**CSCI5411**

**Advanced Cloud Architecting**

**Project Report**

Contents

[Project Description: 3](#_Toc184079473)

[Architecture Diagram: 4](#_Toc184079474)

[Sequence Diagram: 6](#_Toc184079475)

[Services Used: 7](#_Toc184079476)

[Justification of Chosen AWS Services: 8](#_Toc184079477)

[AWS Well-Architected Framework principles and best practices 16](#_Toc184079478)

[Cloud Formation: - 19](#_Toc184079479)

Project Name: AudioBook Extractor

# Project Description:

The **AudioBook Extractor** project enables users to upload files, which are then processed and converted into audio format.

1. **File Upload to S3**:
   * Users initiate a request through the API Gateway to upload their files.
   * The API Gateway triggers a Lambda function (**PutDataInS3**) that checks if the extracted data for the file already exists in the Redis cache.
   * If the extracted data is cached, the Lambda function returns the data to the user.
   * If the data is not cached, the Lambda function stores the file in S3.
   * Once the file is stored in S3, an event notification triggers another Lambda function to extract text from the file using Amazon Textract.
   * The extracted data is then stored in both S3 and Redis(expiration time as 1 hour cache) for future use.
2. **Retrieve Extracted Data from S3**:
   * Users can request to retrieve extracted data through the API Gateway, which calls a Lambda function (**ExtractData**) to fetch the data from S3 and return it to the user.

# Architecture Diagram:

A screenshot of a computer

Description automatically generated

*Figure 1: Architecture Diagram*

# Sequence Diagram:

A diagram of a diagram

Description automatically generated

*Figure 2: Sequence Diagram*

# Services Used:

Total of 11 services are used while creating this project which are divided into 7 categories:

#### Compute:

* + AWS Lambda
  + AWS beanStalk

#### Storage:

* + Amazon S3

#### Database:

* + Amazon ElastiCache for Redis Serverless

#### Networking and Content Delivery:

* + Amazon API Gateway
  + VPC

#### Machine Learning:

* + Amazon Textract

#### Security:

* + AWS GuardDuty
  + Amazon Textract

#### Management and Governance:

* + AWS CloudFormation
  + Amazon CloudWatch

# Justification of Chosen AWS Services:

#### 1. Amazon S3

* **Project Need:**The application needs a durable, scalable storage solution for user-uploaded files (e.g., documents which can be PDF, Images,PNG) and processed data (extracted text).
  + Additionally, it must support event-driven workflows to trigger data extraction processes.
* **Why Amazon S3?**
  + **Scalability:** S3 seamlessly handles growing amounts of uploaded files without manual provisioning. This is critical for an application expecting varying user traffic**.**
  + **Event Notifications:** Native integration with AWS Lambda allows the application to trigger data extraction workflows upon file uploads.
  + **Cost-Effectiveness:** Pay-per-use pricing aligns well with the application's needs, where storage usage varies with user activity.
* **Why Not Other Services?**
  + **EBS (Elastic Block Store):** EBS is block-level storage requiring attachment to EC2 instances, making it unsuitable for storing and accessing unstructured data like user files. Its lifecycle is tied to specific instances, reducing flexibility.
  + **EFS (Elastic File System):** EFS is designed for file-based workloads requiring concurrent access. For this project, object storage with event-driven processing is more relevant, making S3 a better fit.

#### 2. AWS Lambda

**PutDataInS3 Lambda**

* **Project Need:**Handles file uploads, interacts with Redis for caching, and stores new files in S3. A serverless solution is essential to minimize operational overhead and handle unpredictable workloads.

**Data Extraction Lambda**

* **Project Need:**Processes files after upload to S3, extracts text using Textract, and stores results in Redis and S3. A lightweight, event-driven solution aligns with the application's requirements.
* **Why AWS Lambda?**
  + **Serverless Execution:** Automatically scales to handle requests, reducing costs and simplifying management.
  + **Event-Driven Architecture:** Eliminates the need for polling mechanisms or scheduled tasks.
  + **Seamless Integration:** Works natively with S3, API Gateway, and Redis for cohesive workflows.
* **Why Not Other Services?**
  + **Amazon EC2:** Requires provisioning and maintaining instances, leading to higher costs and operational overhead, especially when traffic fluctuates.
  + **AWS Fargate:** While it supports containerized workloads, the added complexity of containers is unnecessary for this project's simple, event-driven tasks.

