Task Management: An AWS-Powered To-Do Application

Paulo Macedo

Universidade de Aveiro

Task Management: An AWS-Powered To-Do Application

Universidade de Aveiro

Paulo Macedo - paulomacedo@ua.pt

07/12/2024

Abstract

The design, development, and deployment of a task management web application on the Amazon Web Services (AWS) platform are presented in this report. The main objective of the project was to create a scalable and safe application by fusing present cloud technology with basic software development techniques. The system features a fully orchestrated environment driven by Infrastructure as Code (IaC) principles through Terraform, utilizing services like the Virtual Private Cloud (VPC), Elastic Container Service (ECS), Relational Database Service (RDS), and AWS Cognito for authentication.

The implementation of Continuous Integration and Continuous Deployment (CI/CD) pipelines is a crucial component of this endeavour. Implementing CI/CD gave essential practical expertise with automated provisioning, version-controlled infrastructure changes, and rapid, iterative delivery processes, even though it was not strictly needed by the project's original requirements. Time restrictions prevented automated tests from being used in the solution's completion, but the pipeline that results provides a strong basis that may be expanded in the future with testing and improved verification methods. In the end, this project fosters a greater understanding of AWS services, agile techniques, and DevOps processes by demonstrating the process of converting theoretical concepts into a functional cloud-native application.

Contents

List of Tables

List of Figures

Acronyms

AWS	Amazon Web Services
VPC	Virtual Private Cloud
ALB	Application Load Balance
ECS	Elastic Container Service
ECR	Elastic Container Registry
RDS	Relational Database Service
API	Application Programming Interface
CIDR	Classless Inter-Domain Routing
JWT	JSON Web Token
OAC	Origin Access Control
NAT	Network Address Translation
IaC	Infrastructure as Code
CI	Continuous Integration

CD	Continuous Delivery
IAM	Identity and Access Management

Chapter 1

Introduction

1.1 Problem Statement

The rapid advancement of technology has increased the demand for web-based applications that are secure, user-friendly, and scalable. However, developing such solutions requires a solid understanding of software development principles, cloud infrastructure, and security practices. This project addresses the need to build a practical software solution that demonstrates the integration of web-based technologies with cloud computing. By focusing on a To-Do List Application, this project emphasizes creating a real-world system with essential functionalities like task management, user authentication, and secure deployment on AWS. The solution aims to combine theoretical knowledge with practical implementation, enabling effective tackle of common challenges in modern software development.

1.2 Motivation

The primary motivation for this project is to provide experience in designing, implementing, and deploying a software application on a cloud platform like AWS. Cloud computing has become a cornerstone of modern software development, delivering scalability, reliability, and flexibility. This project provides an opportunity to gain hands-on experience with industry-standard tools and practices, such as RESTful APIs, relational databases, authentication mechanisms, and agile development workflows. Additionally, the development of a To-Do List Application provides a straightforward yet comprehensive framework to practice these skills, fostering a deeper understanding of end-to-end software development and deployment processes.

1.3 Objectives

The objectives of this project are as follows:

1. **Software Development:** Develop a fully functional To-Do List Application hosted on AWS. The application must include a RESTful API, a web-based user interface (Web UI), and interaction with a relational database for data storage and retrieval.
2. **Agile Workflow:** Employ an agile development methodology with tools like JIRA to manage epics, sprints, and user stories. This approach promotes iterative development and efficient task management.
3. **Authentication and Authorization:** Implement robust authentication and authorization mechanisms using an Identity Provider (IDP) such as AWS Cognito. The solution should ensure that each user's data remains private and secure.
4. **Security and Deployment:** Ensure the application follows best security practices during deployment, such as avoiding unnecessary open ports. The system should be deployed securely on AWS using either EC2 instances or ECS with Docker containers.
5. **Functional Capabilities:** Provide users with features such as task ownership, management, prioritization, deadlines, and sorting/filtering. These functionalities will ensure a comprehensive task management experience.
6. **Documentation:** Use the OpenAPI specification to document the RESTful API, ensuring clear communication and usability for developers and stakeholders.

Chapter 2

Methodology

This chapter details the methodology employed during the development of the project. The Agile methodology, specifically the Scrum framework, which allowed adaptation, and incremental delivery of functionality. Key aspects such as the definitions of "Ready" and "Done," sprint cycles, user stories, and priority management are discussed below.

