ECE547/CSC547 Cloud Architecture

Cloud Architecture Report

AirDrop Innovations Pvt. Ltd.

Fall 2023

Jayraj Mulani, jmulani2
Yashasya Shah, yshah3

November 24, 2023

Contents

A note on plagiarism

We, the team members, understand that copying & pasting material from any source in our project is an allowed practice; we understand that not properly quoting the source constitutes plagiarism. All team members attest that we have properly quoted the sources in every sentence/paragraph we have copy & pasted in our report. We further attest that we did not change words to make copy & pasted material appear as our work.

1 Introduction

1.1 Motivation

We are motivated by the opportunity to reconsider package delivery in a world where life is moving at an ever-increasing speed. Our drive arises from the need to transform a sector characterized by the conventional, frequently inefficient means of moving goods. We envision a revolutionary opportunity to combine efficiency, sustainability, and creativity at the nexus of cloud technology with drone-based delivery. Drone technology represents the future of logistics. Drones provide unparalleled efficiency, ensuring swift, direct, and eco-friendly deliveries. Their ability to handle difficult terrain, plan delivery routes, and maintain real-time tracking aligns perfectly with the challenging needs for customers.

This package delivery process would be enhanced by leveraging cloud technology, which enables scalable infrastructure, real-time updates, and seamless communication. With its ability to create a centralized repository, provide accurate data analytics, and real-time monitoring options, it forms the foundation of our system. This encourages us to make use of the cloud's capabilities to protect, expedite, and optimize every step of the drone package delivery process.

1.2 Executive Summary

Utilizing drone technology has emerged as a game-changing solution in a world where the desire for effective, quick, and sustainable package delivery is on the rise. Drone package delivery presents unique problems including but not limited to logistics, route optimization, real-time monitoring, and compliance. These issues can be addressed by cloud-based technology, which also paves the way for other scalable and effective drone package delivery services. This architecture aims to offer unique solutions to cater industry specific needs, some of which are listed below:

- **Scalable Drone Fleet Management:** This solution provides a platform to manage and scale the drone fleet. It enables dynamic allocation of drones based on delivery demands.
- **Real-time Route Optimization:** Integrated with GPS and weather data, the system optimizes delivery routes, ensuring packages reach their destinations quickly and safely.
- **Remote Monitoring and Control:** Operators can remotely monitor drone health, flight status, and intervene in real-time if necessary. This feature minimizes downtime and enhances overall efficiency.
- **Package Tracking and Customer Alerts:** Customers can track their deliveries in real-time, receiving notifications as their package approaches its destination.
- **Compliance and Regulatory Support:** The system helps the operator stay compliant with aviation regulations, ensuring the safe and legal operation of the drone fleet.
- **Data Analytics for Performance:** Cloud-based analytics provide insights into drone performance, delivery times, and customer preferences, allowing for continuous service improvement.

2 Problem Statement

2.1 The Problem

Traditional package shipping methods haven't been able to keep up with the increasing demands of efficiency, speed, and environmental responsibility in a world that is evolving rapidly. Present systems struggle to keep up with the demands of both businesses and customers since they tend to be inefficient and unsustainable. The challenge at hand involves bridging the gap by leveraging drone delivery; and cloud computing can offer a bunch of advantages to revolutionize the package delivery sector. The central objective of this project is to provide an innovative and scalable solution that optimizes logistics efficiency while enabling prompt, safe, and customer-focused delivery services.

2.2 Business Requirements

[BR1] Optimize costs
Reduce operational expenditures by utilizing cloud technology and optimizing resource distribution to handle fluctuating workloads.

[BR2] Tenant Identification
A robust tenant identification system is essential to ensure the segregation of data and resources, granting access to authorized users within a diverse user base comprising of end consumers, e-commerce sites, operation executives, etc.

[BR3] Real-time Monitoring and Control
Develop a real-time monitoring and control system to manage drone operations and ensure prompt interventions when required.

[BR4] Customer Engagement
Provide real-time parcel tracking and timely client alerts to enhance the delivery experience.

[BR5] Performance Measurement
Utilize cloud-based analytics to continuously measure and improve performance based on key metrics such as delivery times and customer satisfaction.

[BR6] Data Security
Ensure the security and privacy of customer and operational data throughout the system.

[BR7] Continuous Service Improvement
Establish a framework for continuous enhancement based on performance analytics and feedback.

[BR8] High Availability
Ensure that the system is available at all times i.e. 24/7.

[BR9] Delayed Data Collection
Some drones may not be connected to the internet at all times, yet all the data need to be synced to the cloud as and when the network connectivity is established.

2.3 Technical Requirements

Business Requirements	**Technical Requirements**	**Justification**
[**BR1**] Optimize cost	[**TR1.1**] Implement Auto-Scaling	Auto-scaling can prevent unwanted costs by scaling down resources when load decreases. [1]
	[**TR1.2**] Setup workload monitoring and prediction	Setting up workload monitoring and prediction helps right-size resources, avoiding over-provisioning and reducing unnecessary costs.
	[**TR1.3**] Health Checks	It ensures that resources are active and healthy, reducing downtime costs and improving resource efficiency. [2]
	[**TR1.4**] Budgeting alerts	It helps in proactively managing and controlling cloud costs; prevents overspending, and promotes cost-efficient resource allocation.[3]
[**BR2**] Tenant Identification	[**TR2.1**] Multi-Tenancy Architecture	A multi-tenant architecture enables isolation of resources per tenant. This facilitates each tenant (including e-commerce sites, operation executives and end consumer to operate within a secure, private space that is logically separated from other tenants. [4]
	[**TR2.2**] Identity and Access Management (IAM)	Implement a cloud-based IAM system that supports role-based access control (RBAC) to enforce different levels of permissions and access for various users, ensuring that only authorized individuals can access certain data or resources.[35]
	[**TR2.3**] Authentication and Authorization Services	Incorporate secure authentication protocols such as OAuth 2.0, OpenID Connect, or SAML 2.0 for verifying user identities. Additionally, the system will support authorization services to grant permissions based on verified credentials.
[**BR3**] Real time Monitoring and Control	[**TR3.1**] Establish MQTT communication channel(s)	MQTT is a lightweight, efficient protocol for real-time communication, making it ideal for applications that require real-time communication. Its publish-subscribe mechanism eliminates the need for resource-intensive polling, further enhancing its efficiency. [6]
	[**TR3.2**] Implement logging	Logging records all system activities, facilitating audit trails and accountability. Moreover, they help in quicker debugging, thereby reducing issue resolution time.[7]
[**BR4**] Customer Engagement	[**TR4.1**] Update status of deliveries in real time on customer dashboard.	Keeping customers updated about their deliveries, or any unexpected delays help in customer engagement.
	[**TR4.2**] Implement email / notification delivery service	It is helpful for customers to receive real time updates about their deliveries. It enhances customer engagement and improves trust and transparency.

Business Requirements	**Technical Requirements**	**Justification**
[**BR5**] Performance Measurement	[**TR5.1**] Enable monitoring and generate analytics	Data collected while monitoring is later subjected to cloud based analytics that can help identify trends, areas for improvement, and opportunities for optimization.
	[**TR5.2**] Implement logging	Logs store historical data for analysis and trend identification in real-time systems. [7]
[**BR6**] Data Security	[**TR6.1**] Keep data encrypted	Data encryption adds a layer of security for sensitive data like home addresses, phone numbers, delivery routes, etc. [8]
	[**TR6.2**] Implement rate limiting (1000 requests/second)	Rate limiting can prevent Denial of Service attacks and enhance security.
	[**TR6.3**] Health Checks	Regular health checks can help flag potential issues proactively [2]
	[**TR6.4**] Implement logging	Logging records all system activities, facilitating audit trails and accountability. [7]
	[**TR6.5**] Setup distributed architecture	Distributed architectures can implement security measures such as data encryption and network segmentation to protect sensitive data and reduce the attack surface.
[**BR7**] Continuous Service Improvement	[**TR7.1**] Enable monitoring and generate analytics	Generating analytics implies that the monitoring data collected will be processed and analyzed using cloud-based analytics tools to extract meaningful insights and metrics. These analytics can help identify trends, areas for improvement, and opportunities for optimization.
	[**TR7.2**] Health Checks	Health checks provide valuable insights and data which can be analyzed to assess system performance. By utilizing performance analytics derived from health checks, a framework for continuous improvement can be established. [2]
	[**TR7.3**] Implement logging	Logging acts as a reliable interface to continuously monitor the operations happening in the system. Logs can periodically reflect key performance metrics that help in regularly monitoring performance. [7]

Business Requirements	**Technical Requirements**	**Justification**
[**BR8**] High Availability	[**TR8.1**] Implement Auto-Scaling	Auto-scaling ensures high availability by scaling up when load increases. [9]
	[**TR8.2**] Setup workload monitoring and prediction	Monitoring workloads is crucial to ensure sufficient availability of resources, especially when the system is expected to make computations in real-time
	[**TR8.3**] Implement rate limiting	Rate limiting ensures that there are sufficient resources available to serve all the requests, hence ensures high availability. [10]
	[**TR8.4**] Implement version control	Version Control is useful for keeping track of the changes in the code and different deployment versions and in case there is an issue when deploying the newer version, it immediately rolls back to the original version, hence ensuring high availability. [11]
	[**TR8.5**] Enable disaster management and recovery	Disasters are hard to predict and most systems are prone to disasters. A clearly stated disaster management and recovery strategy will ensure high availability.
	[**TR8.6**] Health checks	Conducting health checks helps in ensuring system availability by identifying potential issues that could lead to downtime. Through early detection and resolution of these issues, the system's availability is maintained on a 24/7 basis, meeting the high availability requirement. [12]
	[**TR8.7**] Implement logging	Logging acts as a reliable interface to continuously monitor the operations happening in the system. Configuring apt log levels can help in easier debugging, ensuring high availability.
	[**TR8.8**] Setup distributed architecture	Distributing components across multiple servers or locations enhances system availability. In case of hardware failures or network issues, the system can continue to function, minimizing downtime and ensuring uninterrupted services. [13]
[**BR9**] Delayed Data Collection	[**TR9.1**] Automated data synchronization	All drones may not be connected to the internet at all times. Use of MQTT or other polling mechanism may be needed to ensure that the drones support delayed data synchronization. This would also entail equipping drones with the capacity to store data locally and push the same to cloud when possible. [14]

2.3.1 Conflicting TRs

1. **TR3.1 (Establish MQTT Communication Channel) vs. TR6.1 (Keep Data Encrypted)**

   MQTT is designed for efficient, lightweight communication, which is essential for real-time control and monitoring. But it doesn't support encryption by default. However, the requirement to keep all data encrypted (TR6.1) conflicts with the TR3.1 that requires us to establish MQTT based communication. Moreover, keeping all data encrypted might introduce latency or require more processing power, potentially diminishing the efficiency of the MQTT protocol.

2. **TR1.1 (Auto-Scaling) vs. TR6.2 (Rate Limiting)**

   Auto-scaling is designed to increase resources to handle more load, whereas rate limiting restricts the number of requests to prevent overloading. While both aim at efficiency and security, they operate on opposing principles - one expands capacity, and the other restricts access. Balancing these could be challenging.

2.4 Trade-offs

1. **[BR1] Optimize costs vs. [BR8] High availability**

   Achieving high availability in cloud-based systems demands a distributed architecture, a strategy that involves duplicating cloud resources across various availability zones and regions. This distributed setup serves as a robust assurance of system availability, ensuring continued operation even if one zone or region faces disruptions. However, this approach comes with substantial expenses tied to replicating and overseeing resources spread across diverse locations. Moreover, data movement between these regions can contribute significantly to the overall cost.

   High availability isn't merely about redundancy; it also encompasses the establishment of robust disaster recovery protocols. These mechanisms act as a safety net during unexpected disasters or failures, allowing systems to swiftly recover and maintain operations. Yet, integrating and constantly maintaining these disaster recovery protocols involves ongoing operational expenses. Regularly testing and updating these measures to ensure they remain effective adds to the operational overhead, despite their critical role in safeguarding against potential catastrophes. Balancing the need for high availability with the cost of implementing and managing these resilient systems is an ongoing challenge for organizations operating in the cloud.

2. **[BR6] Data security vs. [BR9] Delayed Data Collection**

   Data security is paramount for drone delivery services, as they handle sensitive customer and operational information. To safeguard this data, stringent access controls, robust encryption, and secure data transfer protocols must be implemented. The intermittent internet connectivity experienced by drones poses a challenge for real-time data collection. To address this, data can be temporarily stored locally on drones until they regain network connectivity. However, this introduces potential security risks, which our solution prioritizes mitigating.

