CSCI 5409 Advanced Topic in Cloud Computing

Term Project

Project Title: Handwritten Text Recognition

Banner ID: B00981006

Table of Contents

1. Project Description

What I Built

The project is a Handwritten text recognition system designed to allow users to upload images of handwritten text. The system processes these images to extract the text using OCR (Optical Character Recognition) and provides the extracted text back to the user.

Functionality

- **Compute services:**
  EC2 – Launches instance that can be used to deploy and host web app.
  Elastic Beanstalk – Simplifies the deployment and management. It is handling provisioning and load balancing.

- **Storage:**
  S3 – Creates bucket to store files of the uploaded handwritten text images, processed files and script of glue.

- **Networking:**
  API Gateway – Creates REST APIs that creates an entry for webapp to allow HTTP requests.

- **General:**
  Textract – Extracts the text from image.
  Glue – It is used for data pre-processing and cleaning. Have setup ETL job to prepare the data. Glue crawler will read from S3 bucket where Textract outputs extracted data.

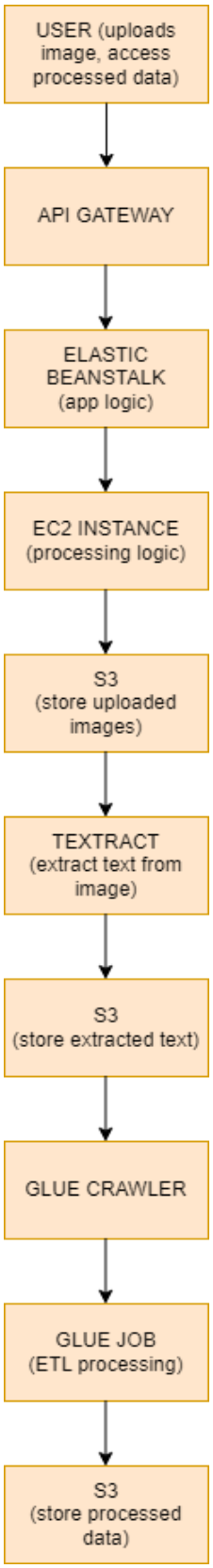

Figure 1 Flowchart of the app

2. Menu Item Requirements

Selected Services and Alternatives

1. **Amazon S3 - Alternatives (EBS, EFS)**
    - **Reason:** S3 is highly scalable, durable, and cost-effective for storing large amounts of unstructured data like images. It also integrates well with other AWS services.
    - **Alternative Comparison:** EBS is suited for block storage and attached to EC2, making it less ideal for object storage. EFS is a file storage service for EC2, more suitable for shared file storage.
2. **Amazon Textract - Alternatives (Tesseract OCR, Google Cloud Vision)**
    - **Chosen:** Amazon Textract
    - **Reason:** Textract is fully managed, easily integrates with other AWS services, and provides high accuracy for text extraction from various document formats.
    - **Alternative Comparison:** Tesseract OCR is open-source but requires more manual setup and integration. Google Cloud Vision is another excellent choice but would involve managing multiple cloud providers. [1]
3. **AWS Glue - Alternatives (Apache Airflow, AWS Lambda)**
    - **Chosen:** AWS Glue
    - **Reason:** AWS Glue is a fully managed ETL service that automatically discovers and processes data. It simplifies the ETL workflow with minimal setup. [2]
    - **Alternative Comparison:** Apache Airflow offers more control but requires more setup and management. AWS Lambda is good for serverless compute tasks but may be complex for complete ETL workflows.
4. **Amazon EC2 - Alternatives (AWS Fargate, AWS Lambda)**
    - **Chosen:** Amazon EC2
    - **Reason:** EC2 provides full control over the computing environment, necessary for running the Flask application and handling various processing tasks.
    - **Alternative Comparison:** AWS Fargate offers a serverless way to run containers but might be overkill for the current scale.[3] AWS Lambda is cost-effective for small, event-driven tasks but less suitable for a web application server.
5. **AWS Elastic Beanstalk - Alternatives (AWS ECS, Kubernetes)**
    - **Chosen:** AWS Elastic Beanstalk
    - **Reason:** Elastic Beanstalk simplifies deployment and management of the application, handling scaling, monitoring, and load balancing.
    - **Alternative Comparison:** AWS ECS and Kubernetes offer more granular control over container orchestration but require more management overhead.