2.1 Agile Concepts

Agile is a methodology that promotes iterative development, collaboration, and flexibility. Scrum, a subset of Agile, was used as the guiding framework for this project. The following concepts were fundamental to the project's execution:

2.1.1 Scrum Framework

Scrum is a lightweight framework designed for managing iterative and incremental projects. It revolves around key roles (Product Owner, Scrum Master, and Development Team), ceremonies (Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective), and artifacts (Product Backlog and Sprint Backlog).

2.1.2 Definition of Ready

The Definition of Ready (DoR) sets the criteria that a task or user story must meet before the development can start work. It ensures all necessary details are provided, allowing the team to begin with clarity and confidence. By following the DoR, the development process is streamlined, leading to better software quality.

2.1.2.1 User Stories

Our user stories adhere to the characteristics defined by the **INVEST** model:

- **Independent:** The story can be developed and delivered without relying on other user stories.
- **Negotiable:** The story is not fixed or rigid. It's a starting point for discussion between the customer (or Product Owner) and the development team.
- **Valuable:** The story must deliver value to the user or customer. Stories that do not add value should be reconsidered.
- **Estimable:** The story must be easy to estimate in terms of effort and time. This helps prioritize it effectively. Stories that are valuable but too time-consuming may need to be reconsidered.
- **Small:** For two-week sprints, the story should take no more than 3–4 days to complete, including everything required to reach a "done" state.
- **Testable:** The story must have clear acceptance criteria, so it can be tested to confirm it meets the requirements.

2.1.2.2 Acceptance Criteria

Each user story includes specific **Acceptance Criteria**, written in the format:

```
Given <Context>, When <Action>, Then <Result>
```

This format ensures that stories are clearly defined and testable.

2.1.2.3 Fibonacci-Based Story Point Scale

Each user story or task is assigned an estimated cost, determined by evaluating its complexity, the effort required, and any associated risks. The Fibonacci sequence is used as the basis for story points to account for the nonlinear nature of complexity and effort estimation.

Story Point	**Effort Required**	**Time Required**
1	Very Small (Trivial)	A few minutes
2	Small	A few hours
3	Medium	One day
5	Large	A few days
8	Very Large	One week
13	Extra Large	Up to one month

Table 2.1: Fibonacci-Based Story Point Scale

2.1.3 Definition of Done

The **Definition of Done (DoD)** outlines the criteria that must be met for a product increment, such as a user story, to be considered complete and ready for release. It ensures all acceptance criteria are fulfilled, helping the team deliver high-quality software and preventing incomplete work from being included in the final increment.

The **Definition of Done** includes the following criteria:

- Meets the acceptance criteria.
- Code is written and reviewed.
- The feature is deployed to the staging environment.

2.1.4 Sprints

The project was executed in multiple sprints, each lasting **one week**. Below is a summary of each sprint, including objectives, accomplishments, and reflections on performance:

2.1.4.1 Sprint 1

Objectives: The primary goals for this sprint included setting up the development environment, finalizing initial user stories, and creating a comprehensive project backlog. Additional objectives were to begin implementing a basic task list view and to develop user registration and login functionality.
Achievements: The development environment was successfully set up, and the project backlog was established. However, none of the user stories planned for the sprint were fully completed.
Reflections: This sprint highlighted the need for greater granularity in defining tasks to ensure they are manageable and achievable within the sprint timeframe.

2.1.4.2 Sprint 2

Objectives: Finish the user registration, login, and the task list view. Implement other expected functionalities related to the to-do app, such as defining tasks as completed, editing, deleting, filtering, sorting, and adding new tasks.
Achievements: From all planned functionalities, only two user stories were completed. By the end of this sprint, users could add a new task and list it.
Reflections: An overestimation of the workload led to many user stories remaining unfinished. Additionally, time management during the sprint should have been better.

2.1.4.3 Sprint 3

Objectives: Transition all incomplete objectives from the previous sprint to this one and aim to complete them.

Achievements: While some issues were resolved, no user stories were fully completed.
Reflections: The lack of sufficient time during the sprint was a significant factor in this outcome. Managing the workload from this sprint alongside other academic responsibilities proved challenging.

2.1.4.4 Sprint 4

Objectives: Transfer all user stories from the previous sprint to this one and aim to complete them.
Achievements: Successfully completed the user registration and login functionalities.
Reflections: Due to the workload from other subjects, no additional user stories were finished. Improved time management should be a priority moving forward.