   Our solution employs encryption for stored data on drones, safeguarding sensitive information even in offline scenarios. Secure data transmission protocols, such as SSL/TLS, ensure that data remains protected during transfer to the cloud. Additionally, appropriate synchronization mechanisms minimize security risks while ensuring data collection and availability.

3 Provider Selection

3.1 Criteria for choosing a provider

1. **Scalability and Performance Capabilities:**
   - This criterion evaluates the ability of the cloud service to efficiently scale up or down according to the demand.
   - It's crucial for handling varying workloads, especially in a dynamic environment like drone-based delivery where the number of drones and the volume of data can fluctuate significantly.
   - Performance capabilities refer to how well the service can maintain speed and efficiency even as it scales.
2. **Data Security and Compliance:**
   - This involves assessing the security measures and protocols in place to protect sensitive data, such as customer information and operational data.
   - Compliance refers to the adherence of the cloud service to industry standards and regulations, which is vital in drone operations where privacy and data security are paramount.
3. **Real-Time Data Processing and Analytics:**
   - Given the need for immediate data processing in drone operations (like route optimization, weather updates, etc.), this criterion looks at how well the cloud service can handle and analyze large streams of data in real time.
   - This capability is essential for making quick decisions and ensuring efficient operations.
4. **Support for IoT and Edge Computing:**
   - This criterion assesses the cloud provider's ability to integrate with and support Internet of Things (IoT) devices, in this case, drones.
   - It includes evaluating tools and services for managing these devices, processing data at the edge (i.e., close to where it's generated), and ensuring seamless communication between the drones and the cloud.
5. **Cost-Effectiveness and Pricing Flexibility:**
   - This involves considering the overall cost of using the cloud service, including any hidden costs.
   - Pricing flexibility is also important; the service should offer a pricing model that aligns with the company's budget and scaling needs.
   - It's essential to find a balance between cost, performance, and features to ensure the project is economically viable.
6. **High Availability and Disaster Recovery:**
   - The service should guarantee high availability, with strategies for disaster management and recovery to ensure the system remains operational 24/7, even in the event of unexpected disruptions.

3.2 Provider Comparison

Criteria	AWS	GCP	Azure
Scalability and Performance Capabilities	AWS leads in scalability due to its extensive global infrastructure, which allows for rapid scaling. Amazon EC2 provides a wide variety of instance types and sizes, making it highly adaptable to different workloads. AWS Auto Scaling simplifies the process of scaling in and out, ensuring that the application maintains the desired performance. **Rank: 1**	GCP is known for its high performance, especially in data-intensive applications. Google Compute Engine offers powerful and efficient virtual machines, while Google Kubernetes Engine is renowned for its containerized application management, making it an excellent choice for applications that require both scalability and data processing power. **Rank: 2**	Azure's scalability is robust, but it's often perceived as more complex to set up and manage compared to AWS and GCP. Azure VM Scale Sets and Azure Kubernetes Service provide good scalability options, but they might require more management effort, which can be a determining factor in dynamic, high-scale environments. **Rank: 3**
Data Security and Compliance	AWS has a comprehensive set of security and compliance services. Its IAM system is particularly strong, offering granular control over resources. AWS also covers a wide range of compliance certifications, but Azure slightly edges it out in terms of the breadth and depth of compliance-specific offerings. **Rank: 2**	GCP offers Google Cloud KMS for encryption key management and Cloud Identity and Access Management (IAM) for access controls. It also employs encryption by default for data at rest and in transit. It also employs encryption by default for data at rest and in transit. **Rank: 3**	Microsoft Azure stands out in this category, especially with its focus on enterprise-level security. It offers an extensive range of compliance solutions, tailored for various industries. Azure's security center provides advanced threat protection, making it a reliable choice for projects requiring stringent data security measures. **Rank: 1**
Real-Time Data Processing and Analytics	AWS Kinesis excels in processing large streams of real-time data, making it ideal for applications like drone monitoring, where data is constantly generated. Its integration with other AWS services, such as AWS analytics and storage solutions, provides a comprehensive ecosystem for real-time data handling. **Rank: 1**	GCP's BigQuery is a powerful tool for analytics, offering fast and efficient querying capabilities. Dataflow complements this by providing a robust, real-time data processing service. However, in direct comparison with AWS for real-time streaming, AWS has a slight edge in terms of maturity and integration options. **Rank: 2**	Azure's tools are powerful but can be more complex to integrate and manage. Azure Stream Analytics is effective for real-time analytics, but it may not be as seamless or as well-integrated as AWS's or GCP's offerings. **Rank: 3**

Criteria	AWS	GCP	Azure
Support for IoT and Edge Computing	AWS's IoT services are comprehensive, offering everything from device management to edge computing solutions. AWS Greengrass, in particular, is notable for allowing local compute, messaging, data caching, and sync capabilities for connected devices. **Rank: 1**	While GCP's IoT services are robust and reliable, they are generally considered to be a step behind AWS and Azure in terms of the breadth of features and the ecosystem's maturity. **Rank: 3**	Azure provides a very competitive set of IoT tools, with Azure IoT Hub excelling in device management and monitoring. Azure IoT Edge enables devices to perform edge computing, which is crucial for drones operating in areas with limited connectivity. **Rank: 2**
Cost Effectiveness and Pricing Flexibility	AWS offers a comprehensive pay-as-you-go model, which can be cost-effective for scalable projects. However, the complexity of AWS's pricing model can sometimes make cost prediction and management challenging. AWS has Budget and alerts where you can set Budgeting alerts so the cost remains under control. **Rank: 2**	Google Cloud Platform is often praised for its innovative pricing model, which includes sustained use discounts and custom machine types, allowing for significant cost savings, especially for long-term use. Overall GCP aims to offer more competitive pricing options as compared to its peers. **Rank: 1**	Azure also has a flexible pricing model, but it can be complex and sometimes more expensive, particularly for larger or more complex deployments. Azure's cost-benefit becomes more evident in organizations already invested in Microsoft's ecosystem. **Rank: 3**
High Availability & Disaster Recovery	AWS offers multiple availability zones (AZs) within regions, allowing redundancy and fault tolerance across these zones. This setup enhances HA by enabling users to distribute their resources across different AZs for increased resilience against failures. It also various DR solutions to enable businesses to create backup plans, replicate data across regions. **Rank 1**	GCP employs a different approach with its multi-region deployments. It emphasizes regional redundancy rather than availability zones and offers a global load balancing service for distributing traffic across regions. Google Cloud's DR options include services like Google Cloud Storage for backup and Google Cloud Disaster Recovery. **Rank 3**	Azure operates with a similar concept of availability zones like AWS, providing redundancy within regions for improved HA. Azure Load Balancer and Traffic Manager help in distributing traffic across zones. It offers a global load balancing service for distributing traffic across regions and also enables failover and failback procedures, ensuring continuity during outages or disasters. **Rank: 2**

3.3 The final selection

As we can see from the above comparison table, that all the cloud providers offer some or the other services and features which make them rank on top per our criteria. The provider with **rank 1** is highlighted in blue. On the first glance, it might appear that it is difficult to choose among any provider. However, upon taking a closer look **AWS** turns out to be taking the winner spot. AWS offers the most comprehensive set of services required to satisfy most if not all the technical requirements laid out in Section 2.3. Below are the list of services offered by the winner. *Please note that we have described in detail only 8 services below (4 per team member). We will be describing how they and others link with our project in the coming sections 4 and 4.2*:

Criteria	**Corresponding AWS Services**
Scalability and Performance Capabilities	AWS EC2, AWS Lambda
Data Security and Compliance	AWS IAM, AWS KMS, AWS WAF
Real-Time Data Processing and Analytics	AWS Kinesis
Support for IoT and Edge Computing	AWS IoT Core
Cost-Effectiveness and Pricing Flexibility	AWS Budgets, AWS Cost Explorer
High Availability and Disaster Recovery	AWS Application Load Balancer, AWS S3

3.3.1 The list of services offered by the winner

1. **Amazon QuickSight**
   - Amazon QuickSight [15] is a scalable, serverless, embeddable, machine learning-powered business intelligence (BI) service built for the cloud.
   - It allows us to create and publish interactive BI dashboards that include ML-powered insights. Moreover, it can connect to a variety of data sources, including Amazon S3, Amazon Redshift, Amazon RDS, and Amazon DynamoDB.
2. **Amazon Managed Grafana**
   - Grafana [16] is a fully managed and secure data visualization service that allows users to query, correlate, and visualize operational metrics, logs, and traces across multiple data sources.
   - Grafana is open source and offers support for Grafana's vast library of pre-built dashboards and plugins for numerous data sources. It can also scale to handle the largest datasets, and it offers a variety of security features.
3. **Amazon Simple Storage Service (S3)**
   - Amazon S3 [17] is an object storage service offering industry-leading scalability, data availability, security, and performance. It offers a pay-as-you-go model so that we end up paying only for the storage that we actually use.
   - S3 is designed to provide 99.9% of durability and stores data for millions of applications used by market leaders in every industry.
4. **Amazon CloudFront**
   - Amazon CloudFront [18] is a fast content delivery network (CDN) service that securely delivers data, videos, applications, and APIs to customers globally with low latency and high transfer speeds.
   - CloudFront is integrated with AWS – both physically and via the AWS network backbone – using the same security measures as other AWS services.

5. **Amazon API Gateway**
    - Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. It can handle millions of API calls per second.
    - API Gateway [19] handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management.
6. **AWS WAF (Web Application Firewall)**
    - AWS WAF [20] is a web application firewall that helps protect web applications or APIs against common web exploits and bots that may affect availability, compromise security, or consume excessive resources.
    - AWS WAF give users control over which traffic to allow or block to web applications by defining customizable web security rules. This includes rules based on rate, geolocation and IP address among others.
7. **Amazon Elastic Container Service (ECS)**
    - Amazon ECS [21] is a highly scalable, high-performance container orchestration service that supports docker containers, enabling us to run and scale containerized applications on AWS.
    - ECS eliminates our need to install, operate, and scale our own cluster management infrastructure. Being secure and reliable (99.9% availability SLA), it is designed to run containerized applications at scale. It offers a variety of security features, such as access control, logging, and auditing.
8. **AWS Lambda**
    - AWS Lambda [22] is a serverless compute service that lets users run code without provisioning or managing servers, and it offers pay-as-you-go model.
    - With Lambda, we can run code for virtually any type of application or backend service with zero administration.
9. **Application Load Balancer** [23]
10. **AWS Key Management Service (KMS)** [24]
11. **AWS Cloudwatch** [26]
12. **Amazon DynamoDB** [25]
13. **Amazon ElastiCache** [27]
14. **AWS IoT Core** [28]
15. **AWS Route53** [29]
16. **AWS Cost Explorer** [30]
17. **AWS Budgets** [31]
18. **AWS Kinesis** [32]
19. **Amazon EC2** [33]
20. **AWS Auto Scaling** [34]
21. **AWS Identity and Access Management(IAM)** [35]
22. **AWS Simple Email Service (SES)** [42]

4 The first design draft

Phew... So many services. Well, no complaints, mainly because they are crucial for our cloud architecture. In this section, we will explore how services act as fundamental building blocks in this architecture, how they will be grouped together to form layers / blocks. Then, in the later section 5 we will see how they interact with one another. We would also use Cloud-formation diagrams to describe the same.

4.1 The basic building blocks of the design

Lets be creative. Lets think of the final cloud architecture as a delicate dish that we are cooking. If so, services are analogous to the ingredients we need to prepare such a delicacy. Each service plays its part, providing some value to our "dish" called cloud architecture. When combined, these services collectively fulfill most, if not all the requirements that we discussed in prior sections. We are going to divide (or combine?) these services to form 5 key building blocks for the architecture. We describe each of them in detail here (along with the services that were skipped in Section 3.3.1). Let's begin cooking.

4.1.1 Compute

In our architecture, we would adopt the Microservices architecture paradigm. It is an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs.[36]. In our case, these are the chefs and sous-chefs; as they would define and implement the 'logic' of the system. Some of the services we plan to use in the architecture that fall in this category are described below:

1. **Amazon EC2**: This is the "pan" in the kitchen, here is where (almost) all the logic is executed. Combined with autoscaling capabilities, EC2 [33] instances are perfect for handling varying workloads while optimizing costs.

2. **Amazon Elastic Container Service (ECS)**: This service will make the deployment process simple, scalable and less error prone. This would eliminate the need for us to install, operate, and scale our own cluster management infrastructure.