3. Deployment Model

The chosen deployment model for this application is a **Hybrid Cloud** model, utilizing both **AWS Elastic Beanstalk** and **EC2 instances**. [4]

Reason for Choosing this Model:

1. **Balance Between Control and Simplicity**:
    - **AWS Elastic Beanstalk** simplifies the deployment and management of applications by automatically handling the infrastructure setup, including load balancing, scaling, and health monitoring. This allows to focus on writing code and managing the application without worrying about the underlying infrastructure complexities.
    - **EC2 Instances** offer granular control over the underlying compute resources, which is beneficial for fine-tuning performance, security, and other configurations specific to the application's needs. This control allows for custom software installations and detailed performance adjustments.
2. **Easy Scaling and Management**:
    - **Elastic Beanstalk** can automatically scale your application up or down based on demand. This auto-scaling capability ensures that the application can handle varying levels of traffic without manual intervention, providing flexibility and reliability.
    - The Elastic Beanstalk management console provides an intuitive interface to monitor application health, logs, and other metrics, making it easier to manage the application lifecycle effectively.
3. **Full Control Over EC2 Instances**:
    - While Elastic Beanstalk abstracts much of the infrastructure complexity, it still provides the option to access and configure the underlying EC2 instances. This flexibility is crucial when I need to install custom software, fine-tune performance settings, or troubleshoot issues directly on the server.

Integration with Existing Components:

- **Current Project Involvement**:
    - **Flask Application**: My Flask application, with the directory structure including `app.py`, `requirements.txt`, `Procfile`, `static/`, `templates/`, and other files, is well-suited for deployment on Elastic Beanstalk. Elastic Beanstalk supports a variety of platforms, including Python, making it straightforward to deploy Flask applications.
    - **AWS Glue for Data Cleaning and Preprocessing**: As part of the handwritten text recognition app, I am using AWS Glue for data cleaning and preprocessing. This component can seamlessly integrate with your application hosted on Elastic Beanstalk, ensuring that the data pipeline remains efficient and effective.
    - **AWS Learners Lab**: Considering the constraints of your AWS Learners Lab environment, Elastic Beanstalk offers an ideal solution by simplifying deployment without the need for extensive IAM role configurations that we can't modify.

- **AWS Services and Tools**:
    - **AWS DynamoDB, RDS, or S3 for Databases**: I had considered using various AWS database services. Elastic Beanstalk can easily integrate with these services, providing secure and scalable storage solutions for the application data.
    - **Amazon Elastic Kubernetes Service (EKS)**: Although I explored using EKS, Elastic Beanstalk provides a more straightforward deployment path while still offering the ability to manage underlying instances if needed. [5]
    - **AWS VPC and API Gateway**: Utilizing VPC with Elastic Beanstalk can enhance the security of your application by isolating it within a private network. API Gateway can be used in conjunction with Elastic Beanstalk to expose the application's functionalities as RESTful APIs.

Implementation Steps:

1. **Prepare the Flask Application**:
    - Ensure the application follows the best practices for deployment on AWS Elastic Beanstalk, including having a `requirements.txt` for dependencies and a `Procfile` to specify the command to run the application.
2. **Deploy to Elastic Beanstalk**:
    - Create an Elastic Beanstalk environment for the Flask application using the AWS Management Console.
    - Upload the application code and configure the environment settings, such as instance type, scaling options, and environment variables.
3. **Configure EC2 Instances**:
    - Access the underlying EC2 instances if necessary to install custom software or apply specific configurations.
    - Use SSH to connect to the instances and perform any required administrative tasks.
4. **Monitor and Scale**:
    - Use the Elastic Beanstalk management console to monitor the health and performance of the application.
    - Configure auto-scaling policies to ensure the application can handle changes in traffic seamlessly.

