NANYANG TECHNOLOGICAL UNIVERSITY

ENHANCING SPEECH RECOGNITION SCALABILITY AND RESILIENCE THROUGH DECOUPLED ARCHITECTURE

Tey Li Zhang Edmund

College of Computing and Data Science

2025

NANYANG TECHNOLOGICAL UNIVERSITY

CCDS24-0015

ENHANCING SPEECH RECOGNITION SCALABILITY AND RESILIENCE THROUGH DECOUPLED ARCHITECTURE

Submitted in Partial Fulfilment of the Requirements
for the Degree of Bachelor of Computing in Computer Science
of the Nanyang Technological University

by

Tey Li Zhang Edmund

College of Computing and Data Science
2025

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi lobortis id nunc a maximus. Nam vitae pellentesque elit. Ut lacus arcu, consectetur at erat sed, sollicitudin suscipit dolor. Sed pharetra, ipsum id volutpat dictum, lacus ante hendrerit mauris, ac tristique orci dui quis dolor. Suspendisse dictum magna vitae fermentum maximus. Quisque vulputate urna id turpis pulvinar suscipit. Praesent sit amet finibus ante. Nullam dignissim sapien ac dolor elementum accumsan. In fermentum tellus eu velit ullamcorper pellentesque id eu eros. Phasellus erat elit, luctus eleifend urna eu, consequat fringilla nisl. Aenean convallis mattis libero, eu dignissim ligula mollis at. Cras pharetra imperdiet risus, in placerat sem ultrices ultrices. Donec eget augue condimentum, efficitur nunc vel, fermentum lacus.

Acknowledgments

The writer uses this section to thank all those he or she is indebted for guidance, financial or any other assistance rendered during the course of the project.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Automatic Speech Recognition (ASR) systems, which convert human speech into text, have become integral to modern voice-driven technologies. While current commercial ASR solutions excel in single-language environments, they often struggle with multilingual scenarios, due to inter- and intra-sentence language variety [1]. The research team at Nanyang Technological University (NTU) Speech Lab addresses this challenge through their innovative multilingual ASR model, capable of transcribing speech in English, Malay, Mandarin, and Singlish [2, 3]. This development is particularly significant in Singapore's context, where code-switching between languages and dialects is commonplace in daily communication.

One prominent user of this ASR system is the Singapore Civil Defence Force (SCDF), which leverages the live transcription service for their emergency call centers [4]. The transcription system enables officers to record key information efficiently, saving critical time during emergencies. Consequently, the availability and reliability of the system are paramount, as any disruption could impede communication in life-saving scenarios.

The ASR system is currently deployed on Kubernetes across Microsoft Azure and Amazon Web Services (AWS) cloud platforms. Users interact with the system by establishing a WebSocket connection to the server, selecting their desired transcription model. The server assigns a worker to handle the task by forwarding audio data to the worker for transcription. The resulting transcript is then sent back to the user.

1.2 Importance

Previous Final Year Project (FYP) students have contributed to various aspects of this ASR system, such as deploying it on AWS with Terraform [5, 6] and enhancing its security [7]. However, significant limitations persist. The direct communication between the server and workers results in a tightly coupled architecture, making it challenging to scale these components independently to meet fluctuating workloads [8]. Additionally, both the server and workers maintain state information, such as audio data and worker statuses, exacerbating the scalability challenge.

System reliability is particularly affected by worker failures, which can occur due to various factors including resource exhaustion, node crashes, or network disruptions. These failures result in unprocessed audio segments, creating gaps in transcription that significantly impact service continuity. Such limitations directly compromise the system's ability to meet its Service Level Objectives (SLOs), particularly in terms of latency and availability, potentially affecting critical operations of its users.

1.3 Objective, Scope, and Significance

The objective of this FYP is to enhance the scalability and availability of the ASR system by transitioning to a decoupled architecture. To accomplish this, the project scope includes redesigning the system architecture to decouple components using a message queue; developing a dynamic and predictive scaling policy for workers; and designing mechanisms to minimize latency during worker failures. The modifications to the underlying ASR model, security aspects of the application and monitoring will be out-of-scope for this project.

The significance of this research lies in its potential to enhance the system's fault tolerance and ability to handle fluctuating traffic loads. By addressing these limitations, the project aims to provide improved support for NTU Speech Lab's users, ensuring the reliability of the ASR service for critical applications such as SCDF's emergency operations.