3. **AWS Lambda**: We need some infrastructure that can respond to events. AWS Lambda can be leveraged for an Event-driven execution [40]. We can run code for virtually any type of application or backend service with zero administration.

4.1.2 Storage

Storage is another crucial building block of our architecture. This would include storage of data, files and even meta-data. We can imagine this as the vessel in which our ingredients and / or the finished dish resides. AWS offers many different services concerning secure, scalable storage of data. Among those, we are planning to leverage the following services in our architecture.

1. **Amazon Simple Storage Service (S3)**: It is an object storage service offering industry-leading scalability, data availability, security, and performance [17]. It is designed to provide 99.9% of durability. This will enable us to store files and objects on cloud. We can also store archives of reports or logs.

2. **Amazon DynamoDB**: It is a key-value and document database that delivers single-digit millisecond performance at any scale [25]. It is a fully managed, multi-region, multi-master database with built-in security, backup and restore, and in-memory caching for internet-scale applications. We choose to leverage a NoSQL database so that it offers us sufficient flexibility in terms of defining data model.

3. **Amazon ElastiCache**: It is a web service that makes it easy to deploy, operate, and scale an in-memory cache in the cloud. It allows us to improve the performance of web applications by providing the ability to retrieve information from fast, managed, in-memory caches, instead of relying entirely on slower disk-based databases.

4.1.3 Security

Security is probably an "invisible", yet crucial ingredient of our architectural cuisine. In a drone-based delivery service, security within the cloud architecture is crucial for protecting customer data like delivery locations, ensuring safe transmission of flight instructions, preventing hacking attempts on drones, and securing the network to prevent data interception or manipulation, all of which are vital for maintaining trust and the integrity of the delivery operations. We leverage the following services to enhance the security.

1. **AWS WAF (Web Application Firewall)**: A web application firewall helps protect web applications or APIs against common web exploits and bots that may affect availability, compromise security, or consume excessive resources. [20]

2. **AWS KMS (Key Management Service)**: It is a managed service that makes it easy for us to create and manage cryptographic keys used to secure data. It is easily integrated with other AWS services to encrypt the data and control access to the keys that decrypt it. [24]

3. **AWS IAM (Identity and Access Management):** IAM is used to securely manage identities and access to AWS services and resources [41]. It would help in controlling access to drone control systems, ensuring only authorized personnel can manage and monitor drone operations, preventing unauthorized access to critical flight data or control mechanisms, thus enhancing overall system security and integrity.

4.1.4 Monitoring & Visualization

Monitoring and visualization in a drone delivery service's cloud architecture is meant to offer real-time insights into drone performance, delivery statuses, airspace conditions, and system health. This enables operators to track drones, identify issues promptly, optimize routes, ensure compliance with regulations, and enhance overall operational efficiency and safety. Here are some of the services that we leverage for the same.

1. **Amazon CloudWatch**: It will provide real-time monitoring and logging for various AWS resources and applications within the drone delivery service's cloud architecture [26]. It also helps track metrics, set alarms for critical events, and gain insights into system health and performance, ensuring timely responses to operational issues and optimizing resource utilization.

2. **Amazon Managed Grafana**: Grafana complements CloudWatch by offering advanced visualization capabilities. It allows for the creation of customized dashboards and graphical representations of CloudWatch metrics, providing a more user-friendly and customizable interface for monitoring drone operations. Grafana can enhance the visualization of complex data sets, aiding in better decision-making and system analysis.

3. **Amazon QuickSight** QuickSight serves as a comprehensive business intelligence tool, leveraging the data collected from CloudWatch and other sources within the drone delivery service's architecture [15]. It facilitates in-depth analytics, data exploration, and the creation of insightful visualizations and reports. QuickSight enables stakeholders to gain valuable insights into operational trends, performance metrics, and customer behavior, supporting data-driven decision-making for continuous improvement and optimization of the delivery service.

4.2 Top-level, informal validation of the design

Service	Role in Project	Mapped TR	Justification
Amazon QuickSight [15]	This service is used for creating interactive BI dashboards, analyzing drone performance, and deriving insights through machine learning-powered analytics.	TR5.1, TR7.1	This service provides powerful visualization tools that help in monitoring performance and identifying areas for service improvement. QuickSight also enables creation of meaningful analytics to make informed business decisions.
Amazon Managed Grafana [16]	This service offers data visualization for operational metrics, logs, and traces, aiding in the effective monitoring and control of the drone fleet.	TR3.2	This service enables detailed and intuitive visualization of system logs, essential for real-time monitoring and troubleshooting.
Amazon S3 [17]	This service provides secure, scalable object storage for managing data and metadata including files, archived, backups, data dumps, etc.	TR1.2, TR6.5, TR8.5, TR8.8	S3 buckets can be used to store archived reports generated while performing workload monitoring. It can also be leveraged as a backend for services like Terraform, which would make it easy to setup distributed architecture. Data is regularly backed up on S3 buckets to enable disaster recovery.
Amazon CloudFront [18]	This service accelerates content delivery, including API responses and application data, ensuring low latency and high transfer speeds globally.	TR4.1	CloudFront can be used to deliver real-time drone delivery status updates to users. This can help users track the progress of their deliveries and see when they can expect their packages to arrive. It can also be used to deliver drone delivery marketing content, such as ads and promotional materials. This can help attract new customers to the drone delivery service.
Amazon API Gateway [19]	This service manages APIs for drone control and customer interaction, ensuring secure, scalable, and efficient handling of API requests.	TR4.1, TR6.2	API Gateway can be used to expose drone delivery data and functionality to applications, such as mobile apps, web apps, and IoT devices. This can allow users to track the status of their deliveries, receive delivery notifications, and even schedule drone deliveries. It can also be used to enforce rate limiting at the gateway itself.
AWS WAF (Web Application Firewall) [20]	This service protects web applications and APIs from common web exploits, enhancing the overall security of the drone delivery system.	TR6.1, TR6.2	Rate-based rules allow us to limit the number of requests that can be made to the web applications from a particular IP address or range of IP addresses. This can help protect against denial-of-service (DoS) attacks.

Service	Role in Project	Mapped TR	Justification
Amazon Elastic Container Service (ECS) [21]	This service manages containerized applications, simplifying deployment and scaling of the cloud infrastructure.	TR6.5, TR8.4, TR8.5, TR8.8	This service provides a robust container orchestration service that supports version control and aids in disaster management. It also enables us to easily deploy applications, hence facilitating in developing a distributed architecture.
AWS Lambda [22]	This service enables serverless computing, running code for various back-end services without the need for server management.	TR1.1, TR4.1 TR8.1	This service optimizes costs and ensures consistent performance and high availability through its auto-scaling capabilities. Since it can respond to events, we can leverage the same to ensure the status of delivery is updated in real time.
ALB (Application Load Balancer [23]	This service manages web traffic across multiple targets, ensuring efficient routing and load balancing of incoming requests.	TR1.1, TR6.5, TR6.2, TR8.3, TR8.8	This service provides scalable traffic management and workload monitoring, crucial for maintaining high availability. It will help dividing load across distributed compute instances. It can also help in implementing rate limiting at the LB layer.
AWS KMS (Key Management Service) [24]	This service manages cryptographic keys, ensuring the security of encrypted data across the system.	TR6.1	This service provides a secure way to manage encryption keys, thereby enhancing the overall data security of the system.
Amazon DynamoDB [25]	This service provides a fast, scalable NoSQL database service for managing key-value and document data.	TR6.5, TR8.5, TR8.8	This service offers a distributed database architecture that enhances data security and ensures high availability.
Amazon ElastiCache [27]	This service improves the performance of web applications by providing a fast, managed in-memory cache.	TR4.1	This service reduces downtime and improving the efficiency of resource usage. This is crucial for system performance and contributes in generating real time updates for drone fleet as well as customers.
AWS IoT Core [28]	This service connects devices to the AWS cloud, handling billions of messages and device interactions.	TR3.1, TR9.1	This service supports real-time communication protocols and enabling efficient data synchronization, even in intermittent network conditions.
AWS SES [42]	This service is used for reliable, scalable emails to communicate with customers at the lowest industry prices	TR4.2	AWS Simple Email Service can be used to provide delivery status updates to customers via email. This can help in customer engagement and enhance customer satisfaction.

Service	Role in Project	Mapped TR	Justification
AWS Route 53 [29]	This service plays a crucial role in domain name system (DNS) management, which is essential for efficiently routing user requests to the appropriate infrastructure resources.	TR7.2, TR8.1	This service enhances high availability by intelligently routing user requests to operational endpoints, effectively handling traffic surges and server unavailability. It also offers health checking to monitor and reroute traffic from unhealthy endpoints, maintaining the operational integrity of our system.
AWS Cost Explorer [30]	This service provides a detailed view of AWS costs and usage, aiding in effective financial management.	TR1.4, TR8.4	This service enables proactive cost management and control, aligning with budgeting requirements.
AWS Budgets [31]	This service allows setting custom budgets for tracking costs and usage, with alerts for threshold breaches.	TR1.4	This service provides tools for monitoring and controlling costs to optimize resource allocation and expenditure.
AWS Kinesis [32]	This service facilitates real-time data collection, processing, and analysis, crucial for timely insights.	TR5.1, TR7.1, TR9.1	This service provides capabilities for real-time data processing and analysis, essential for performance measurement and service improvement. We can also leverage this service to enable delayed data collection.
Amazon EC2 [33]	This service offers scalable computing resources, adapting quickly to changing requirements.	TR1.1, TR1.2, TR8.1	This service provides flexible and scalable computing capacity, crucial for workload monitoring and maintaining high availability.
AWS Auto Scaling [34]	This service automatically adjusts computing resources based on demand, ensuring optimal performance.	TR1.1, TR8.1	This service optimizes resource allocation and cost, while ensuring the system can handle varying loads efficiently.
Amazon CloudWatch [26]	This service monitors and observes AWS cloud resources and applications, tracking metrics, logs, and setting alarms.	TR3.2, TR5.2, TR6.4, TR7.3, TR8.7, TR1.3, TR6.3, TR7.2, TR8.6	This service s critical for comprehensive system logging and health checks. It enables effective monitoring, alerting, and logging capabilities across the system, supporting performance measurement, operational health monitoring, and facilitating proactive issue resolution.
Amazon IAM [41]	This service manages secure access to AWS services and resources, handling user and group permissions.	TR2.1, TR2.2, TR2.3	This service addresses Tenant Identification by enabling segregated access control for different user groups. It also fulfills Identity and Access Management and Authentication and Authorization Services requirements by providing a robust framework for managing user identities, access permissions, and secure authentication protocols, ensuring system security and data privacy.

4.3 Action items and rough timeline [Skipped]

5 The second design

5.1 Use of the Well-Architected framework

1. **Operational Excellence:**[43]
    - **Continuous Integration and Continuous Deployment:** Given our utilization of AWS ECS, we will be able to devise automation pipelines that seamlessly build, test, and deploy services across regions. Leveraging S3 buckets enables us to configure robust backups and establish a resilient backend for a Terraform-driven infrastructure, ensuring automated provisioning with efficiency and reliability.
    - **Real-time Monitoring and Updates:** By integrating Amazon CloudWatch and AWS Lambda, we've unlocked the capability to actively monitor operations in real-time while orchestrating automated responses to dynamically shifting conditions. This synergy empowers us to proactively address and adapt to evolving scenarios, ensuring a more responsive and optimized operational environment.
    - **Continuous Improvement Process:** Frequent assessments and refinements of our operational procedures, guided by insights from AWS Quicksight, constitute a pivotal element in our strategy. This iterative approach guarantees that our drone delivery system undergoes consistent evolution, aligning with and surpassing business requirements while staying adaptive and innovative in the ever-changing landscape.
2. **Security:** [44]
    - **Data Protection:** We safeguard sensitive data through a robust security framework by leveraging services such as AWS WAF and IAM and have developed defenses against unauthorized access and potential threats. This proactive approach ensures the integrity and confidentiality of our data, reinforcing our dedication to maintaining a secure and resilient operational environment.
    - **Compliance and Regulatory Support:** We shall study and make sure our follows all the rules for aviation and data protection. We aim to we meet the requirements for safe flying and keeping your information secure and private.
    - **Proactive Threat Management:** Consistent security check-ups and tapping into advanced AWS security features play a crucial role in our strategy. They enable us to stay ahead by spotting and tackling potential security risks before they become problems, ensuring a safer and more resilient system overall.
    - **Encryption:** We take data security seriously by implementing encryption at every step - during transit and while it's stored. Leveraging tools like AWS KMS alongside other encryption methods, we ensure that your data stays protected and inaccessible to unauthorized access, bolstering our commitment to keeping your information safe and secure.
3. **Reliability:** [45]
    - **Dependable Drone Operations:** Our workload is engineered for seamless scalability, effortlessly adapting to changing demand levels. This ensures consistent and dependable service delivery, regardless of fluctuations, guaranteeing a reliable experience for our users every step of the way.
    - **Service Continuity Planning:** We've employed Amazon DynamoDB and AWS Lambda services to establish resilient systems capable of managing failures and maintaining uninterrupted operations. This strategic implementation ensures our ability to navigate challenges effectively, providing consistent and reliable service without interruptions.
    - **Lifecycle Management:** We diligently conduct comprehensive testing and monitoring at every stage of our workloads' lifecycle. This meticulous approach guarantees our system's capacity to adeptly manage failures, ensuring a robust and resilient operational environment at all times.
4. **Performance Efficiency:** [46]
    - **Resource Optimization:** By employing AWS Auto Scaling and Amazon EC2, we optimize the utilization of computing resources, swiftly adapting to fluctuating demands. This strategic utilization enables us to maintain efficiency by dynamically adjusting resources, ensuring a responsive system that efficiently meets changing needs without unnecessary resource strain.