2.1.4.5 Sprint 5

Objectives: Complete the remaining user stories from the previous sprints. Enable users to modify their profile information, fully integrate the backend with the frontend using signed Application Programming Interface (API) calls (via the Cognito-provided token), and deploy the application infrastructure to AWS.
Achievements: All user stories were successfully completed, including the integration of the backend with the frontend and deployment to AWS.
Reflections: This sprint was the most productive due to a reduced workload from other subjects and a better understanding of the AWS workflow. This understanding enabled faster iterations and efficient completion of all user stories and deployment.

Chapter 3

Implementation

The Implementation chapter provides a comprehensive overview of the development and deployment processes undertaken to build the ToDo web application. This chapter is structured to guide the reader through the architectural framework, the core functionalities of the application, and the deployment strategies.

3.1 Architecture

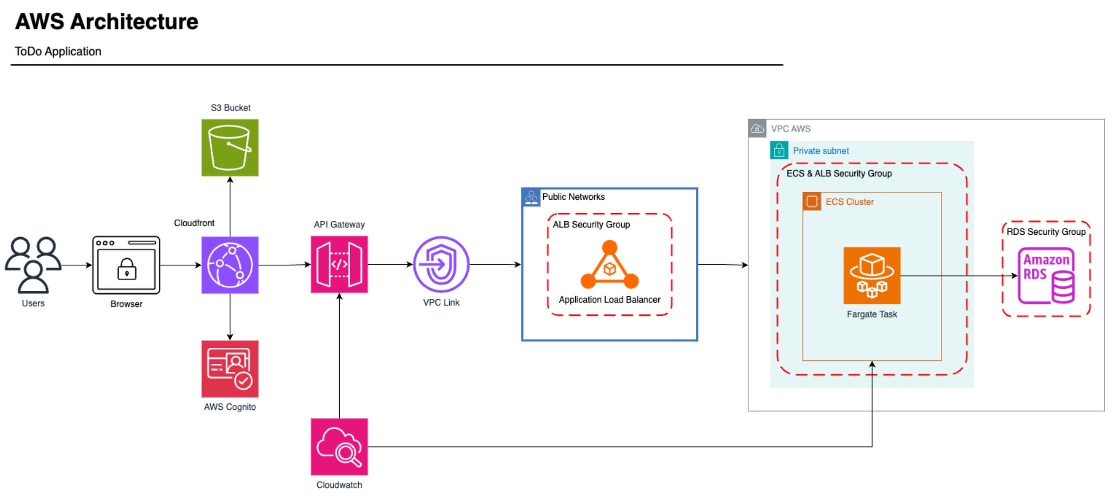

Figure 3.1: ToDo Application - AWS Architecture

This architecture consists of several AWS services working together to host a secure, scalable web application. The main components include:

- Virtual Private Cloud (VPC) and Networking

- Security Groups
- Application Load Balance (ALB)
- Elastic Container Service (ECS)
- Elastic Container Registry (ECR)
- Relational Database Service (RDS)
- API Gateway with VPC Link
- Cognito for Authentication
- CloudFront and S3 for Frontend Hosting

3.2 Components and Connections

The architecture is built upon a VPC that establishes an isolated network environment defined by a Classless Inter-Domain Routing (CIDR) block. Within this VPC, public and private subnets are distributed across two availability zones, ensuring both redundancy and high availability. The public subnets provide external connectivity and contain resources such as the Internet Gateway, Network Address Translation (NAT) Gateway, and ALB, while the private subnets host critical internal components like ECS tasks running on Fargate, the RDS database instance, and API Gateway VPC Link endpoints. Internet-bound traffic from public subnets routes through the Internet Gateway, and private subnet traffic passes through the NAT Gateway. These routing decisions are enforced via distinct route tables associated with each subnet type.

A system of security groups governs all network traffic. The ALB security group permits inbound requests from the API Gateway VPC Link and routes outbound responses to ECS tasks. The ECS security group accepts inbound application traffic from the ALB and grants outbound access to the RDS database, which in turn is protected by the RDS security group to limit inbound connections to authorized ECS tasks. The API Gateway security group ensures that the VPC Link can interact securely with the ALB.

Core application services run on ECS, where an ECS cluster hosts services defined by ECS task definitions and managed by ECS services that integrate seamlessly with the ALB. Docker images for these tasks reside in an ECR repository, which enforces a lifecycle policy to retire outdated images, but in the current implementation there's no version control. Beyond the backend infrastructure, the API Gateway serves as the entry point for RESTful calls, using Cognito for JSON Web Token (JWT)-based authentication. Through the VPC Link, the Gateway privately connects to the ALB, preserving the isolated nature of the environment. The user interface is hosted as static content on S3 (without versioning as well) and distributed through CloudFront, which employs Origin Access Control (OAC) to maintain strict security.