1.4 Report Organisation

This report is structured into five chapters, each focusing on a specific aspect of the project:

- **Chapter 1: Introduction** - This chapter provides an overview of the project, including its background, importance, objectives, scope, and significance.
- **Chapter 2: Literature Review** - This chapter reviews previous FYP works on the ASR system and introduces the relevant technologies used in the project.
- **Chapter 3: Analysis and Design** - This chapter presents an analysis of the current ASR system architecture and identifies its limitations. It then describes the proposed decoupled architecture, including the use of message queues and dynamic scaling policies.
- **Chapter 4: Detailed Implementation** - This chapter provides a detailed implementation of the new architecture, the development of the dynamic scaling policy, and the mechanisms implemented to handle worker failures.
- **Chapter 5: Conclusion and Future Work** - This chapter summarizes the contributions of the project. It also outlines potential areas for future research and development to further improve the ASR system.

Chapter 2

Literature Review

The development of reliable and scalable Automatic Speech Recognition (ASR) systems requires an understanding of modern distributed systems technologies and architectural design. This literature review examines the technologies and approaches relevant to enhancing ASR system scalability and resilience. The review begins with previous work on the system, followed by an analysis of key technologies in distributed systems, containerization, message queues, cloud infrastructure, and chaos engineering.

2.1 Previous Work

Putra [7] had worked on the same ASR system and his project effectively highlights the advantages of transitioning from a tightly coupled ASR system to a decoupled microservices architecture using Apache Kafka. The use of Kubernetes and other modern cloud-native tools such as Kyverno, Falco, and Knative demonstrates a robust effort to address scalability, reliability, and security challenges in ASR systems. The project discusses the integration of Kafka as the message broker to decouple the master and worker components, ensuring fault tolerance and enabling the system to recover from worker crashes without losing data.

2.1.1 Research Gap

A significant research gap exists in the implementation's choice of message broker technology. The selection of Kafka for the ASR system raises concerns regarding its fit for scenarios requiring high data accuracy and context preservation. While Kafka's high throughput and durability are valuable for many systems, ASR workflows are fundamentally bottlenecked by the processing speed of workers, not the message broker. This mismatch between technology choice and system requirements suggests that RabbitMQ would be a more suitable alternative due to its task-oriented features and built-in support for state-dependent processing, which better aligns with the sequential nature of speech processing tasks.

Furthermore, while Putra's work [7] demonstrated promising results, a practical limitation emerged as the research team no longer has access to his project's codebase. This circumstance has created an opportunity to revisit the system's architecture with fresh perspective, particularly in the areas of component decoupling and scaling mechanisms. The current project therefore aims to not only address the technological fit of the message broker but also to establish a well-documented implementation that can be maintained and evolved by the research team.

2.2 Distributed Systems Architecture

2.2.1 Evolution of Microservices

The transition from monolithic to microservices architecture represents a fundamental shift in distributed systems design. Newman [9] defines microservices as small, autonomous services that work together, focusing on modularity and independent deployability. This architectural style has gained prominence due to its ability to support scalability, maintainability, and team autonomy [10]. In the context of ASR systems, microservices architecture enables independent scaling of components and improved fault isolation.

2.2.2 gRPC in Modern Applications

gRPC is a high-performance Remote Procedure Call (RPC) [11], which led to a significant advancement in service-to-service communication. Niswar et al. [12] conducted performance analyses showing that gRPC outperforms REST APIs and GraphQL in terms of response time for fetching both flat and nested data, as well as CPU utilisation. It utilises Protocol Buffers which provide a language-agnostic interface definition [13], allowing services written in different programming languages to communicate efficiently. This is crucial in microservices architectures where different teams might develop services using different technologies. These features make gRPC particularly suitable for ASR systems where reliable, high-performance communication between components is essential.

2.3 Containerization

2.3.1 Docker

Docker is a service that leverages on operating system level virtualisation to package software into containers [14]. Bernstein [15] explains how Docker containers package applications with their dependencies. This consistency is crucial for ASR systems, where complex model and service dependencies must be managed effectively.

2.3.2 Kubernetes

Kubernetes is a container orchestration tool used to automate the deployment and management of containers [16]. Burns et al. [17] detail its architecture and ability to manage containerized applications at scale. For ASR systems, Kubernetes provides essential features such as automatic scaling, self-healing, and rolling updates [18], which are crucial for maintaining service reliability.