- **Technological Adaptability:** We've crafted our architecture to grow alongside emerging technologies, ensuring our system remains at the forefront of performance efficiency. This approach allows us to continuously adapt, integrating cutting-edge advancements to maintain peak performance levels, keeping our systems agile and effective in a rapidly evolving landscape.
- **Load Balancing and Elasticity:** Thanks to services like Amazon ECS and Load Balancers, we've fine-tuned our system to expertly manage fluctuating loads, ensuring peak performance consistently. This optimization guarantees that our system operates at its best, effortlessly handling varying demands without compromising on efficiency or reliability.

5. **Cost Optimization:** [47]

   - **Effective Financial Management:** We closely monitor our expenses by leveraging tools like AWS Cost Explorer and AWS Budgets. This allows us to ensure that our operations remain within our financial means, enabling effective budget management and financial sustainability.
   - **Resource Utilization Monitoring:** We continuously evaluate our resource utilization to maximize the value derived from our investments. This ongoing assessment ensures that we optimize our resources effectively, guaranteeing that every investment delivers its utmost value in supporting our operations.
   - **Cost-Efficient Scaling:** Through the strategic implementation of auto-scaling and careful selection of resource sizes, we've fine-tuned our system to provide services while minimizing costs. This optimization ensures that we deliver top-notch services while keeping expenses at the most economical level possible.

6. **Sustainability:** [48]

   - **Energy-Efficient Operations:** We're exploring ways to make our drone operations more energy-efficient, reducing the environmental impact.
   - **Cloud Infrastructure Management:** We aim to use our cloud resources in a way that minimizes energy consumption, aligning with sustainable practices.
   - **Eco-Friendly Technological Choices:** In every aspect of our project, from hardware selection to software design, we consider the environmental impact, striving to make sustainable choices.

5.2 Discussion of pillars

5.2.1 Security

The Security pillar of the AWS Well-Architected Framework is a critical component that provides comprehensive guidance for maintaining the security of AWS workloads. This pillar encompasses various aspects, including:

1. **Protection of Data, Systems, and Assets:** The primary objective of the Security pillar is to safeguard data, systems, and assets. This protection is essential for leveraging cloud technologies to enhance overall security
2. **Design Principles and Best Practices:** It offers a detailed overview of security-related design principles and best practices. These principles and practices are crucial for designing, delivering, and maintaining secure AWS workloads
3. **Key Focus Areas:** The Security pillar emphasizes protecting information and systems. Important areas of focus include ensuring the confidentiality and integrity of data, effectively managing user permissions, and establishing controls for the detection of security events
4. **Implementation Guidance:** The framework provides actionable recommendations and questions that guide users in implementing security best practices in their AWS environment. This guidance is valuable for organizations looking to optimize their security posture in the cloud

In summary, the Security pillar of the AWS Well-Architected Framework plays a pivotal role in guiding users to apply best practices and current recommendations for securing AWS workloads. It covers the protection of data and systems, offers a comprehensive overview of design principles, focuses on key security aspects, and provides practical implementation guidance.

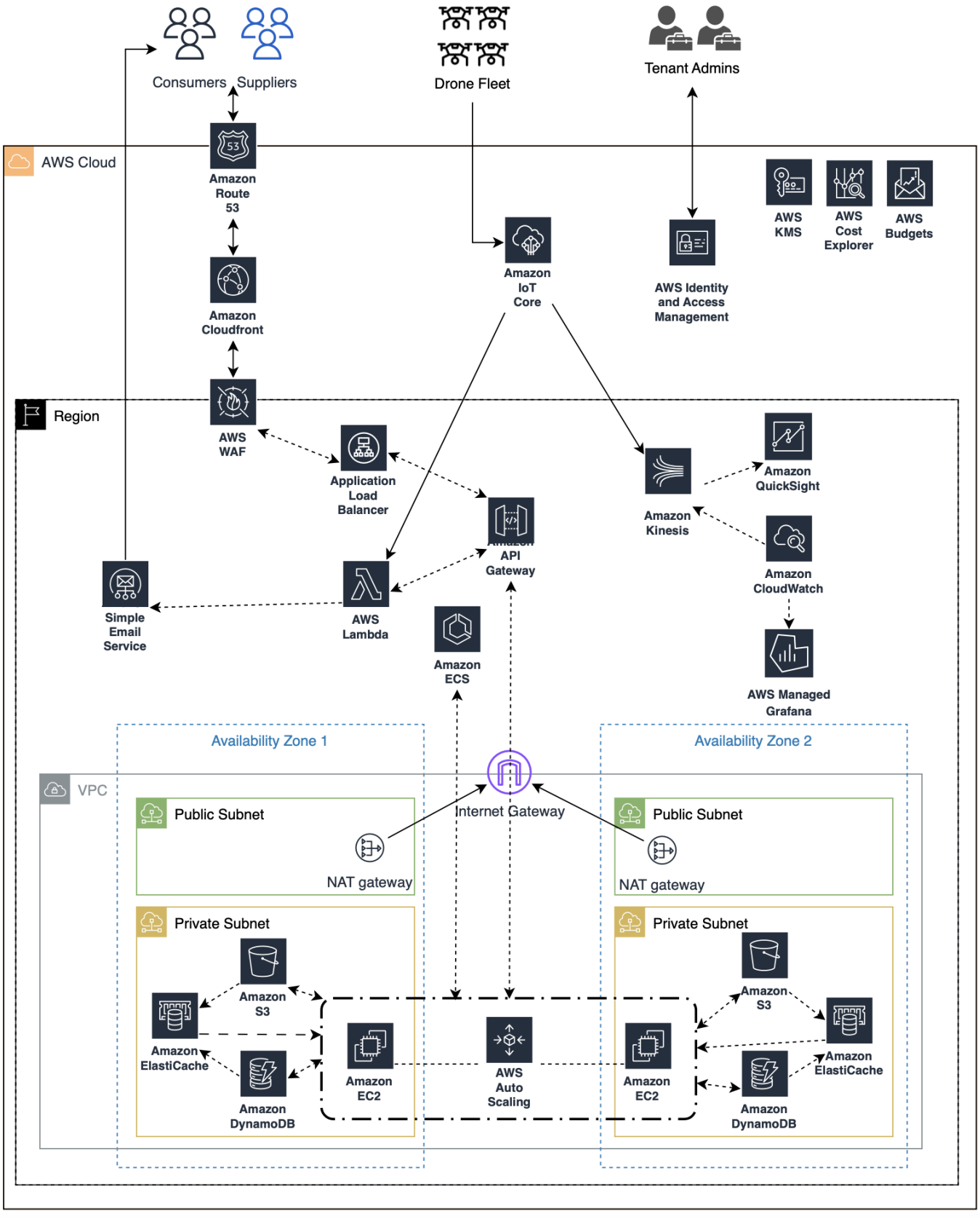
5.2.2 Reliability

The Reliability pillar of the AWS Well-Architected Framework is a fundamental component that provides essential guidance for designing, delivering, and maintaining reliable AWS environments. Here's an overview of its key aspects:

1. **Purpose and Focus:** The Reliability pillar is dedicated to ensuring that workloads perform their intended functions correctly and consistently as expected. It encompasses the entire lifecycle of a workload, focusing on its ability to operate and be tested throughout its duration
2. **Design Principles and Best Practices:** This pillar provides an in-depth overview of design principles and best practices related to reliability. Users can find prescriptive guidance on implementation in the Reliability Pillar whitepaper, which includes high-level implementation details, architectural patterns, and additional resources.
3. **Resiliency:** One of the primary aspects of reliability in cloud workloads is resiliency. This involves the ability to recover from disruptions, dynamically acquire resources to meet demand, and mitigate issues such as misconfigurations.
4. **Overcoming Traditional Challenges:** The paper on the Reliability pillar highlights how achieving reliability in traditional on-premises environments can be challenging due to factors like single points of failure, lack of automation, and elasticity. The guidance provided aims to build architectures with strong foundations, resilient structures, consistent change management, and proven failure recovery processes.
5. **Target Audience and Business Impact:** The Reliability pillar is particularly relevant for technology professionals like CTOs, architects, developers, and operations team members. Understanding and implementing the best practices and strategies for reliability can significantly enhance the likelihood of business success, as well-architected workloads are more likely to meet business objectives.

In summary, the Reliability pillar of the AWS Well-Architected Framework is designed to guide users in creating AWS environments that are reliable and resilient. It emphasizes the importance of resiliency, provides a framework for overcoming common challenges in cloud computing, and offers detailed guidance and resources for technology professionals. The adoption of these practices and principles contributes significantly to the success of business workloads in the cloud.

5.3 Use of Cloud formation diagrams



In this section, we design the cloud architecture and describe some of the key components of the same. Before we describe the diagram in detail, there are a few points to be noted here:

1. It is a multi-region deployment. Only one region is shown here for legibility purposes. But, when actually implemented, the Elastic Disaster Recovery Service would automatically replicate the entire region along with all its components.
2. The architecture diagram consists of three types of services:
   - **Zonal services:** Zonal services are confined to a specific Availability Zone (AZ) within an AWS Region. They are designed for highly localized applications where low latency and availability are critical. Examples: AWS EC2, AWS ElastiCache, etc.
   - **Regional services:** Regional services span an entire AWS Region, providing a broader reach and higher availability compared to zonal services. They are suitable for applications that require consistent performance and fault tolerance across multiple AZs within a Region. Example: AWS Application Load Balancer (ALB), AWS Lambda, etc.
   - **Global services:** Global services are available in all AWS Regions, transcending geographical boundaries. They are ideal for applications that require seamless access from anywhere in the world. Examples of global services include: AWS IAM, AWS KMS, etc.

5.3.1 Users

We can see from the diagram above that 3 different types of users that interact with our system:

1. **Suppliers and Consumers**: Suppliers are the ones who initiate the package delivery. Consumers are the ones who get the packages through drones. Both of them are crucial stakeholders in our system.
2. **Drone Fleet**: They interact with the system using IoT core service. They receive important insights from the service and publish their status to the service.
3. **Tenant Admins**: They are the ones who would create, manage and maintain the cloud infrastructure, assign roles, allocate quotas, monitor alerts, etc.

5.3.2 Flows

1. **Web Application Flow**: This flow is mainly meant for suppliers and consumers. They interact with the system using the DNS provided by Route 53 service and reach the fast and scalable CDN provided by CloudFront. This forwards the request to WAF, which sits at the edge of a region. WAF filters out the malicious requests and forwards legitimate requests to Application load balancer, which in-turn passes on the request to the API gateway. The request is then handled by Lambda or EC2 auto scaling group depending on the request. The EC2 instance communicates with S3 and DynamoDB or cache and responds to the requests. If required, the SES service notifies users about the real time updates of the drone fleet.

   We can see that each of the Availability Zone is divided into a public and a private subnet. The public subnet hosts a NAT Gateway that interacts with the Internet Gateway to communicate with the internet. Notice that this is a unidirectional data flow and external requests are not handled in this flow. The private subnet hosts all the critical compute and storage components.
2. **Logging and Monitoring Flow**: AWS CloudWatch monitors logs across all the services in a region. These logs are utilized by the Grafana service to create visualizations. They are also passed to Kineses as log streams which then sends them to QuickSight to generate analytics and further visualizations.
3. **IoT service Flow**: The drone fleet communicates with the cloud using MQTT by leveraging the IoT core service. These data is passed on as streams to Kineses for generating analytics and visualizations using QuickSight. This event driven flow triggers the lambda functions that enables us to not only monitor the data published by the drones, but also notify users about the latest updates. We can also see a connection between the lambda function that receives data from IoT core service and API Gateway. This facilitates our delayed data collection. If the drones publish the data in a delayed fashion, our APIs can get the same stored into our database.

