ↪ ssl certificate for helloskygroup.com

```

3. Methodology and setup

```
34 route53_base_stack = Route53BaseStack(app,
35     f"{props['company_abbreviation']}-{props['namespace']}"
36     f"{props['environment']}-Route53BaseStack",
37     props,
38     description=f"This stack deploys the
39         {props['company_abbreviation']}-"
40         f"{props['namespace']}-{props['environment']} base DNS
41         infrastructure"
42         f". This stack creates the
43         hosted zone and the
44         SSL/TLS "
45         f"wildcard certificate for
46         'helloskygroup.com'.",
47     env=env
48 )
49
50 # Define and run the network stack.
51 network_stack = NetworkStack(app,
52     f"{props['company_abbreviation']}-{props['namespace']}"
53     f"-{props['environment']}"
54     f"NetworkStack",
55     props,
56     wildcard_cert=route53_base_stack.wildcard_cert,
57     description=f"This stack deploys the {props['company_abbreviation']}-{props['namespace']}"
58         f"{props['environment']} network and
59         all its components.",
60     env=env
61 )
62
63 # Define and run the RDS Aurora stack to create the MySQL database cluster with
64     RDS Aurora.
65 rds_aurora_stack = RDSAuroraStack(app,
66     f"{props['company_abbreviation']}-{props['namespace']}"
67     f"-{props['environment']}"
68     f"RDSAuroraStack",
69     props,
70     vpc=network_stack.vpc,
71     description=f"This stack deploys the
72         {props['company_abbreviation']}-"
73         f"{props['namespace']}-{props['environment']} RDS Aurora MySQL
74         database"
75         f"cluster. Multiple databases will
76         be created within the
77         cluster.",
```

```

65         env=env
66     )
67
68     # Define and run the Route 53 stack to create the DNS infrastructure.
69     route53_records_stack = Route53RecordsStack(app,
70                                                 f"{props['company_abbreviation']}-{p}
71                                                 ↳ rops['namespace']}"
72                                                 f"{props['environment']}-Route53Reco
73                                                 ↳ rdsStack",
74                                                 props,
75                                                 hosted_zone=route53_base_stack.hoste
76                                                 ↳ d_zone,
77                                                 internet_facing_alb=network_stack.in
78                                                 ↳ ternet_facing_alb,
79                                                 description=f"This stack deploys the
80                                                 ↳ {props['company_abbreviation']}"
81                                                 ↳ f"{props['namespace']}-{
82                                                 ↳ props['environment']}
83                                                 ↳ } DNS Records"
84                                                 ↳ f" infrastructure. This
85                                                 ↳ stack creates all
86                                                 ↳ the DNS records.",
87                                                 env=env
88     )
89
90     # Define and run the ECS ECR stack.
91     # This stack depends on the network stack.
92     ecr_ecs_stack = ECSECRStack(app,
93                                 f"{props['company_abbreviation']}-{props['namespace']
94                                 ↳ rops['environment']}"
95                                 ↳ f"{props['environment']}-ECSECRStack",
96                                 props,
97                                 vpc=network_stack.vpc,
98                                 internet_facing_alb_secg=network_stack.internet_faci
99                                 ↳ ng_alb_secg,
100                                grafana_tg=network_stack.grafana_tg,
101                                description=f"This stack deploys the {props['company']
102                                ↳ _abbreviation']}-{props['namespace']}"
103                                ↳ f"{props['environment']} ECS and ECR
104                                ↳ infrastructure.",
105                                env=env
106    )
107
108    # TODO: #68 Add dependencies between stacks.
109
110    # print(iam_stack.iam_props['ecs_ecr_user'])
111    # Synthesize a cloud assembly from 'app'. The cloud assemblies include
112    ↳ everything needed to deploy 'app' to the cloud.

```

3. Methodology and setup

```
98 # It includes AWS CloudFormation template for each stack and a copy of any file
    ↳ assets or Docker images that are
99 # referenced in the 'app'.
100 app.synth()
```

Listing 2. AWS CDK app.py snippet.

3.3 AWS Network access and security

One of the challenges with implementing a networked system, especially on cloud platforms like AWS, is ensuring that traffic flows in the expected way with proper security in place. The proposed solution, being a networked solution involving communications to and from various applications, has a rigorous network design. Figure <REFERENCE TO THE NETWORK DESIGN IMAGE> shows how network within the proposed AWS infrastructure was designed.

AWS has a concept of Virtual Private Cloud also known as VPC, which is simply an isolated private network that can be broken down into various subnets depending on the architecture. The proposed solution has one VPC broken down into three subnets; public, private and isolated-private subnets for each availability zone.