2.3.3 Helm

Helm is a package manager for Kubernetes applications [19]. Helm utilises Charts, as reusable packages that contains pre-configured Kubernetes resources [20], making complex application deployments more manageable. These Charts function as templates that can be customized through value files [19], enabling environment-specific configurations while maintaining consistency in the underlying architecture.

2.4 Amazon Web Services (AWS)

AWS is a cloud service provider that offers a wide range of products and services. These services span compute, storage, networking, database, and container management, among others [21].

2.4.1 Virtual Private Cloud (VPC)

Amazon VPC forms the networking foundation for AWS resources, providing an isolated virtual network environment in the cloud [22]. It enables users to define network architecture with custom IP address ranges, subnets, and routing tables [22]. A key feature is its ability to span multiple Availability Zones (AZs) within a region, enhancing system resilience through geographical distribution [23].

2.4.2 Elastic Compute Cloud (EC2)

Amazon EC2 is a computing service that provides scalable virtual machines (instances) in the cloud [24]. It offers a wide range of instance types optimized for different use cases, from compute-intensive applications to memory-intensive workloads [25]. EC2 instances can be launched across multiple AZs for high availability. EC2 pricing models including on-demand, reserved, and spot instances to optimize costs based on workload patterns [26].

2.4.3 Identity and Access Management (IAM)

IAM provides fine-grained access control to AWS resources [27]. It implements the principle of least privilege through a comprehensive policy framework that defines who (principal) can do what (actions) on which resources under specific conditions [28]. IAM enables organizations to manage user identities, roles, and permissions centrally, ensuring secure access to cloud resources while maintaining compliance requirements.

2.4.4 Elastic Kubernetes Service (EKS)

Amazon EKS is a managed Kubernetes service that simplifies the deployment, management, and scaling of containerized applications [29]. It automatically manages the availability and scalability of the Kubernetes control plane across multiple AZs [29]. EKS integrates seamlessly with other AWS services and supports various deployment models, including hybrid architectures that span cloud and on-premises environments [30].

2.4.5 Elastic Container Registry (ECR)

Amazon ECR is a managed container registry service that simplifies the storage, management, and deployment of container images [31]. It provides encrypted image storage and integrates with AWS IAM for access control [32]. ECR features automatic image scanning for vulnerabilities [33] and lifecycle policies for image management [34], making it an essential component in container-based architectures.

2.4.6 Elastic File System (EFS)

Amazon EFS provides scalable, fully managed network file storage for use with AWS cloud services and on-premises resources [35]. Supporting the Network File System version 4 (NFSv4) protocol [36], EFS can be accessed concurrently by thousands of compute instances [37]. It automatically scales throughput when files are added or removed [37], making it ideal for applications requiring shared file access across multiple instances or containers.

2.5 Infrastructure-as-Code (IaC)

IaC is an approach to managing and provisioning computing infrastructure through configuration files, rather than through physical hardware configuration or interactive configuration tools. This method allows for the automation of infrastructure setup, ensuring consistency and reducing the risk of human error [38].

2.5.1 Terraform

Terraform is an IaC tool that allows for the declarative management of cloud resources [39]. This means that we define the desired final state of our architecture, and Terraform will apply changes only when necessary to achieve that state [40]. Unlike traditional manual deployment through cloud provider consoles (often referred to as ”ClickOps” [41]), Terraform enables organizations to define their infrastructure using declarative configuration files. This code-driven approach transforms infrastructure deployment from a manual, error-prone process into an automated, version-controlled workflow.

Terraform enables teams to consistently replicate environments across development, testing, and production stages [38], ensuring that infrastructure configurations remain identical at each phase. By maintaining infrastructure as code, teams can version control their changes, enabling peer reviews and the ability to roll back modifications if issues arise. Terraform also manages dependencies between different cloud resources automatically [42], reducing the complexity of infrastructure deployment.

2.6 In-memory Data Storage

2.6.1 Redis

Redis is a high-performance, in-memory data store commonly used as a cache and a key-value database [43]. Redis is fast, and thus well-suited for use in ASR systems, where it can store session information and temporary transcription data. This enables rapid access to frequently used data and facilitates reliable state management, ensuring smooth and responsive system performance.