4. Delivery Model

Hybrid Cloud (IaaS and PaaS)

Services Used:

1. **AWS Elastic Beanstalk (PaaS)**:
    - Simplifies the deployment and management of my application.
    - Handles infrastructure setup, load balancing, scaling, and health monitoring.
    - Example of a PaaS, as it abstracts much of the underlying infrastructure management, allowing to focus on the application itself.
2. **AWS EC2 (IaaS)**:
    - Provides granular control over the virtual servers where the application runs.
    - Allows to manage and configure the underlying compute resources directly.
    - Example of an IaaS, as it provides virtualized computing resources over the internet.

Integration with AWS API Gateway:

- **AWS API Gateway**:
    - Although not fitting perfectly into IaaS, PaaS, or SaaS, it can be considered a part of the broader PaaS ecosystem because it provides managed services for creating, deploying, and managing APIs. [6]
    - Integrates seamlessly with AWS services like Lambda (serverless compute, part of PaaS), EC2 (IaaS), and DynamoDB (managed NoSQL database, part of PaaS).

Summary:

- **IaaS (Infrastructure as a Service)**: The use of **AWS EC2** falls under IaaS, providing with virtualized infrastructure to run the applications and manage the underlying compute resources directly.
- **PaaS (Platform as a Service)**: Your use of **AWS Elastic Beanstalk** and **AWS API Gateway** falls under PaaS, offering you a platform to deploy, manage, and scale your applications without handling the underlying infrastructure complexities directly.

In summary, your deployment model leverages both IaaS and PaaS:

- **IaaS**: For direct control over infrastructure (EC2 instances).
- **PaaS**: For simplified application deployment and management (Elastic Beanstalk and API Gateway).

This hybrid approach combines the flexibility and control of IaaS with the simplicity and efficiency of PaaS, providing a robust and scalable solution for the application deployment and management needs.

5. Architecture Diagram

Architecture Diagram:

Description:

1. **User Interaction:**
    - Users interact with the system through a web interface where they can upload images.
2. **API Gateway:**
    - AWS API Gateway manages API requests and routes them to the appropriate backend service.
3. **Elastic Beanstalk:**
    - Deploys and manages the Flask application that handles image uploads and processing requests.
4. **EC2 Instance:**
    - Hosts the Flask application and processes the images.
5. **S3 Bucket:**
    - Stores the uploaded images and processed text data.
6. **Amazon Textract:**
    - Extracts text from the images stored in S3.
7. **AWS Glue:**
    - Performs data cleaning and processing on the extracted text.