#### 3.AWS Elastic Beanstalk

* **Purpose**:  
  Elastic Beanstalk is used to deploy and manage the frontend of the application, simplifying the process of provisioning, monitoring, and scaling resources.

**Why AWS Elastic Beanstalk?**

* Elastic Beanstalk automates the deployment process for web applications, eliminating the need to manage infrastructure.
* Elastic Beanstalk supports auto-scaling to handle fluctuating traffic. For example, if the user base grows, Beanstalk automatically scales up resources to meet demand.
* This ensures the application remains responsive without manual intervention.
* Built-in integration with Amazon CloudWatch allows real-time monitoring of application health and performance.
* Beanstalk charges only for the underlying resources used (EC2, RDS, etc.) without any additional management costs.

**Comparison with Alternatives**

**1. Amazon ECS/EKS (Containers):**

* While ECS/EKS offers greater control over containerized environments, they require more setup and management expertise.

**2. Traditional EC2 Instances:**

* Deploying on EC2 would require manual provisioning, configuration, scaling, and monitoring, leading to increased operational overhead.

#### 4. Amazon Textract

* **Project Need:**Extracts text from uploaded documents to convert them into audio. The application demands an accurate, managed text-extraction solution integrated with AWS services.
* **Why Amazon Textract?**
  + **Accuracy:** Optimized for extracting text from both structured and unstructured documents, this solution supports multiple languages including English, German, French, Spanish, Italian, and Portuguese. It can process a variety of file formats such as PDF, PNG, and JPG. Additionally, it is capable of extracting text from handwritten content, making it a versatile tool for a wide range of document types and formats.
  + **Integration:** Works seamlessly with S3 and Lambda for automated workflows.
* **Why Not Other Services?**
  + **Amazon Rekognition:** Primarily designed for image and video analysis, lacking document-specific OCR capabilities.
  + **Third-Party OCR Solutions:** Would introduce additional costs, latency, and complexity in integration.

#### 5. Amazon API Gateway

* **Project Need:**Acts as the interface between users and the backend. It must provide scalable HTTP endpoints for uploading and retrieving files while ensuring security and reliability.
* **Why API Gateway?**
  + **Managed Service:** No need for server management, reducing operational effort.
  + **Scalability:** Automatically adjusts to handle varying traffic loads.
  + **Security Features:** Supports API keys, throttling, and request validation to protect the backend.
  + **Integration with Lambda:** Simplifies routing requests to Lambda functions for processing.
* **Why Not Other Services?**
  + **Elastic Load Balancer (ELB):** While ELB distributes traffic, it does not provide API management capabilities like throttling or authentication.

#### 7. Amazon ElastiCache (Redis)

* **Project Need:**A caching layer is essential to store extracted text for quick retrieval, reducing redundant processing for files that have already been analyzed.
* **Why ElastiCache (Redis)?**
  + **Low Latency:** Provides in-memory data storage for near-instantaneous access.
  + **Managed Service:** AWS handles setup, maintenance, and scaling, reducing operational burden.
  + **Serverless:** The which is implemented in this project is serverless, so we Don’t have to worry about any complexity.
* **Why Not Other Services?**
  + **Memcached:** Redis offers advanced capabilities (e.g., persistence, pub/sub) that are lacking in Memcached, making Redis the better choice for this project.

#### 8. VPC

* **Project Need:**Ensures all resources (Lambdas, S3, Redis) operate within a secure, isolated network. Sensitive data requires protection from external threats.
* **Why VPC?**
  + **Resource Isolation:** Keeps resources secure by restricting access within a private network.
  + **Customizability:** Supports subnets, routing tables, and security groups for fine-grained control over network traffic.
* **Why Not Other Services?**
  + **Public Internet Access:** Exposing resources directly to the internet increases the risk of unauthorized access.