2.7 Message Queues

Message queues enable asynchronous communication between services in distributed systems, providing temporary message storage and reliable delivery mechanisms [44]. This asynchronous pattern facilitates decoupling of producers and consumers [45], allowing components to scale independently and operate without direct dependencies on each other.

2.7.1 Apache Kafka

Apache Kafka is designed as a distributed log-based messaging system, and its main use case is for ingesting and streaming real-time data [46]. Kafka's architecture centers around append-only logs (topics) divided into partitions, where messages are immutably stored and accessed via offset-based positioning [47].

2.7.2 RabbitMQ

RabbitMQ implements the Advanced Message Queuing Protocol (AMQP) [48] and operates as a broker-based message queue [49]. It provides sophisticated message routing capabilities through exchanges and queues, supporting various patterns including publish-subscribe, and request-reply communications [50]. RabbitMQ offers features such as message acknowledgments, dead letter queues, and priority queuing [50], making it particularly suitable for complex message routing requirements.

2.7.3 Comparison of Kafka and RabbitMQ

While both systems are robust message brokers, their architectural differences significantly impact their suitability for ASR applications. Below, we compare Kafka and RabbitMQ across key dimensions relevant to ASR systems.

Message Ordering

Kafka's partitioned architecture, while enabling high throughput, cannot guarantee message ordering across partitions. Dobbelaere and Esmaili [51] note that Kafka's ordering guarantees are limited to individual partitions.

RabbitMQ, through its queue-based architecture, maintains strict FIFO (First-In-First-Out) ordering within queues in the same channel [51], making it more suitable for ASR systems where speech context and sequence are crucial.

Message Delivery Guarantees

Kafka provides at-least-once delivery semantics through offset management [51], but consumers must handle offset commits carefully to avoid message reprocessing.

RabbitMQ's acknowledgment mechanism offers more flexible delivery guarantees, with built-in support for message acknowledgment [51] and automatic requeuing of unprocessed messages [52].

Processing Requirements

For ASR systems, where maintaining speech context is paramount [53], RabbitMQ's single-queue consumer model aligns better with the need for sequential processing.

The research shows that while Kafka's throughput advantage is significant for high-volume streaming [51], this benefit is less relevant for ASR workloads where processing speed is typically bounded by the speech recognition models rather than message throughput.

Message Queue Selection

Based on these considerations and supported by Dobbelaere and Esmaili's [51] findings, RabbitMQ emerges as the more appropriate choice for ASR systems due to its strong message ordering guarantees, flexible acknowledgement mechanisms, and support for sequential processing requirements. By leveraging RabbitMQ's features, ASR systems can ensure accurate transcription results and maintain the context of speech data throughout the processing pipeline.

2.8 Chaos Engineering

Chaos engineering is an approach to deliberately introducing failures to identify weaknesses and improve resilience [54]. By simulating conditions like Kubernetes pod failure, network delay, and node stress [55], chaos engineering helps us observe system behaviour under stress and improve system robustness [54]. It also enhances incident response time by enabling a better understanding of failure scenarios [54]. Some chaos engineering tools include Chaos Mesh [56] and AWS Fault Injection Simulator (FIS) [57].

Chapter 3

Analysis and Design Approach

Chapter 4

Detailed Implementation

4.1 Background

Motivation

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi lobortis id nunc a maximus. Nam vitae pellentesque elit. Ut lacus arcu, consectetur at erat sed, sollicitudin suscipit dolor. Sed pharetra, ipsum id volutpat dictum, lacus ante hendrerit mauris, ac tristique orci dui quis dolor. Suspendisse dictum magna vitae fermentum maximus. Quisque vulputate urna id turpis pulvinar suscipit. Praesent sit amet finibus ante. Nullam dignissim sapien ac dolor elementum accumsan. In fermentum tellus eu velit ullamcorper pellentesque id eu eros. Phasellus erat elit, luctus eleifend urna eu, consequat fringilla nisl. Aenean convallis mattis libero, eu dignissim ligula mollis at. Cras pharetra imperdiet risus, in placerat sem ultrices ultrices. Donec eget augue condimentum, efficitur nunc vel, fermentum lacus.