5.4 Validation of the design

Now that we have designed the architecture, in this section, we revisit the TRs and validate each TR against our design. We mention the services satisfying the TR and justify the same.
Note: You might not see all the (repetitive) TRs as although each TR deserved a justification as to why it satisfies the BR, we thought it was redundant to list and justify the services for the same TR twice if it appears.

Business Requirements	**Technical Requirements**	**Associated Service(s) and Justification**
[**BR1**] Optimize cost	[**TR1.1**] Implement Auto-Scaling	We have used the AWS Auto Scaling service along with AWS Application Load balancer to implement Auto-Scaling. Some of the logic is handled by AWS Lambda which is inherently scalable on-demand.
	[**TR1.2**] Setup workload monitoring and prediction	As we can see from the diagram, we have setup AWS CloudWatch along with Grafana, Kineses and QuickSight to generate, monitor and visualize logs. QuickSight also generate other important business analytics helpful for the stakeholders to make informed decisions.
	[**TR1.3**] Health Checks	AWS CloudWatch implements health checks using a combination of monitoring agents, data collection.
	[**TR1.4**] Budgeting alerts	We can see in the global services that AWS Cost Explorer and AWS Budgets have been setup to effectively monitor costs and send out budgeting alerts.
[**BR2**] Tenant Identification	[**TR2.1**] Multi-Tenancy Architecture	We have used services like AWS EC2, DynamoDB and Lambda that facilitates us in creating multi-tenant isolated architectures. EC2 instances and/or Lambda functions can be isolated among tenanats using IAM roles and policies to control access to resources.
	[**TR2.2**] Identity and Access Management (IAM)	As seen in the diagram, we use AWS IAM to manage roles and fine grained access to various cloud infrastructure.
	[**TR2.3**] Authentication and Authorization Services	So far as the atchitecture itself is concerned, IAM does the authorization. For the web application, our APIs will handle the same at code level. We may rely on open sources authentication providers like Keycloak for the same.
[**BR3**] Real time Monitoring and Control	[**TR3.1**] Establish MQTT communication channel(s)	We leverage AWS IoT Core and AWS Kineses service to establish a publish-subscribe based communication mechanism with the drone fleet. This enables us to send, receive and efficiently process virtually billions of messages.
	[**TR3.2**] Implement logging	Logging plays a crucial part in monitoring. AWS CloudWatch logs all the critical and non-critical events happening across the entire cloud infrastructure. Combining the same with Grafana gives us the ability to monitor logs and visualize the same in real time.

Business Requirements	**Technical Requirements**	**Associated Service(s) and Justification**
[**BR4**] Customer Engagement	[**TR4.1**] Update status of deliveries in real time on customer dashboard.	We use highly efficient Content Delivery Networks like AWS CloudFront to build and deliver the contents on a dashboard for customers. This way, both the suppliers and consumers can get updates in (near) real-time about the drone deliveries.
	[**TR4.2**] Implement email / notification delivery service	As we can see, some of the events generated by the lambda service trigger email notifications to the customers using the AWS SES service.
[**BR5**] Performance Measurement	[**TR5.1**] Enable monitoring and generate analytics	We generate analytics and monitor the results and insights using services like AWS QuickSight and AWS Managed Grafana.
[**BR6**] Data Security	[**TR6.1**] Keep data encrypted	We use AWS EKS service combined with AWS DynamoDB and AWS S3 to keep the data encrypted.
	[**TR6.2**] Implement rate limiting (1000 requests/second)	AWS Web Application Firewall helps us with rate limiting. We can define custom policies to allow/deny requests.
	[**TR6.5**] Setup distributed architecture	We can see in the diagram that we have setup a distributed architecture by deploying services in multiple availability zones and regions.
[**BR7**] Continuous Service Improvement	[**TR7.1**] Enable monitoring and generate analytics	The analytics generated by AWS QuickSight will be analyzed by the admins that can help in continuous service improvement.
[**BR8**] High Availability	[**TR8.4**] Implement version control	We have used AWS ECS service to manage containers for the images. We can version images by release tags. This can help us roll-back to previous stable releases in case of failures in newer ones.
	[**TR8.5**] Enable disaster management and recovery	We store backups in S3 Buckets. We deploy the infrastructure in multiple availability zones, thereby ensuring disaster management and recovery
[**BR9**] Delayed Data Collection	[**TR9.1**] Automated data synchronization	We use services like AWS IoT core that leverages Publish Subscribe mechanism. As mentioned in the IoT Service flow [3], we leverage this to ensure automated data synchronization and hence enable delayed data collection.

5.5 Design principles and best practices used

5.5.1 Design Principles

1. **Perform operations as code [49]:** We use AWS Lambda alongside Amazon API Gateway to automate our operations, which aligns with our goal for real-time monitoring **(TR3.2)**. This ensures that our responses to events are consistent and error-free, leveraging the power of code to define and update our workload.

2. **Make frequent, small, reversible changes [49]:** We have used AWS Auto Scaling and Amazon EC2 which allows us to implement incremental changes, thereby enhancing our system's scalability **(TR1.1)**. This approach not only reduces risk but also empowers us to adapt swiftly to evolving requirements.

3. **Anticipate failure [49]:** We have utilized the combination of Amazon CloudWatch and Amazon S3 to proactively monitor address failures **(TR7.2)**, constantly refining our operational procedures to mitigate risks before they impact our services.

4. **Automatically recover from failure [50]:** We've configured AWS CloudWatch to initiate automatic recovery processes (TR8.6), with Amazon S3 serving as our resilient backup and recovery solution, guaranteeing continuous service availability.

5. **Test recovery procedures [50]:** We regularly test our recovery strategies (TR7.3) using AWS Lambda and Amazon EC2, ensuring our services can quickly rebound from disruptions and maintain reliability.

6. **Scale horizontally [50]:** Our architecture is designed to distribute operations across multiple Amazon EC2 instances (TR8.1), utilizing AWS Auto Scaling to maintain high availability and fault tolerance.

7. **Implement cloud financial management [51]:** We have used tools like AWS Cost Explorer and AWS Budgets which are central to our financial management strategy **(TR1.4)**, helping us to monitor and optimize our spending effectively.

8. **Adopt a consumption model [51]:** Our use of AWS Lambda and AWS Auto Scaling reflects our commitment to a consumption-based model (TR1.1), where we scale our resources to match demand, ensuring we pay only for the resources we need.

9. **Measure overall efficiency [51]:** We apply Amazon QuickSight and AWS Managed Grafana (TR5.1) to measure our workload's efficiency, ensuring that we deliver value while managing our costs prudently.

5.5.2 Best Practices

1. **Prepare, Operate, Evolve** [52][53][54]

   We have employed budgeting alerts to enhance our operational excellence. By doing so, we are not only prepared to manage our financial resources but also have a system that evolves with our spending patterns, ensuring that we stay within budget while maximizing the capabilities of AWS. This approach exemplifies our commitment to maintaining operational discipline and enhancing the predictability of our operational expenses.

   TR Satisfied - TR1.4 Budgeting alerts.

2. **Identity and Access Management**

   We've implemented a robust multi-tenancy architecture and identity management system, which are foundational to our security posture. Through these implementations, we ensure that each user and service operates with the minimum necessary privileges, enhancing the security of our system and protecting our resources from unauthorized access. This not only strengthens our compliance with the principle of least privilege but also underpins our entire security strategy.

   TRs Satisfied - TR2.1 Multi-Tenancy Architecture, TR2.2 IAM

3. **Infrastructure Protection**[56]

   We have adopted a distributed architecture to protect our infrastructure, which aligns with the reliability pillar of the framework. This setup enables us to distribute our workload across multiple resources, avoiding single points of failure and significantly reducing the potential impact of outages or attacks, thereby reinforcing our system's resilience and operational stability.

   TR Satisfied - TR8.8 Setup distributed architecture

4. **Data Protection**[57]

   We have employed AWS Key Management Service (KMS) and advanced data encryption techniques to meticulously classify and encrypt our data, both at rest and in transit. This practice is fundamental in ensuring that sensitive information is rigorously protected, thereby meeting the highest standards of confidentiality and integrity as mandated by the security pillar of the AWS Well-Architected Framework.

   TR Satisfied - TR6.1 Keep data encrypted

5. **Incident Response**[58]

   We have developed a comprehensive incident response plan, bolstered by extensive logging mechanisms using services like Amazon CloudWatch and QuickSight. These tools enable us to rapidly detect and respond to incidents, ensuring minimal impact on our operations. Our approach to logging goes beyond merely recording events; it is an integral part of our strategy for quickly understanding and addressing potential security issues, leveraging the advanced capabilities of Amazon CloudWatch and QuickSight for real-time analysis and insights.

   TR Satisfied - TR7.3 Implement logging

6. **Failure Management**[59]

   We've implemented a robust disaster recovery strategy, essential for managing and recovering from system failures. This strategy includes regularly backing up our data to alternate S3 and DynamoDB locations and rigorously testing our recovery procedures. Such practices underscore our dedication to providing a reliable and resilient cloud service.

   TR Satisfied - TR8.5 Enable disaster management and recovery

7. **Foundations**[60]

   We have laid strong foundations for reliability by implementing sophisticated workload monitoring and predictive scaling. These practices ensure we can dynamically adapt our resources to meet the demands of our workload, preventing over-provisioning and undersupply, thus maintaining our system's reliability even as demand fluctuates.

   TR Satisfied - TR8.2 Setup workload monitoring and prediction

8. **Monitoring**[61]

   We've strategically utilized AWS QuickSight and Grafana to enhance our monitoring capabilities. QuickSight allows us to create intuitive and interactive data visualizations, crucial for understanding our workload and making informed decisions. Grafana, on the other hand, complements this by offering in-depth analysis and observability. This combination empowers us to not only monitor our resources efficiently but also to transform vast datasets into actionable insights, ensuring that our resources are optimized for both performance and cost

   TR Satisfied - TR5.1 Enable monitoring and generate analytics

9. **Expenditure Awareness**[62]

   We maintain a high level of expenditure awareness by employing AWS budgeting alerts. This vigilant approach to cost management ensures that we are always informed about our spending and can make timely adjustments to stay aligned with our financial objectives.

   TR Satisfied - TR1.4 Budgeting alerts

10. **Matching Supply & Demand**[63]

    We have expertly matched the supply of resources with demand by leveraging AWS Auto-Scaling. This ensures that we can seamlessly handle traffic spikes and lulls, maintaining optimal performance and cost-effectiveness.

    TR Satisfied - TR1.1 Implement Auto-Scaling

5.6 Trade offs revisited

1. **[BR1] Optimize costs vs. [BR8] High availability**: **[SELECTED: HIGH AVAILABILITY]**

   Cost optimization (BR1) is fundamental to operating sustainably, especially in a competitive market. It involves careful resource allocation and scaling to avoid unnecessary expenditure. However, the importance of high availability (BR8) cannot be understated, as it ensures service reliability and customer satisfaction.

   We have used AWS services such as AWS Auto Scaling, Amazon EC2, and Amazon Route 53 to strike a balance between cost optimization (BR1) and high availability (BR8). While AWS Auto Scaling dynamically adjusts resources to maintain optimal cost efficiency, strategies like deploying across multiple Availability Zones with Amazon EC2 and using Route 53 for DNS failover mechanisms increase availability. The trade-off generally favors high availability, as the drone delivery service is highly dependent on constant uptime to maintain service levels and customer trust. Costs are managed by scaling down resources during low demand and employing cost-effective services like Amazon S3 for storage, but the architecture is designed to prioritize availability even if it incurs additional costs. The rationale is that the cost of downtime, both in terms of direct revenue loss and reputational damage, can far outweigh the expenses incurred in maintaining redundant systems and resources.

2. **[BR6] Data security vs. [BR9] Delayed Data Collection**: **[SELECTED: DATA SECURITY]**

   Data security (BR6) is paramount, as it protects sensitive customer information and maintains trust. AWS KMS is used to ensure data encryption, while AWS IoT Core manages secure data transfer. On the other hand, timely data collection (BR9) is essential for operational efficiency and service responsiveness. AWS Lambda can process and synchronize data efficiently when connectivity is established.