3.3.1 Public subnet

The public subnet in this proposed solution does not contain any resources, except a Network Address Translation or NAT gateway that is used by resources in the private subnet to access the internet. Table 3.3.1 and 3.3.2 show the inbound and outbound traffic rules respectively configured on the public subnet network access control list or NACL.

| Inbound traffic | | | | | |
|-----------------|-------------|----------|------------|-----------|------------|
| Rule | Type | Protocol | Port range | Source | Allow/Deny |
| 100 | HTTP (80) | TCP (6) | 80 | 0.0.0.0/0 | Allow |
| 110 | HTTPS (443) | TCP (6) | 443 | 0.0.0.0/0 | Allow |
| 120 | Custom TCP | TCP (6) | 1024-65535 | 0.0.0.0/0 | Allow |
| * | All IPV4 | All | All | 0.0.0.0/0 | Deny |

Table 3.3.1. Public subnet NACL inbound traffic rules

- **Rule 100:** Allows inbound HTTP traffic on port 80 towards any IPv4 address on the internet.
- **Rule 110:** Allows inbound HTTPS traffic on port 443 towards any IPv4 address on the internet.

- **Rule 120:** Allows returning TCP traffic from the internet responding to requests from the subnet. The specified port ranges are ephemeral ports as defined by the Internet Assigned Number Authority or IANA and Internet Engineering Task Force or IETF in their Request for Comments or RFC 6056 document [14].
- **Rule *:** Block every other non previously evaluated IPv4 traffic.

| Outbound traffic | | | | | |
|------------------|-------------|----------|------------|-------------|------------|
| Rule | Type | Protocol | Port range | Destination | Allow/Deny |
| 100 | HTTP (80) | TCP (6) | 80 | 0.0.0.0/0 | Allow |
| 110 | HTTPS (443) | TCP (6) | 443 | 0.0.0.0/0 | Allow |
| 120 | Custom TCP | TCP (6) | 1024-65535 | 0.0.0.0/0 | Allow |
| * | All IPV4 | All | All | 0.0.0.0/0 | Deny |

Table 3.3.2. Public subnet NACL outbound traffic rules

The rules explanation are similar to those for inbound traffic in table 3.3.1, except that instead of inbound it is outbound.

3.3.2 Private subnet

Most of the infrastructure components are deployed in the private subnet where only specific traffic from the public and isolated-private subnets are allowed in. In this subnet is where the UAV command and control center user interface is deployed, in containers using the AWS Fargate serverless service. The rules for this subnet have to be carefully defined so that;

- Fargate services can pull docker images from docker hub public repositories on the internet.
- The UAV, and several command and control application services can talk to each other.

Table 3.3.3 and 3.3.4 show the inbound and outbound traffic rules respectively configured on the private subnet network access control list or NACL.

3. Methodology and setup

| Inbound traffic | | | | | |
|-----------------|-------------|----------|------------|-------------|------------|
| Rule | Type | Protocol | Port range | Source | Allow/Deny |
| 100 | HTTP (80) | TCP (6) | 80 | 0.0.0.0/0 | Allow |
| 110 | HTTPS (443) | TCP (6) | 443 | 0.0.0.0/0 | Allow |
| 120 | Custom TCP | TCP (6) | 1024-65535 | 0.0.0.0/0 | Allow |
| 130 | Custom TCP | TCP (6) | 3306 | 10.0.4.0/28 | Allow |
| 140 | Custom TCP | TCP (6) | 3306 | 10.0.5.0/28 | Allow |
| * | All IPV4 | All | All | 0.0.0.0/0 | Deny |

Table 3.3.3. Private subnet NACL inbound traffic rules

- **Rule 100:** Allows inbound HTTP traffic on port 80. This is so that the AWS Elastic Container Service or ECS tasks can pull images from the public Dockerhub registry.
- **Rule 110:** Allows inbound HTTPS traffic on port 443.
- **Rule 120:** Allows returning TCP traffic from the internet responding to requests from the subnet.
- **Rule 130 and Rule 140:** Allows inbound traffic on port 3306 from MySQL database running in the AWS Relational Database Service or AWS within the isolated-private subnets of both the Availability Zones.
- **Rule *:** Blocks every other non previously evaluated IPv4 traffic.

| Outbound traffic | | | | | |
|------------------|-------------|----------|------------|-------------|------------|
| Rule | Type | Protocol | Port range | Destination | Allow/Deny |
| 100 | HTTP (80) | TCP (6) | 80 | 0.0.0.0/0 | Allow |
| 110 | HTTPS (443) | TCP (6) | 443 | 0.0.0.0/0 | Allow |
| 120 | Custom TCP | TCP (6) | 1024-65535 | 0.0.0.0/0 | Allow |
| 130 | Custom TCP | TCP (6) | 3306 | 10.0.4.0/28 | Allow |
| 140 | Custom TCP | TCP (6) | 3306 | 10.0.5.0/28 | Allow |
| * | All IPV4 | All | All | 0.0.0.0/0 | Deny |

Table 3.3.4. Private subnet NACL outbound traffic rules

- **Rule 100:** Allows outbound HTTP traffic on port 80 towards any IPv4 address.