Cast	Actor / Actress
Harvey Specter	Gabriel Macht
Mike Ross	Patrick J Adams
Jessica Pearson	Gina Torres

Table 4.1: Casts

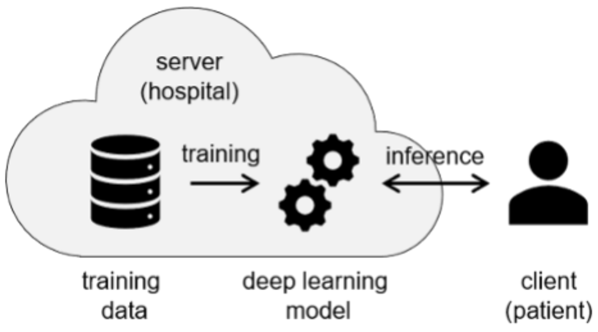

Figure 4.1: Image

Chapter 5

Conclusion and Future Work

5.1 Background

Motivation

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi lobortis id nunc a maximus. Nam vitae pellentesque elit. Ut lacus arcu, consectetur at erat sed, sollicitudin suscipit dolor. Sed pharetra, ipsum id volutpat dictum, lacus ante hendrerit mauris, ac tristique orci dui quis dolor. Suspendisse dictum magna vitae fermentum maximus. Quisque vulputate urna id turpis pulvinar suscipit. Praesent sit amet finibus ante. Nullam dignissim sapien ac dolor elementum accumsan. In fermentum tellus eu velit ullamcorper pellentesque id eu eros. Phasellus erat elit, luctus eleifend urna eu, consequat fringilla nisl. Aenean convallis mattis libero, eu dignissim ligula mollis at. Cras pharetra imperdiet risus, in placerat sem ultrices ultrices. Donec eget augue condimentum, efficitur nunc vel, fermentum lacus .

Cast	Actor / Actress
Harvey Specter	Gabriel Macht
Mike Ross	Patrick J Adams
Jessica Pearson	Gina Torres

Table 5.1: Casts

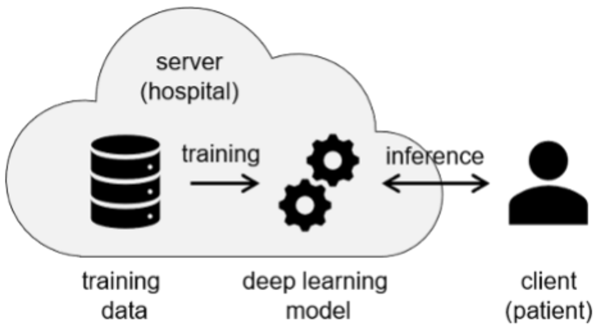

Figure 5.1: Image

Bibliography

[1] H. Liu, L. P. Garcia, X. Zhang, A. W. H. Khong and S. Khudanpur, *Enhancing code-switching speech recognition with interactive language biases*, 2023. arXiv: 2309.16953 [eess.AS]. [Online]. Available: https://arxiv.org/abs/2309.16953.

[2] AI Singapore, *Speech lab*. [Online]. Available: https://aisingapore.org/aiproducts/speech-lab/ Accessed: 21/01/2025.

[3] O. Chia, "Ai masters singlish in key breakthrough to serve healthcare and patients' needs," *The Straits Times*, 14th Nov. 2024. [Online]. Available: https://www.straitstimes.com/singapore/ai-masters-singlish-in-key-breakthrough-to-serve-healthcare-and-patients-needs Accessed: 21/01/2025.

[4] I. Liew, "Plan to use ai to help emergency call operators," *The Straits Times*, 10th Jul. 2018. [Online]. Available: https://www.straitstimes.com/singapore/plan-to-use-ai-to-help-emergency-call-operators Accessed: 21/01/2025.

[5] S. Yu, "Deploying speech recognition system using kubernetes cluster - infrastructure as code with terraform and terragrunt," B.Eng. dissertation, Nanyang Technol. Univ., Singapore, 2023. [Online]. Available: https://hdl.handle.net/10356/165869.

[6] K. S. Lee, "Deploying asr system for scalability and robustness on aws," B.Eng. dissertation, Nanyang Technol. Univ., Singapore, 2022. [Online]. Available: https://hdl.handle.net/10356/156701.

[7] T. Putra, “Deploying automatic speech recognition system for scalability, reliability, and security with kubernetes,” B.Eng. dissertation, Nanyang Technol. Univ., Singapore, 2023. [Online]. Available: `https://hdl.handle.net/10356/171933`.

[8] P. Kookarinrat and Y. Temtanapat, “Design and implementation of a decentralized message bus for microservices,” in *2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2016, pp. 1–6. DOI: `10.1109/JCSSE.2016.7748869`.