   Data security is non-negotiable, particularly when handling sensitive customer information. Thus, even though drones may experience intermittent connectivity, the architecture ensures that any data collected is encrypted at rest using AWS KMS, and transferred securely using protocols supported by AWS IoT Core. Delayed data collection is managed by storing data locally on drones in a secure manner and synchronizing it with the cloud as soon as connectivity is restored. Security measures are meticulously implemented and monitored, emphasizing their supremacy in the trade-off. The justification is that any compromise on security could lead to legal and reputational repercussions that could jeopardize the entire service, whereas delays in data collection are often a tolerable inconvenience by comparison.

5.7 Discussion of an alternate design [Skipped]

6 Kubernetes experimentation

6.1 Experiment Design

In this experiment, we choose a different application for the demonstration used in this section. Before describing the experiment design for kubernetes, let us briefly describe the application that we are going to deploy and come up with a few relevant TRs from the same.

6.1.1 The Application

For the sake of simplicity, we deploy an open source application called NextCloud on AWS EKS cluster. Below are the technical requirements we will be working with during this experiment.

6.1.2 Technical Requirements

[TR1] Use of container orchestration

[TR2] Implement auto-scaling using kubernetes HPA.

[TR3] Average CPU Utilization of 50% should trigger auto-scaling

6.1.3 Components and Tools

In this experiment, we will be using the below components and tools:

- **NextCloud App:** Nextcloud is a self-hosted file-sync and cloud storage platform that gives us control over our data. It is a free and open-source alternative to popular cloud storage services like Dropbox and Google Drive. Nextcloud can be used to store and share files, sync files across devices, and collaborate on documents. [37]
- **Amazon EKS (Elastic Kubernetes Service):** The cloud platform for orchestrating and managing containerized applications.
- **Kubernetes:** Container orchestration tool to manage and deploy application containers.
- **Locust:** Open-source load testing tool to simulate user traffic and generate load.

6.1.4 Description

This experiment was done by leveraging the above mentioned components and tools. The objective of this experiment is to assess the scalability and performance of an application hosted on Amazon EKS using Kubernetes by simulating varying levels of user load with Locust.

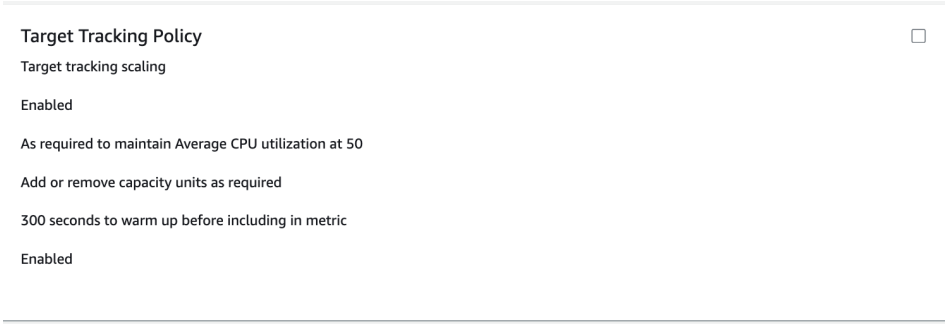
In this experiment, the services use a virtual IP address that acts as a single entry point to a group of pods providing the same service. When traffic reaches this service, it is then distributed to individual pods based on the selected load balancing strategy. In our case, we selected an IP-based round robin load balancing technique, where each new connection request is directed to the next available pod IP address in a cyclical manner. Kubernetes utilizes a distributed system approach for load balancing. The control plane components like the node server, scheduler, and controllers are distributed across nodes in the cluster. Additionally, load balancing occurs across multiple pods, which are distributed across different nodes in the cluster, enhancing scalability and fault tolerance.

We created a cluster with the default configurations as offered by AWS EKS. Once the cluster was active, we create a managed node group with initial count of 2 nodes, minimum of 2 and a maximum of 5 nodes. The instance types we chose were $t3.micro$ and $t3.medium$. So far as the pods are concerned, we started with conservative resource requests and limits for our pods, with CPU request as 0.2 and memory limit of 200Mi. We used the public docker image available for NextCloud and deployed the same on our cluster.

To conduct this experiment, we consider both the cases while deploying the application:

- Without HPA (Horizontal Pod Autoscaler) enabled.
- With HPA enabled.

For the second part of the experiment, we've set up both Cluster Autoscaling with managed NodeGroups and Horizontal Pod Autoscaling (HPA). Below is a glimpse of the node autoscaling policy:



Cluster Autoscaling dynamically manages the cluster's node count, responding to pod failures or rescheduling onto other nodes [Cluster Autoscaling]. For HPA, we've configured the CPU usage threshold at 50%. Additionally, we've specified a minimum of 1 pod and a maximum of 5 pods. To provision these resources, we utilized kubectl, Docker, and the AWS Console. So far as the load is concerned, we experiment with 2 different load generation strategies. Initially, we experiment by increasing the load linearly. Upon observing results for a couple of different rates, we experiment with an exponential increase in load. Some of the commmands used were as below:

- Update *kubectl* config to point to our cluster:

```
aws eks update-kubeconfig --name random-cluster --region=us-east-2
```

- Setup deployment, service, metrics service and autoscaler:

```
# Deployment
kubectl apply -f deployment.yaml

# Service
kubectl apply -f service.yaml

# Metrics Service
kubectl apply -f \
    https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

# HPA
kubectl autoscale deployment test-depl --cpu-percent=50 --min=1 --max=5
```

We used the below deloyment and service yaml files to setup the cluster:

```yaml
# deployment.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-depl
spec:
  replicas: 2
  selector:
    matchLabels:
      app: docker
  template:
    metadata:
      labels:
        app: docker
    spec:
      containers:
        - name: test-container
          image: library/nextcloud
          imagePullPolicy: Always
          ports:
            - containerPort: 80
            - containerPort: 443
          resources:
              limits:
                cpu: "0.4"
                memory: "400Mi"
            requests:
              cpu: "100m"
              memory: "100Mi"
```

```yaml
# service.yaml

apiVersion: v1
kind: Service
metadata:
  name: test-service
spec:
  selector:
    app: docker
  ports:
    - protocol: TCP
      name: http
      port: 80
      targetPort: 80
    - protocol: TCP
      name: https
      port: 443
      targetPort: 443

  type: LoadBalancer
```

6.2 Workload generation with Locust

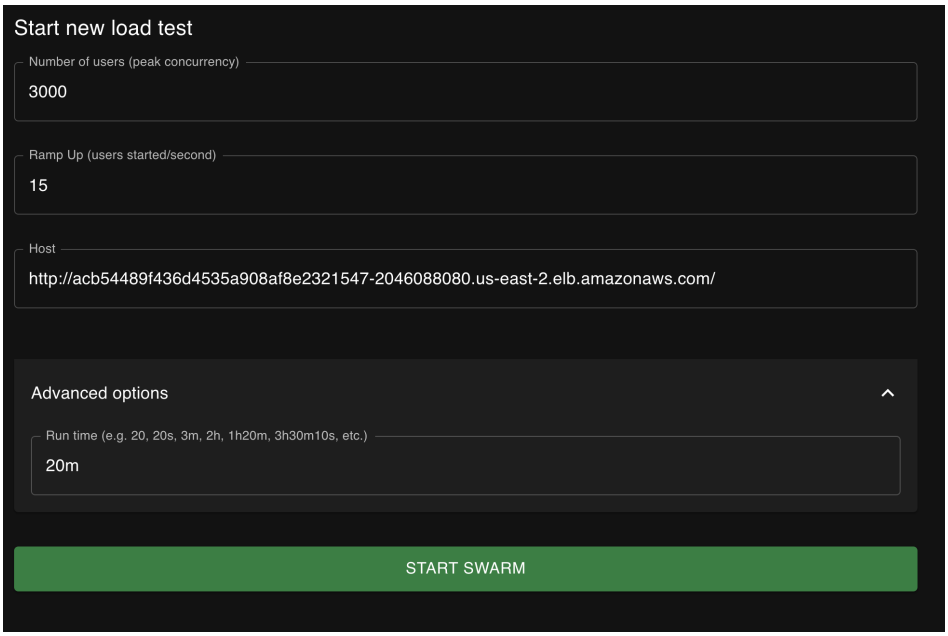
Locust is an open source load testing framework [38], that allows users to define user behaviour with Python code, and swarm our systems with millions of simultaneous users. They offer rich and informative documentation [39] and is minimalistic, so its really straightforward to work with.

The fake load was generated using locust. Below is a python code snippet for the same. We can see that we only try to hit the root endpoint '/' in the script for the sake of simplicity. The host name was provided in the command to run the locust, or there is an option to update the same in the locust UI as well.

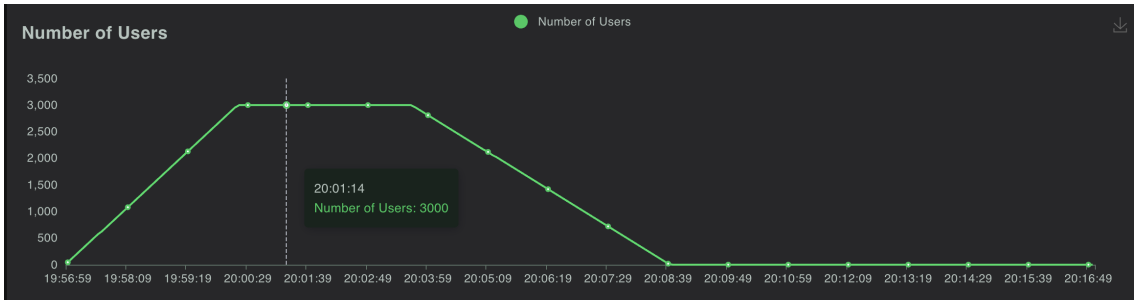
```python
# locustfile.py

from locust import HttpUser, task, between

class MyUser(HttpUser):
    wait_time = between(1, 3)

    @task
    def my_task(self):
        # Define your HTTP request here
        self.client.get("/")

# Running with a linear increase in user count
# locust -f locustfile.py -H LOAD_BALANCER_URL --modern-ui
```

6.2.1 Linear

Locust provides linear increase in load by default. The experiment was performed with the following configurations.



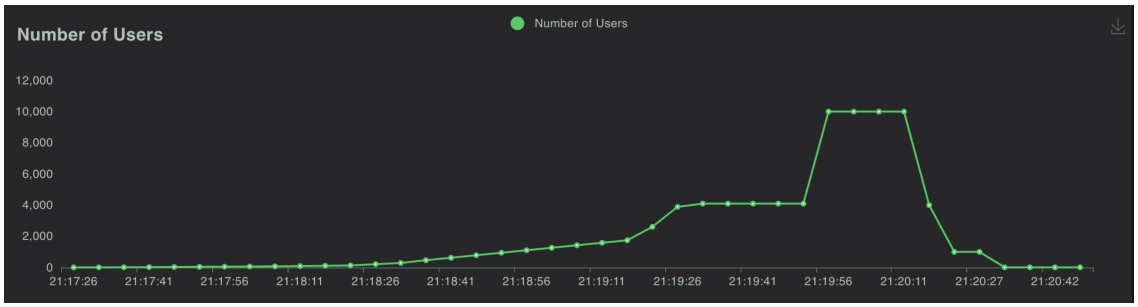
We can see in the test a linear increase/decrease in the behaviour as below:



We can also see a linear decrease. That is because we edited the configurations midway to be able to test the scaling down of the autoscaler. We can see that the peak users is 3000.