The RDS instance is deployed within a private subnet to ensure data security. Currently, the RDS database instance resides in a single availability zone, which may pose a risk in the event of an specific availability zone failure. Ideally, deploying the RDS instance across multiple availability zones would enhance fault tolerance and ensure higher availability, minimizing downtime and improving overall resilience.

3.3 Data Flow

In this environment, end users first interact with the frontend delivered by CloudFront, which fetches static assets from the S3 bucket. Before accessing protected API endpoints, users authenticate through Cognito, receiving a JWT token for subsequent requests. The API Gateway, upon verifying the presented JWT, forwards authorized requests via the VPC Link to the internal ALB. The ALB then balances incoming traffic and routes it to the ECS tasks running the application logic. During request processing, the ECS tasks interact with the RDS database as needed, and the resulting responses are returned through the ALB to the API Gateway, which then delivers them back to the client's browser. This flow ensures that authentication, routing, application logic, and database interactions occur securely within a well-defined network boundary.

3.4 Application Functionalities

The Reminders Application integrates with AWS Cognito for user authentication and manages user accounts and tasks through a backend API. Below is a high-level summary of the core functionalities.

- **User Authentication and Session Management:**
  - Use AWS Cognito Hosted UI for user sign-in.
  - Retrieve the current authenticated user and session tokens.
  - If authenticated, store user data (username, Cognito user ID) and send to backend to ensure user existence in the database.
- **User Navigation and Onboarding:**
  - From the home page, authenticated users are redirected to the `/reminders` page.
  - Unauthenticated users are prompted to sign in through the Hosted UI.
- **User Profile Management (UserProfile Page):**
  - Fetch user profile information (full name, phone number, email, account creation date) and task statistics (total tasks created and completed) from the backend using the user's `userId` and ID token.

  - Edit user attributes such as full name and phone number; update changes persistently.
  - Display static information like email, tasks created, tasks completed, and account creation date.
- **Task Management (Reminders Page):**
  - Retrieve user tasks from the backend with filtering (by completed and uncompleted) and sorting (by criteria like creation date, deadline, status, priority).
  - Create, edit, update, and delete tasks.
  - Maintain up-to-date task lists and reflect changes immediately in the UI.
  - Open a task modal for adding or editing tasks.

Integration and Workflow

Overall, the application:

1. Authenticates the user via Cognito and stores their account in the backend upon verification.
2. Provides a user profile page to view and edit personal details.
3. Offers a reminders (tasks) page allowing the user to view, filter, sort, create, update, complete, and delete tasks.

3.5 Deployment

The deployment of the ToDo web application leverages Terraform for infrastructure management and a Continuous Integration (CI) and Continuous Delivery (CD) workflow for automatic provisioning and updates. This approach ensures consistent, repeatable deployments while reducing manual intervention.

3.5.1 Infrastructure as Code with Terraform

Terraform, an Infrastructure as Code (IaC) tool, is employed to define all the AWS infrastructure components needed by the application, including the VPC, subnets, security groups, ECS clusters, and RDS instances. While it was not a mandatory requirement for this project, and using the AWS console would have been a completely understandable approach for simpler cases, leveraging Terraform offers significant long-term benefits. Representing infrastructure configurations in version-controlled code makes it easier to audit changes, revert to previous states when needed, and maintain a comprehensive record of modifications. Moreover, Terraform's modular design supports the reuse of configuration blocks across multiple environments, simplifying the process of scaling as the infrastructure becomes more complex.

3.5.2 Automated CI/CD Workflows

A CI/CD pipeline ensures that updates to the infrastructure or application code are automatically deployed whenever changes are merged into the main (production) branch. Although no automated tests have been implemented at this time, the pipeline still serves as a mechanism to deliver stable changes to the production environment without manual oversight. The steps are as follows:

1. **Code Commit and Merge**: Developers make changes to the application codebase or Terraform configurations and merge their work into the main branch.

2. **CI/CD Pipeline Execution**: A CI/CD service detects the merge event and initiates a pipeline run. Since no automated testing stages are currently configured, this step primarily involves preparing and validating Terraform configurations.

3. **Infrastructure Provisioning with Terraform**: The pipeline executes Terraform to apply any new or updated infrastructure configurations. This ensures the AWS environment remains consistent with the latest code, accounting for factors like new ECS services, updated Identity and Access Management (IAM) policies, or changes to the database configuration.