[9] S. Newman, *Building Microservices, 2nd Edition*. O’Reilly Media, 2021.

[10] L. De Lauretis, “From monolithic architecture to microservices architecture,” in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2019, pp. 93–96. DOI: `10.1109/ISSREW.2019.00050`.

[11] gRPC, *About grpc*. [Online]. Available: `https://grpc.io/about/` Accessed: 22/01/2025.

[12] M. Niswar, R. Safruddin, A. Bustamin and I. Aswad, “Performance evaluation of microservices communication with rest, graphql, and grpc,” *International Journal of Electronics and Telecommunications*, pp. 429–436, Jun. 2024. DOI: `10.24425/ijet.2024.149562`.

[13] Google LLC, *Protocol buffers*. [Online]. Available: `https://protobuf.dev/` Accessed: 22/01/2025.

[14] Amazon Web Services, *What’s the difference between docker and a vm?* [Online]. Available: `https://aws.amazon.com/compare/the-difference-between-docker-vm/` Accessed: 22/01/2025.

[15] D. Bernstein, “Containers and cloud: From lxc to docker to kubernetes,” *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014. DOI: `10.1109/MCC.2014.51`.

[16] Red Hat, *What is kubernetes?* [Online]. Available: `https://www.redhat.com/en/topics/containers/what-is-kubernetes` Accessed: 22/01/2025.

[17] B. Burns, B. Grant, D. Oppenheimer, E. Brewer and J. Wilkes, “Borg, omega, and kubernetes,” *ACM Queue*, vol. 14, pp. 70–93, 2016. [Online]. Available: `http://queue.acm.org/detail.cfm?id=2898444`.

[18] VMware, *What is kubernetes?* [Online]. Available: https://www.vmware.com/topics/kubernetes#kubernetes-features Accessed: 22/01/2025.

[19] Red Hat, *What is helm?* [Online]. Available: https://www.redhat.com/en/topics/devops/what-is-helm Accessed: 22/01/2025.

[20] Helm, *Charts*. [Online]. Available: https://helm.sh/docs/topics/charts/ Accessed: 22/01/2025.

[21] Amazon Web Services, *Aws cloud services*. [Online]. Available: https://aws.amazon.com/products/ Accessed: 22/01/2025.

[22] Amazon Web Services, *What is amazon vpc?* [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html Accessed: 22/01/2025.

[23] M. Haken, *Improving performance and reducing cost using availability zone affinity*. [Online]. Available: https://aws.amazon.com/blogs/architecture/improving-performance-and-reducing-cost-using-availability-zone-affinity/ Accessed: 22/01/2025.

[24] Amazon Web Services, *What is amazon ec2?* [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html Accessed: 22/01/2025.

[25] Amazon Web Services, *Amazon ec2 instance types*. [Online]. Available: https://aws.amazon.com/ec2/instance-types/ Accessed: 22/01/2025.

[26] Amazon Web Services, *Amazon ec2 on-demand pricing*. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/ Accessed: 22/01/2025.

[27] Amazon Web Services, *What is iam?* [Online]. Available: https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html Accessed: 22/01/2025.

[28] Amazon Web Services, *How iam works*. [Online]. Available: https://docs.aws.amazon.com/IAM/latest/UserGuide/intro-structure.html Accessed: 22/01/2025.

[29] Amazon Web Services, *What is amazon eks?* [Online]. Available: https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html Accessed: 22/01/2025.

[30] Amazon Web Services, *Deploy amazon eks clusters across cloud and on-premises environments*. [Online]. Available: `https://docs.aws.amazon.com/eks/latest/userguide/eks-deployment-options.html` Accessed: 22/01/2025.

[31] Amazon Web Services, *What is amazon elastic container registry?* [Online]. Available: `https://docs.aws.amazon.com/AmazonECR/latest/userguide/what-is-ecr.html` Accessed: 22/01/2025.

[32] Amazon Web Services, *Identity and access management for amazon elastic container registry*. [Online]. Available: `https://docs.aws.amazon.com/AmazonECR/latest/userguide/security-iam.html` Accessed: 22/01/2025.

[33] Amazon Web Services, *Scan images for software vulnerabilities in amazon ecr*. [Online]. Available: `https://docs.aws.amazon.com/AmazonECR/latest/userguide/image-scanning.html` Accessed: 22/01/2025.