- **Rule 110:** Allows outbound HTTPS traffic on port 443 towards any IPv4 address.
- **Rule 120:** Allows all outbound response TCP traffic.
- **Rule *:** Blocks every other non previously evaluated IPv4 traffic.

<TALK ABOUT SECURITY GROUPS>

3.3.3 Isolated-private subnet

The isolated-private subnet hosts the MySQL database running in AWS Relational Database Service. This subnet only talks to the private subnet, and has no direct connection to the internet. This improves the infrastructure security through not exposing the database directly to the internet.

<ADD NETWORK FLOW DESIGNS>

Describe the solution on a higher level. Discuss HLDs.

3.4 Software in the loop UAV simulator

3.5 UAV Communication

<Talk about Mavlink... and how mavlink is used in the project>

Chapter 4

Discussion

- Challenges with setting up network flows (Talk about vpc flow log)

Chapter 5

Conclusion

5.1 Future Work

5.1.1 Low latency communication

5.1.2 Collision avoidance navigation

Bibliography

- [1] V. PRISACARIU, “The history and the evolution of uavs from the beginning till the 70s – doaj”, *JOURNAL OF DEFENSE RESOURCES MANAGEMENT*, vol. 8, pp. 181–189, 1 Apr. 2017, ISSN: 2068-9403. [Online]. Available: <https://doaj.org/article/7644116bc43d48a9ab450a50196b179a>.
- [2] J.-L. Samaan, “The proliferation of drones in the middle east has done nothing to change the strategic order there”, *Orient XXI*, 2022. [Online]. Available: <https://orientxxi.info/magazine/the-proliferation-of-drones-in-the-middle-east-has-done-nothing-to-change-the>, 5393 (visited on 05/15/2022).
- [3] M. G. Levy, “Drones have transformed blood delivery in rwanda”, *WIRED*, 2022. [Online]. Available: <https://www.wired.com/story/drones-have-transformed-blood-delivery-in-rwanda/> (visited on 05/15/2022).
- [4] W. Africa, “Covid-19 response in rwanda: Use of drones in community awareness | who | regional office for africa”, *World Health Organization*, Jul. 2020. [Online]. Available: <https://www.afro.who.int/news/covid-19-response-rwanda-use-drones-community-awareness>.
- [5] D. Pedro, J. P. Matos-Carvalho, F. Azevedo, *et al.*, “Ffau—framework for fully autonomous uavs”, *Remote Sensing 2020, Vol. 12, Page 3533*, vol. 12, p. 3533, 21 Oct. 2020, ISSN: 2072-4292. DOI: 10.3390/RS12213533. [Online]. Available: <https://www.mdpi.com/2072-4292/12/21/3533/htm%20https://www.mdpi.com/2072-4292/12/21/3533>.
- [6] A. W. Services and G. R. Energy, *Ge renewable energy case study | amazon eks | aws*. [Online]. Available: <https://aws.amazon.com/solutions/case-studies/ge-renewable-energy/>.
- [7] A. W. Services, *What is aws*, 2022. [Online]. Available: <https://aws.amazon.com/what-is-aws/>.
- [8] A. W. Services, *Aws pricing*, 2022. [Online]. Available: <https://aws.amazon.com/pricing/>.
- [9] A. W. Services, *Global infrastructure regions and azs*. [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/.

5. Bibliography

- [10] B. Gozluklu and R. Gupta, *Automating wind farm maintenance using drones and ai | aws for industries*, Jul. 2021. [Online]. Available: <https://aws.amazon.com/blogs/industries/automating-wind-farm-maintenance-using-drones-and-ai/>.
- [11] A. W. Services, *Construct hub*. [Online]. Available: <https://constructs.dev/>.
- [12] A. W. Services, *What is the aws cdk? - aws cloud development kit (cdk) v2*. [Online]. Available: <https://docs.aws.amazon.com/cdk/v2/guide/home.html>.
- [13] A. W. Services, *Apps - aws cloud development kit (cdk) v2*. [Online]. Available: <https://docs.aws.amazon.com/cdk/v2/guide/apps.html>.
- [14] M. Larsen and F. Gont, *Recommendations for Transport-Protocol Port Randomization*, RFC 6056, Jan. 2011. DOI: 10.17487/RFC6056. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc6056#section-2>.