4. **Application Deployment**: Once the infrastructure state is confirmed, the updated application images are pushed to the ECS environment. This may involve refreshing ECS tasks with the new image versions. If the application logic or its dependencies have changed, these updates take effect immediately in the production environment.

5. **Post-Deployment Review**: Although there are currently no automated tests, the development team relies on manual verification and infrastructure monitoring tools (Cloudwatch) to ensure that the deployment was successful and that all services are functioning as intended. In the future, integrating automated tests would further enhance confidence in the deployment process.

3.5.3 Benefits of the Automated Deployment Strategy

Despite the absence of automated tests, adopting an Infrastructure as Code approach with Terraform and integrating CI/CD pipelines delivers several advantages. Deployments are more consistent, as all changes are defined in code and applied in a controlled manner. Infrastructure state can be tracked over time, simplifying maintenance and debugging. With minimal manual intervention, deployments become more efficient, enabling faster updates and a more agile response to changing requirements. While currently limited in its automated validation steps, this pipeline provides a solid foundation that can be expanded with automated tests and additional verification processes in the future. Although CI/CD was not mandated for this project, its implementation was a deliberate choice made to gain practical knowledge and improve operational efficiency.

Chapter 4

Conclusion

The goal of this project was to develop a cloud-native, safe, and useful task management application that runs on AWS. All of the main goals—a scalable backend, a responsive frontend, dependable user authentication, and a smooth integration of infrastructure components like ECS, RDS, and Cognito—were achieved through meticulous planning and iterative development utilising agile methodologies. Using Terraform as an IaC tool was very beneficial because it allowed for easier deployments and future scalability while offering the ability to manage and version the entire environment.

The decision to adopt a CI/CD pipeline, despite not being mandated, proved beneficial for exploring best practices in automated infrastructure management. The current solution, while fully operational, also leaves room for improvement. Enhancing database resilience with multi-AZ deployments, refining authentication flows, and introducing comprehensive testing are natural next steps.

Looking back, the sprints provided valuable lessons on the need for adaptability and incremental improvement. Early on, I had trouble finishing planned user stories, had trouble predicting effort effectively, and had trouble juggling project goals with academic obligations. Unfinished tasks were finished at the end of the last sprint, resulting in a stable and fully functional solution.

All things considered, this project provided an opportunity to obtain hands-on experience with CI/CD processes, Terraform-based infrastructure management, and important AWS services. The capacity to produce reliable, scalable cloud-based apps can be strengthened by applying the lessons learnt here regarding architectural choices, iterative development, and DevOps integration to more complicated and production-oriented contexts.

Bibliography

[1] Hashcorp, *Hashcorp Terraform AWS*, 2023. [Online]. Available: https://registry.terraform.io/providers/hashicorp/aws/latest/docs.

[2] AWS, *Test CORS for an API Gateway API*, 2024. [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-test-cors.html.

[3] AWS, *CORS for REST APIs in API Gateway*, 2024. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/cors.html.

[4] AWS, *Using cross-origin resource sharing (CORS)*, 2024. [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-cors.html.

[5] AWS, *Terraform and CORS-Enabled AWS API Gateway*, 2024. [Online]. Available: https://mrponath.medium.com/terraform-and-aws-api-gateway-a137ee48a8ac.

[6] AWS, *Routing traffic to a website that is hosted in an Amazon S3 bucket*, 2024. [Online]. Available: https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/RoutingToS3Bucket.html.

[7] AWS, *Use an Amazon CloudFront distribution to serve a static website*, 2024. [Online]. Available: https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/getting-started-cloudfront-overview.html.

[8] Olumoko Moses, *Hosting a Static Website on AWS Using S3, CloudFront, and Route53*, 2024. [Online]. Available: https://dev.to/aws-builders/guide-to-hosting-a-static-website-on-aws-using-s3-cloudfront-and-route53-with-just-7-steps-220b.

[9] AWS, *Hosting a static website using Amazon S3*, 2024. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/WebsiteHosting.html.

[10] AWS, *Understanding the access token*, 2024. [Online]. Available: https://docs.aws.amazon.com/cognito/latest/developerguide/amazon-cognito-user-pools-using-the-access-token.html.

[11] AWS, *Getting credentials*, 2024. [Online]. Available: `https://docs.aws.amazon.com/cognito/latest/developerguide/getting-credentials.html`.