#### 9. AWS GuardDuty

* **Project Need:**Protects sensitive resources (e.g., S3, Lambda, Redis) in the VPC from security threats such as unauthorized access, data exfiltration, or malicious activities. In this application, users may upload files, and some of these files could potentially be malicious. GuardDuty plays a critical role in detecting and mitigating such threats.
* **Why AWS GuardDuty?:**
  + **Threat Detection for Malicious Files:**GuardDuty monitors file activities and metadata within S3 buckets to detect malicious behaviors or patterns.
  + **For example:**
    - If a file contains malware or exhibits unusual activity after being uploaded, GuardDuty can flag the threat.
    - Suspicious data exfiltration attempts or unauthorized access to files can also trigger alerts.

#### 10. AWS CloudFormation

**Purpose:**

AWS CloudFormation automates the provisioning and management of infrastructure as code (IaC), ensuring consistent and repeatable deployments.

**Why CloudFormation?**

* **Infrastructure as Code**: Easily define, manage, and version infrastructure in code.
* **Simplified Management**: Automatically handles resource dependencies.
* **Seamless Integration**: Works well with other AWS services like Lambda, S3, and VPC.

#### 11. Amazon CloudWatch

**Purpose:**

CloudWatch monitors and logs AWS resources, providing visibility into application performance.

**Why CloudWatch?**

* **Real-Time Monitoring**: Helps monitor Lambda functions and other resources.
* **Troubleshooting**: Logs allow comparing hashkeys generated across different Lambdas, assisting in debugging.
* **Integration**: Seamlessly integrates with AWS services like Lambda, S3, and API Gateway.

*12. Amazon Virtual Private Cloud (VPC)*

**Purpose:**  
Amazon VPC enables the provisioning of isolated network environments for applications, ensuring secure and controlled access to resources like Redis.

**How It Fits Into Our Application:**  
Our application uses Redis for caching, and Redis must be deployed within the VPC for security and accessibility. By configuring the Lambda functions to run within the same VPC, they can securely access Redis using private IP addresses. This ensures low latency and prevents exposure of Redis to the public internet, adhering to best practices for secure architecture.

# AWS Well-Architected Framework principles and best practices

#### 1. Operational Excellence

* + The entire infrastructure is defined and deployed using AWS CloudFormation, ensuring consistency and making it easy to replicate or modify the environment as needed.
  + I have enabled Amazon CloudWatch logging and monitoring for all Lambda functions to track performance metrics, identify issues.
  + I have utilized AWS Lambda for all key operations to ensure automation and reduce manual overhead. For instance, the PutDataInS3 Lambda automates file uploads and caching checks, while the text extraction process is seamlessly triggered by S3 event notifications.
  + Elastic Beanstalk is used to deploy and manage the backend API services with minimal manual intervention.

#### 2. Security

* + All resources, including S3, Lambda, ElastiCache, are hosted within a VPC for enhanced network isolation, ensuring that no resources are directly exposed to the internet.
  + I have implemented AWS GuardDuty to continuously monitor and detect threats in the environment, providing alerts for suspicious activities.
  + All sensitive data stored in S3 is encrypted at rest using SSE-S3, and encryption in transit is ensured by enforcing HTTPS for all communications.
  + API Gateway endpoints are secured with request validation and throttling to prevent abuse or unauthorized access.

#### 3. Reliability

* + The architecture leverages Amazon S3, which provides 99.99% availability, ensuring the durability and reliability of file storage.
  + Lambda functions automatically scale to handle varying workloads, ensuring that the system remains available even during peak usage.
  + To manage failures, S3 Event Notifications are configured to retry triggering the extraction Lambda in case of errors.
  + I have implemented CloudWatch Alarms to monitor critical metrics and trigger notifications or recovery actions if thresholds are breached.
  + I have designed the Virtual Private Cloud (VPC) architecture to span multiple Availability Zones (AZs), ensuring that the system remains highly available and resilient to failures in any single AZ.
  + The VPC includes multiple private subnets, each located in a different AZ within the region. Services like Amazon ElastiCache (Redis) and AWS Lambda are distributed across these subnets to enhance fault tolerance.
  + Elastic Beanstalk uses multi-AZ deployment options for backend services, ensuring high availability. It also provides automatic load balancing and failover mechanisms.

#### 4. Performance Efficiency

* + The system uses Amazon ElastiCache (Redis) for caching extracted text, reducing latency and avoiding redundant processing.
  + AWS Lambda automatically scales to match the incoming request volume, ensuring optimal resource utilization without manual intervention.
  + The architecture is designed to be event-driven, using S3 events to initiate workflows, eliminating unnecessary polling or periodic checks.
  + Performance metrics for Lambda, Redis, and Elastic Beanstalk are continuously monitored using CloudWatch.