6.2.2 Exponential

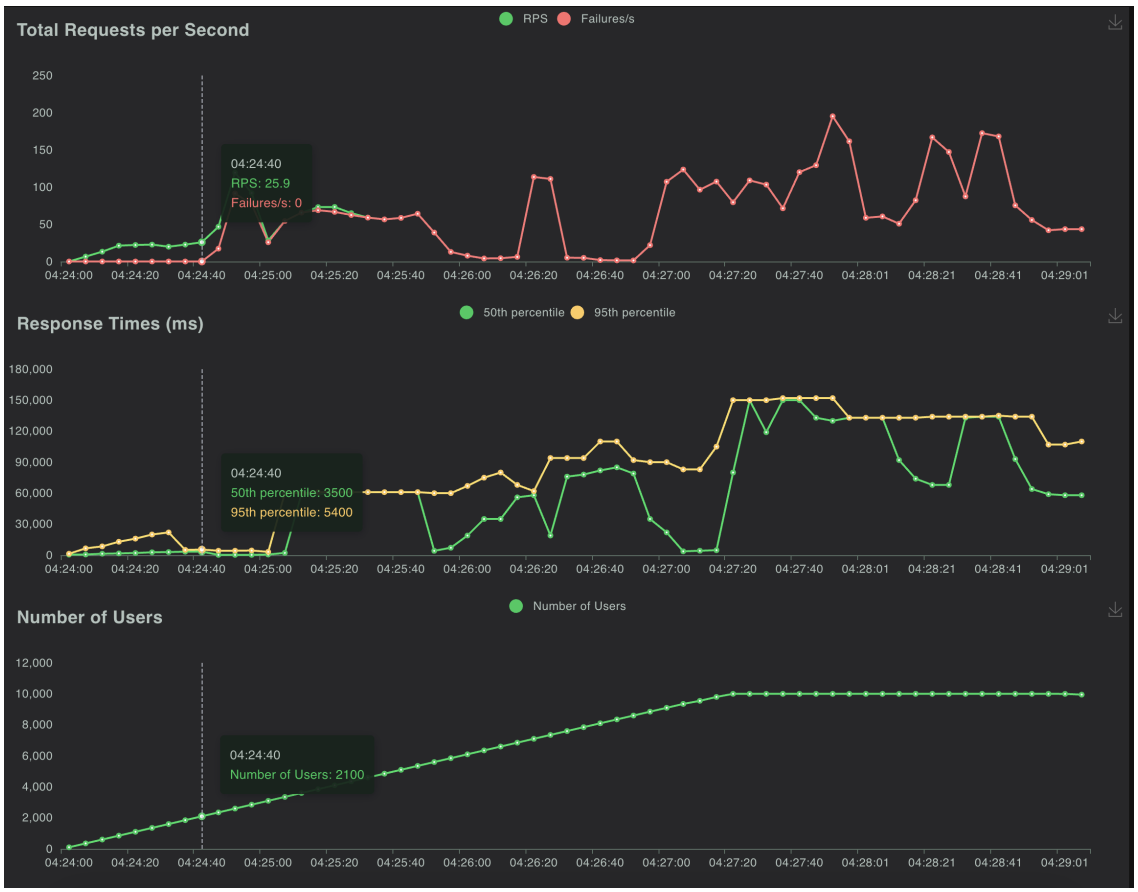
Upon looking for a couple of options to vary the load exponentially, we couldn't find much about the same. So, we decided to do an exponential change to the number of users manually. In the locust UI, we get an option to update the configuration in real time. We leveraed the same and varied the users by a factor of 2. We started out by 1 user, and kept increasing by a factor of 2 until a peak of 10000 concurrent users.



6.3 Analysis of the results

6.3.1 Experiment 1: Without Auto-Scaling

In this experiment, we tried to increase the load linearly using locust. Just to get an idea on how does our pods behave when they are not autoscaled, we did not configure the HPA. Below are the results.




```
~/Doc/Semester 3/C/kubernetes-experiment  main !3 ?7  kubectl get nodes --watch
0
NAME                                          STATUS   ROLES    AGE     VERSION
ip-172-31-12-35.us-east-2.compute.internal    Ready    <none>   4m29s   v1.28.3-eks-e71965b
ip-172-31-43-122.us-east-2.compute.internal   Ready    <none>   26m     v1.28.3-eks-e71965b
ip-172-31-12-35.us-east-2.compute.internal    Ready    <none>   6m38s   v1.28.3-eks-e71965b
ip-172-31-43-122.us-east-2.compute.internal   Ready    <none>   29m     v1.28.3-eks-e71965b
ip-172-31-12-35.us-east-2.compute.internal    Ready    <none>   11m     v1.28.3-eks-e71965b
ip-172-31-43-122.us-east-2.compute.internal   Ready    <none>   34m     v1.28.3-eks-e71965b
ip-172-31-12-35.us-east-2.compute.internal    Ready    <none>   16m     v1.28.3-eks-e71965b
ip-172-31-43-122.us-east-2.compute.internal   Ready    <none>   39m     v1.28.3-eks-e71965b
ip-172-31-12-35.us-east-2.compute.internal    Ready    <none>   21m     v1.28.3-eks-e71965b
ip-172-31-43-122.us-east-2.compute.internal   Ready    <none>   44m     v1.28.3-eks-e71965b
ip-172-31-12-35.us-east-2.compute.internal    Ready    <none>   27m     v1.28.3-eks-e71965b
ip-172-31-43-122.us-east-2.compute.internal   Ready    <none>   49m     v1.28.3-eks-e71965b
ip-172-31-12-35.us-east-2.compute.internal    Ready    <none>   32m     v1.28.3-eks-e71965b
ip-172-31-43-122.us-east-2.compute.internal   Ready    <none>   54m     v1.28.3-eks-e71965b
ip-172-31-12-35.us-east-2.compute.internal    Ready    <none>   37m     v1.28.3-eks-e71965b
ip-172-31-43-122.us-east-2.compute.internal   Ready    <none>   60m     v1.28.3-eks-e71965b

~/Doc/Semester 3/C/kubernetes-experiment  main !3 ?8
```

```
Local Terminal (1)

Last login: Sun Nov 19 14:44:33 on ttys000
~  kubectl get pods --watch
NAME                     READY   STATUS    RESTARTS   AGE
test-depl-f97fbd5c6-pnht8   1/1     Running   0          16m
```

As we can see in the above results, the app started giving a bunch of failures when the number of concurrent users reached 2100. Since there was no autoscaling enabled, the pod count remains just 1. We can also see that there are only 2 instances in the nodes list on top.

To try out a bit more with better instance, we gave a similar experiment a try with $t3.medium$ instances in the node group. Below are the results for the same. We can now see that the number of concurrent users where failures start kicking in increased from 2100 earlier to almost 6000 in this case.

This experiment satisfies the TR1 (mentioned above). Even though auto-scaling is not implemented, we have used kubernetes to deploy the app in this experiment.

6.3.2 Experiment 2: With Auto-Scaling (Linear Load)

In this experiment, we setup the Node Autoscaling along with a Horizontal Pod Autoscaler. We used a linear increase and decrease in the load using locust for the same. Below are the results for the same. Because there was autoscaling enabled, we experimented with *t3.micro* instace to avoid costs. In this case, we could see the number of pods increasing and decreasing linearly.

```
~/Doc/Semester 3/C/kubernetes-experiment  main !3 ?9   kubectl get deployment --watch
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
test-depl    2/2     2            2           6m11s
test-depl    2/3     2            2           8m14s
test-depl    2/3     2            2           8m14s
test-depl    2/3     2            2           8m14s
test-depl    2/3     3            2           8m14s
test-depl    3/3     3            3           8m16s
test-depl    3/5     3            3           8m29s
test-depl    3/5     3            3           8m29s
test-depl    3/5     3            3           8m29s
test-depl    3/5     5            3           8m29s
test-depl    4/5     5            4           8m31s
test-depl    5/5     5            5           8m31s
test-depl    5/2     5            5           25m
test-depl    5/2     5            5           25m
test-depl    2/2     2            2           25m
test-depl    2/1     2            2           25m
test-depl    2/1     2            2           25m
test-depl    1/1     1            1           25m
```

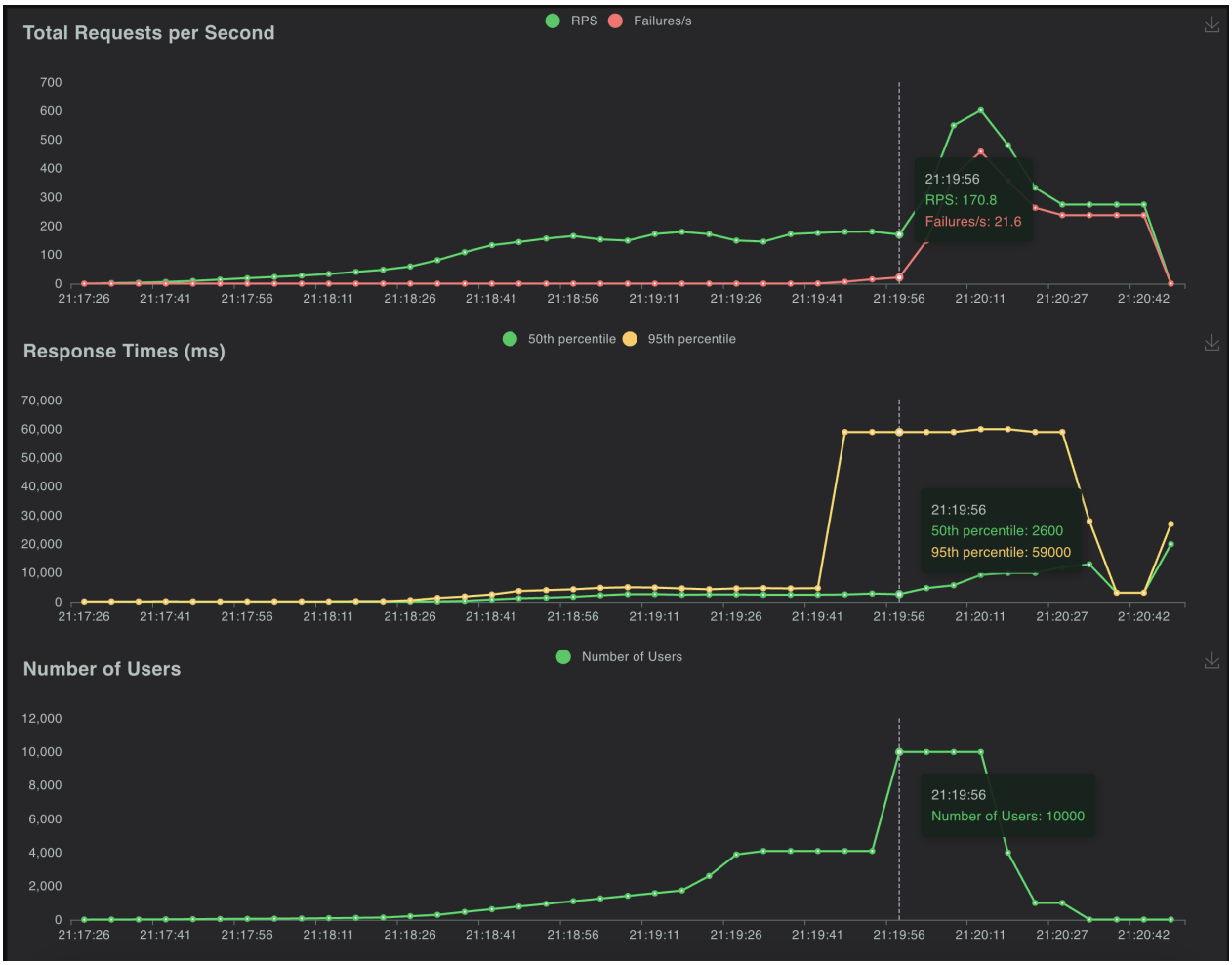
We can see some initial failures that happen while auto scaling is happening. It takes a few moments to spawn up new pods / containers. The auto scaling still triggers failures as we approach the peak (after 2880 concurrent users). But as we keep decreasing the load, it manages to scale down and satisfy a lot of requests. Overall the failure % is less compared to the case without any auto scaling. We can see that this satisfies the TR2.