[12] Amplify, *Amplify Documentation for React*, 2024. [Online]. Available: `https://docs.amplify.aws/react/`.

[13] AWS, *Working with user attributes*, 2024. [Online]. Available: `https://docs.aws.amazon.com/cognito/latest/developerguide/user-pool-settings-attributes.html`.

[14] AWS, *User pool managed login*, 2024. [Online]. Available: `https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-pools-managed-login.html#cognito-user-pools-app-integration-amplify`.

[15] AWS, *Restrict access to an Amazon Simple Storage Service origin*, 2024. [Online]. Available: `https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/private-content-restricting-access-to-s3.html`.

[16] AWS, *Setting permissions for website access*, 2024. [Online]. Available: `https://docs.aws.amazon.com/AmazonS3/latest/userguide/WebsiteAccessPermissionsReqd.html`.

[17] AWS, *Enabling website hosting*, 2024. [Online]. Available: `https://docs.aws.amazon.com/AmazonS3/latest/userguide/EnableWebsiteHosting.html`.

[18] AWS, *Blocking public access to your Amazon S3 storage*, 2024. [Online]. Available: `https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-control-block-public-access.html`.

[19] AWS, *Configuring a static website on Amazon S3*, 2024. [Online]. Available: `https://docs.aws.amazon.com/AmazonS3/latest/userguide/HostingWebsiteOnS3Setup.html`.

[20] AWS, *AWS CLI Command Reference*, 2024. [Online]. Available: `https://docs.aws.amazon.com/cli/latest/reference/cognito-idp/initiate-auth.html?highlight=initiate%20auth`.

[21] Andrew Tarry, *AWS HTTP Api Gateway with Cognito and Terraform*, 2024. [Online]. Available: `https://andrewtarry.com/posts/aws-http-gateway-with-cognito-and-terraform/`.

[22] Carlos Hernández, *User control with Cognito and API Gateway*, 2024. [Online]. Available: `https://medium.com/carlos-hernandez/user-control-with-cognito-and-api-gateway-4c3d99b2f414`.

[23] AWS, *Working with storage for Amazon RDS DB instances*, 2024. [Online]. Available: `https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_PIOPS.StorageTypes.html`.

[24] AWS, *Amazon RDS for PostgreSQL updates*, 2024. [Online]. Available: https://docs.aws.amazon.com/AmazonRDS/latest/PostgreSQLReleaseNotes/postgresql-versions.html#postgresql-versions-version170Preview.

[25] AWS, *Amazon ECS task IAM role*, 2024. [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-iam-roles.html.

[26] AWS, *Amazon ECS container logs for EC2 and Fargate launch types*, 2024. [Online]. Available: https://docs.aws.amazon.com/prescriptive-guidance/latest/implementing-logging-monitoring-cloudwatch/ec2-fargate-logs.html.

[27] AWS, *Start AWS Fargate logging for your cluster*, 2024. [Online]. Available: https://docs.aws.amazon.com/eks/latest/userguide/fargate-logging.html.

[28] AWS, *Logging and Monitoring in Amazon Elastic Container Service*, 2024. [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-logging-monitoring.html.

[29] Nidhi Ashtikar, *AWS RDS-Convert Single-AZ to Multi-AZ with Terraform*, 2024. [Online]. Available: https://nidhiashtikar.medium.com/aws-rds-convert-single-az-to-multi-az-with-terraform-6cff39c34e45.

[30] AWS, *Authenticate users using an Application Load Balancer*, 2024. [Online]. Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/listener-authenticate-users.html.

[31] Porsche Digital, *Cognito user authentication on ALB vs API Gateway*, 2024. [Online]. Available: https://medium.com/next-level-german-engineering/cognito-user-authentication-alb-api-gateway-13b16f319d32.

[32] Artem Hatchenko, *ALB: Cognito authentication in an unsupported region*, 2024. [Online]. Available: https://medium.com/@artem.hatchenko/alb-cognito-authentication-in-an-unsupported-region-a13bb5c3696c.

[33] Sergei Shevlyagin, *Run a Python Selenium web scraper on AWS Fargate*, 2019. [Online]. Available: https://medium.com/@sshevlyagin/run-a-python-selenium-web-scraper-on-aws-fargate-5a84a54d042e.

[34] Dr. Tri Basuki Kurniawan, *Deploy Docker Container as serverless architecture to AWS Fargate*, 2021. [Online]. Available: https://medium.com/thelorry-product-tech-data/deploy-docker-container-as-serverless-architecture-to-aws-fargate-1121bafa1d8c.