[34] Amazon Web Services, *Automate the cleanup of images by using lifecycle policies in amazon ecr*. [Online]. Available: `https://docs.aws.amazon.com/AmazonECR/latest/userguide/LifecyclePolicies.html` Accessed: 22/01/2025.

[35] Amazon Web Services, *What is amazon elastic file system?* [Online]. Available: `https://docs.aws.amazon.com/efs/latest/ug/whatisefs.html` Accessed: 22/01/2025.

[36] Amazon Web Services, *How amazon efs works*. [Online]. Available: `https://docs.aws.amazon.com/efs/latest/ug/how-it-works.html` Accessed: 22/01/2025.

[37] Amazon Web Services, *Amazon efs performance*. [Online]. Available: `https://docs.aws.amazon.com/efs/latest/ug/performance.html` Accessed: 22/01/2025.

[38] Amazon Web Services, *What is infrastructure as code?* [Online]. Available: `https://aws.amazon.com/what-is/iac/` Accessed: 22/01/2025.

[39] HashiCorp, *What is terraform?* [Online]. Available: `https://developer.hashicorp.com/terraform/intro` Accessed: 22/01/2024.

[40] HashiCorp, *Terraform language documentation.* [Online]. Available: `https://developer.hashicorp.com/terraform/language/` Accessed: 22/01/2025.

[41] Y. Qiu *et al.*, “Simplifying cloud management with cloudless computing,” in *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, ser. HotNets ’23, Cambridge, MA, USA: Association for Computing Machinery, 2023, pp. 95–101, ISBN: 9798400704154. DOI: `10.1145/3626111.3628206`. [Online]. Available: `https://doi-org.remotexs.ntu.edu.sg/10.1145/3626111.3628206`.

[42] HashiCorp, *Use cases.* [Online]. Available: `https://developer.hashicorp.com/terraform/intro/use-cases` Accessed: 22/01/2025.

[43] IBM, *What is redis?* [Online]. Available: `https://www.ibm.com/think/topics/redis` Accessed: 22/01/2025.

[44] Amazon Web Services, *What is a message queue?* [Online]. Available: `https://aws.amazon.com/message-queue/` Accessed: 22/01/2025.

[45] G. Fu, Y. Zhang and G. Yu, “A fair comparison of message queuing systems,” *IEEE Access*, vol. 9, pp. 421–432, 2020.

[46] Amazon Web Services, *What is apache kafka?* [Online]. Available: `https://aws.amazon.com/what-is/apache-kafka/` Accessed: 22/01/2025.

[47] Apache Software Foundation, *Documentation.* [Online]. Available: `https://kafka.apache.org/documentation/` Accessed: 22/01/2025.

[48] Broadcom Inc., *Which protocols does rabbitmq support?* [Online]. Available: `https://www.rabbitmq.com/docs/protocols` Accessed: 22/01/2025.

[49] L. Johansson, *Part 1: Rabbitmq for beginners - what is rabbitmq?* [Online]. Available: `https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html` Accessed: 22/01/2025.

[50] PubNub, *What is rabbitmq?* [Online]. Available: `https://www.pubnub.com/guides/rabbitmq/` Accessed: 22/01/2025.

[51] P. Dobbelaere and K. S. Esmaili, “Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper,” Jun. 2017, pp. 227–238. DOI: `10.1145/3093742.3093908`.

[52] Broadcom Inc., *Negative acknowledgements*. [Online]. Available: `https://www.rabbitmq.com/docs/nack` Accessed: 22/01/2025.

[53] D. Wang, X. Wang and S. Lv, "An overview of end-to-end automatic speech recognition," *Symmetry*, vol. 11, no. 8, p. 1018, 2019.

[54] S. Gunja, *What is chaos engineering?* [Online]. Available: `https://www.dynatrace.com/news/blog/what-is-chaos-engineering/` Accessed: 22/01/2025.

[55] Chaos Mesh, *Basic features*. [Online]. Available: `https://chaos-mesh.org/docs/basic-features/` Accessed: 22/01/2025.

[56] Chaos Mesh, *Chaos mesh overview*. [Online]. Available: `https://chaos-mesh.org/docs/` Accessed: 22/01/2025.

[57] Amazon Web Services, *What is aws fault injection service?* [Online]. Available: `https://docs.aws.amazon.com/fis/latest/userguide/what-is.html` Accessed: 22/01/2025.