#### 5. Cost Optimization

* + The serverless approach with AWS Lambda eliminates the need for provisioning and maintaining servers, ensuring that costs scale with usage.
  + The ElastiCache (Redis) solution provides efficient in-memory storage, reducing the need to repeatedly query S3 or perform costly extraction operations with Textract.
  + API Gateway usage is optimized by implementing throttling, ensuring cost control even during high traffic.

#### 6. Sustainability

* **Implementation:**
  + By leveraging serverless technologies like Lambda and API Gateway, I have minimized the environmental impact of underutilized resources.
  + Beanstalk's managed scaling ensures that only necessary resources are active, reducing waste and contributing to energy efficiency.
  + The architecture follows AWS's shared responsibility model, where AWS optimizes its data centers for energy efficiency, and I focus on efficient resource usage within the application.

# Cloud Formation: -

#### 1. Redis Serverless Cache

* **RedisServerlessCache**: A serverless Redis cache is created using AWS::ElastiCache::ServerlessCache. It's named MyServerlessRedisCache, and the Engine is set to redis, providing a highly scalable, managed cache without requiring server provisioning.

#### 2. Lambda Functions

* **InsertDataIntoS3CF**: A Lambda function named InsertDataIntoS3CF is created to insert data into an S3 bucket. The function code is sourced from an S3 bucket (term-assignment-initial) There are lot of imports which we have to do hence we are not placing the code of function in yaml file instead we are putting in S3. The Lambda function is configured with a VPC (Virtual Private Cloud) for enhanced network security, referencing specific subnets and security groups. It has a REDIS\_ENDPOINT environment variable set to the Redis serverless cache's endpoint.
* **GetDataFromS3**: Another Lambda function GetDataFromS3 is set up to retrieve data from S3. It also references the same S3 bucket and is deployed in the same VPC as InsertDataIntoS3CF.
* **S3TriggerCF**: This Lambda function is set to trigger whenever new data is uploaded to an S3 bucket. It's also within the same VPC and tied to specific subnets and security groups for controlled access. It has a REDIS\_ENDPOINT environment variable set to the Redis serverless cache's endpoint.

#### 3. API Gateway

* **ApiGatewayRestApi**: The FileUploadAPI is defined in API Gateway with REGIONAL endpoint configuration, enabling API methods for file upload and retrieval.
* **PutFileIntoS3Resource**: The /PutFileIntoS3 resource is created under the API Gateway for uploading files into S3. It integrates directly with the InsertDataIntoS3CF Lambda function.
* **GetTextFromFileResource**: Similarly, the /GetTextFromFile resource is set up for retrieving text from files stored in S3. This is integrated with the GetDataFromS3 Lambda function.
* **Permissions and CORS**: The API Gateway permissions (AWS::Lambda::Permission) allow the gateway to invoke the Lambda functions, and CORS headers are set up for both POST (file upload) and GET (retrieve file content) methods.

#### 4. S3 Bucket

* **termassignmentbucketcf**: An S3 bucket is created to store files. It is also configured to trigger the S3TriggerCF Lambda function whenever an object is created in the bucket.

#### 5. Elastic Beanstalk

* **MyElasticBeanstalkApplication**: An Elastic Beanstalk application is created for deploying a React application. The application is bundled as a .zip file (React.zip) and stored in the same S3 bucket (term-assignment-initial).
* **MyElasticBeanstalkAppVersion**: This defines the version of the Elastic Beanstalk application based on the zip file.
* **MyElasticBeanstalkEnvironment**: The environment is configured with the application name and solution stack (Amazon Linux 2023 v6.4.0 running Node.js 22). Additionally, environment variables (such as the API Gateway URL) and IAM roles (LabRole, LabInstanceProfile) are configured for the EC2 instances running the React application.

#### 6. GuardDuty

* **GuardDutyDetector**: The GuardDuty service is enabled with a detector that will monitor for malicious activity in your AWS environment. It is set to publish findings regularly to enable threat detection.

A screenshot of a computer

Description automatically generated

*Figure 3: Cloud Formation 1*

A screenshot of a computer

Description automatically generated*Figure 4: Cloud Formation 2*