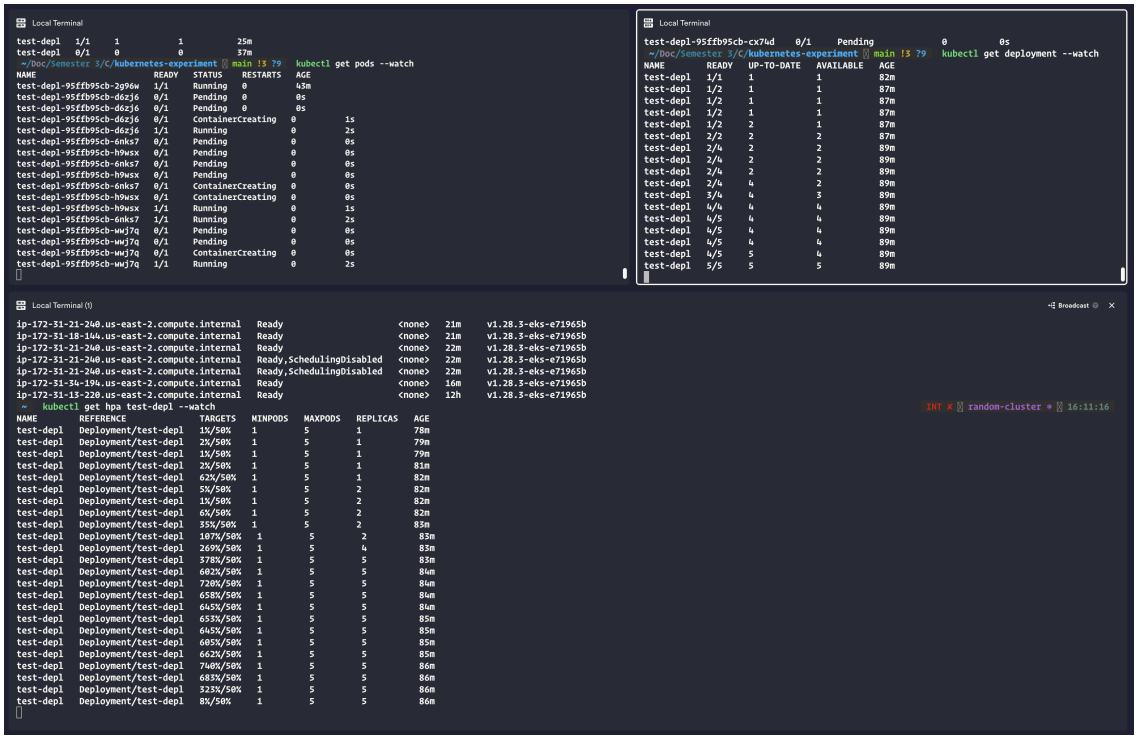
6.3.3 Experiment 3: With Auto-Scaling (Exponential Load)

In this case, we experiment by manually increasing the load exponentially. Because the load is exponentially increasing, we choose to user $t3.medium$ instances as opposed to $t3.micro$ instance in the previous experiment. We can then compare the results of this one with experiment 1 conducted with similar instances but without auto scaling. We can see that with auto scaling enabled on $t3.medium$ instances, we can handle almost 10000 concurrent users with very minimal failures. (As opposed to about 3000 users without autoscaling). We can also see the autoscaler in action and events, quickly adjusting to load in the figures below.

Hence, we observed the best performance with autoscaler enabled running on Autoscaling Node Group with $t3.medium$ instances that could handle exponentially increasing load really well. We can see that this satisfies the TR2. We have enabled auto scaling using HPA and seen that even with exponential increase in load, $t3.medium$ instances are able to handle load really well.

Below are some screenshots of the results.





Warning	FailedGetScale	an hour ago	horizontal-pod-autoscaler	deployments/scale.apps "test-depl" not found
Normal	ScalingReplicaSet	an hour ago	deployment-controller	Scaled up replica set test-depl-86f7f555c5 to 2
Warning	FailedGetResourceMetric	an hour ago	horizontal-pod-autoscaler	failed to get cpu utilization: missing request for cpu in container test-container of Pod test-depl-86f7f555c5-pcgpm
Warning	FailedComputeMetricsReplicas	an hour ago	horizontal-pod-autoscaler	invalid metrics (1 invalid out of 1), first error is: failed to get cpu resource metric value: failed to get cpu utilization: missing request for cpu in container test-container of Pod test-de
Normal	ScalingReplicaSet	an hour ago	deployment-controller	Scaled up replica set test-depl-95ffb95cb to 1
Normal	ScalingReplicaSet	an hour ago	deployment-controller	Scaled down replica set test-depl-86f7f555c5 to 1 from 2
Normal	ScalingReplicaSet	17 minutes ago	deployment-controller	Scaled up replica set test-depl-95ffb95cb to 2 from 1
Normal	ScalingReplicaSet	an hour ago	deployment-controller	Scaled down replica set test-depl-86f7f555c5 to 0 from 1
Normal	SuccessfulRescale	an hour ago	horizontal-pod-autoscaler	New size: 3; reason: cpu resource utilization (percentage of request) above target
Normal	ScalingReplicaSet	an hour ago	deployment-controller	Scaled up replica set test-depl-95ffb95cb to 3 from 2
Normal	SuccessfulRescale	an hour ago	horizontal-pod-autoscaler	New size: 5; reason: cpu resource utilization (percentage of request) above target
Normal	ScalingReplicaSet	an hour ago	deployment-controller	Scaled up replica set test-depl-95ffb95cb to 5 from 3
Normal	SuccessfulRescale	10 minutes ago	horizontal-pod-autoscaler	New size: 2; reason: All metrics below target
Normal	ScalingReplicaSet	30 minutes ago	deployment-controller	Scaled down replica set test-depl-95ffb95cb to 2 from 5
Normal	SuccessfulRescale	9 minutes ago	horizontal-pod-autoscaler	New size: 1; reason: All metrics below target
Normal	ScalingReplicaSet	9 minutes ago	deployment-controller	Scaled down replica set test-depl-95ffb95cb to 1 from 2
Warning	FailedGetResourceMetric	17 minutes ago	horizontal-pod-autoscaler	failed to get cpu utilization: unable to get metrics for resource cpu: unable to fetch metrics from resource metrics API: the server is currently unable to handle the request (get pods.
Warning	FailedComputeMetricsReplicas	17 minutes ago	horizontal-pod-autoscaler	invalid metrics (1 invalid out of 1), first error is: failed to get cpu resource metric value: failed to get cpu utilization: unable to get metrics for resource cpu: unable to fetch metrics fr
Normal	SuccessfulRescale	17 minutes ago	horizontal-pod-autoscaler	New size: 2; reason: cpu resource utilization (percentage of request) above target
Warning	FailedGetResourceMetric	16 minutes ago	horizontal-pod-autoscaler	failed to get cpu utilization: unable to get metrics for resource cpu: no metrics returned from resource metrics API
Warning	FailedComputeMetricsReplicas	16 minutes ago	horizontal-pod-autoscaler	invalid metrics (1 invalid out of 1), first error is: failed to get cpu resource metric value: failed to get cpu utilization: unable to get metrics for resource cpu: no metrics returned from
Normal	SuccessfulRescale	16 minutes ago	horizontal-pod-autoscaler	New size: 4; reason: cpu resource utilization (percentage of request) above target
Normal	ScalingReplicaSet	16 minutes ago	deployment-controller	Scaled up replica set test-depl-95ffb95cb to 4 from 2
Normal	SuccessfulRescale	15 minutes ago	horizontal-pod-autoscaler	New size: 5; reason: All metrics below target
Normal	ScalingReplicaSet	15 minutes ago	deployment-controller	Scaled up replica set test-depl-95ffb95cb to 5 from 4
Normal	SuccessfulRescale	11 minutes ago	horizontal-pod-autoscaler	New size: 4; reason: All metrics below target
Normal	ScalingReplicaSet	11 minutes ago	deployment-controller	Scaled down replica set test-depl-95ffb95cb to 4 from 5
Normal	ScalingReplicaSet	10 minutes ago	deployment-controller	Scaled down replica set test-depl-95ffb95cb to 2 from 4

7 Ansible playbooks [Skipped]

8 Demonstration [Skipped]

9 Comparisons [Skipped]

10 Conclusion

This cloud architecture project was a complex and challenging endeavor that required careful planning, critical decision-making, and continuous iteration. The project began by clearly defining the problem that needed to be solved, taking into account both business and technical requirements. Choosing a cloud provider was a critical decision that had a significant impact on the project's success. The provider was selected based on a set of predefined criteria and a detailed comparison of different options. The chosen provider not only met the project's requirements but also offered a suite of services that were essential for the project's success.

The project's design evolved over time, from a simple initial design to a complex and detailed blueprint. The design was based on the AWS Well-Architected Framework and incorporated best practices for scalability, security, and reliability. We also experimented with Kubernetes to validate the scalability an application. These experiments proved how autoscaling (pod and node combined) could handle varying loads effectively, which is important for real-world implementation.

In conclusion, this project was a valuable learning experience that taught us a lot about designing cloud architectures. We understood the importance of using structured methodologies, validating designs thoroughly, and considering trade-offs carefully. The lessons learned from this project will be valuable for future projects. Some of them are listed below:

10.1 The lessons learned

1. **Define the problem clearly:** Before starting any cloud architecture project, it is important to clearly define the problem that needs to be solved. This will help to ensure that the project is focused on the right requirements and that the chosen solution is the best fit for the problem.
2. **Choose a cloud provider carefully:** The selection of a cloud provider is a critical decision that can have a significant impact on the project's success. It is important to carefully evaluate different providers and select one that meets the project's requirements and offers the services that are needed.
3. **Use a structured methodology:** There are a number of different structured methodologies that can be used to design cloud architectures. Using a structured methodology will help to ensure that the design is well-thought-out and that all requirements are met.
4. **Validate designs thoroughly:** It is important to validate cloud architecture designs thoroughly to ensure that they are scalable, secure, and reliable. This can be done through a variety of methods, such as testing, modeling, and simulation.
5. **Consider trade-offs carefully:** There are often tradeoffs that need to be considered when designing cloud architectures. For example, there is often a trade-off between scalability and cost. It is important to carefully consider these tradeoffs and make decisions that are in the best interests of the organization.
6. **Use the AWS Well-Architected Framework:** The AWS Well-Architected Framework is a valuable resource for designing cloud architectures. The framework provides a set of best practices that can help to ensure that architectures are scalable, secure, efficient, cost-effective, and reliable.
7. **Experiment with Kubernetes:** Kubernetes is a container orchestration platform that can be used to manage containerized applications in the cloud. Experimenting with Kubernetes can help to validate the scalability of a cloud architecture.
8. **Continuously assess and improve:** Cloud architectures are not static. They need to be continuously assessed and improved to meet the changing needs of the organization.
9. **Document the architecture:** It is important to document the cloud architecture so that it can be easily understood and maintained.
10. **Learn from mistakes:** Everyone makes mistakes, and this is especially true when designing cloud architectures. It is important to learn from mistakes and make sure that they are not repeated in future projects.

10.2 Possible continuation of the project [Skipped]

References

[1] Cost Optimization with AWS Auto Scaling

[2] Health Checks

[3] Budgeting Alerts

[4] Designing architectures for multi-tenancy

[5] RBAC in Cloud

[6] MQTT Is Important for Edge-to-cloud Connectivity

[7] Cloud Logging

[8] Secure Aerial Data Delivery with Lightweight Encryption

[9] Auto-Scaling for High Availability

[10] Rate Limiting and DOS Atttack

[11] High Availability and Disaster Recovery Plan for Version Control System

[12] Video on Regular Health Checks To Ensure High Availability

[13] multi-region architecture: The key to high availability risk mitigation

[14] Delay-reliability-aware protocol adaption

[15] Amazon Quicksight Documentation

[16] Amazon Managed Grafana Documentation

[17] Amazon S3 Documentation

[18] Amazon CloudFront Documentation

[19] Amazon API Gateway Documentation

[20] AWS WAF Documentation

[21] Amazon ECS Documentation

[22] AWS Lambda Documentation

[23] AWS Application Load Balancer Documentation

[24] AWS KMS Documentation

[25] Amazon DynamoDB Documentation

[26] Amazon Cloudwatch Documentation

[27] Amazon ElastiCache Documentation

[28] AWS IoT Core Documentation

[29] AWS Route 53 Documentation

[30] AWS Cost Explorer Documentation

[31] AWS Budgets Documentation

[32] Amazon Kinesis Documentation

[33] Amazon EC2 Documentation

[34] AWS Auto Scaling Documentation

[35] AWS Identity and Access Managemen Documentation

[36] What are Microservices?

[37] NextCloud Home Page

[38] Locust Official Website

[39] Locust Documentation

[40] AWS Lambda - Event-driven invocation

[41] AWS IAM Documentation

[42] AWS Simple Email Service (SES) Documentation

[43] AWS Operational Excellence Pillar whitepaper

[44] AWS Security Pillar whitepaper

[45] AWS Reliability Pillar whitepaper

[46] AWS Performance Efficiency Pillar whitepaper

[47] AWS Cost Optimization Pillar whitepaper

[48] AWS Sustainability Pillar whitepaper

[49] Design Principle Operational excellence

[50] Design Principle Reliability

[51] Design Principle Cost

[52] Best Practices Prepare

[53] Best Practices Operate

[54] Best Practices Evolve

[55] Best Practices Iam

[56] Best Practices Infrastructure protection

[57] Best Practices Data protection

[58] Best Practices Incident response

[59] Best Practices Failure management

[60] Best Practices Foundations

[61] Best Practices monitoring

[62] Best Practices Expenditure and usage awareness

[63] Best Practices Manage demand and supply resources