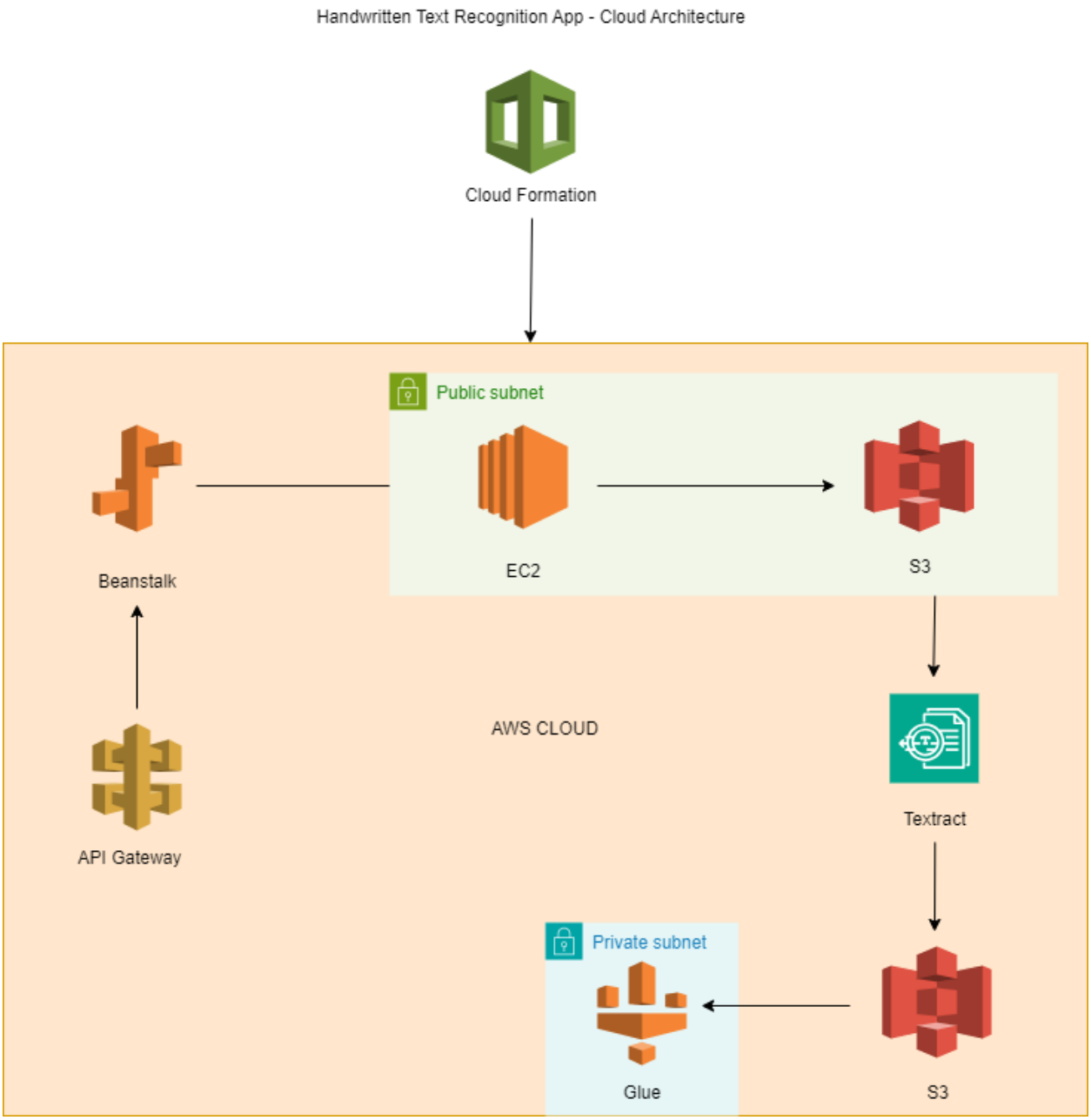

Figure 2 Architecture Diagram

6. Security Analysis

Approach to Security:

To ensure good security for the application, it is essential to protect data in transit, data at rest, and manage access effectively. Below is an elaboration on each aspect of the security approach:

Data in Transit:

1. **Use HTTPS for All Communication**:
    - **HTTPS (Hypertext Transfer Protocol Secure)** encrypts data sent between the client (user) and the server, preventing interception by unauthorized parties.
    - **Configuration**: Ensure that your application endpoints in AWS API Gateway enforce HTTPS by redirecting HTTP traffic to HTTPS. This can be done by setting up custom domain names in API Gateway.
2. **Enforce SSL/TLS with API Gateway**:
    - **SSL/TLS (Secure Sockets Layer/Transport Layer Security)** protocols are used to secure communication over a computer network.
    - **Implementation**: API Gateway can enforce the use of SSL/TLS by requiring clients to use HTTPS endpoints. This ensures that all data transmitted between users and the backend services is encrypted and secure.

Data at Rest:

1. **Use AWS S3 Server-Side Encryption**:
    - **Server-Side Encryption (SSE)** in S3 encrypts data at rest, protecting it from unauthorized access.
    - **Configuration**: Enable SSE for the S3 buckets, choosing an appropriate encryption method (SSE-S3, SSE-KMS, or SSE-C). SSE-S3 uses AES-256 encryption, while SSE-KMS integrates with AWS Key Management Service (KMS) for additional security controls. [7]
2. **Use VPC to Isolate EC2 Instances**:
    - **VPC (Virtual Private Cloud)** allows to create a secure network environment for the EC2 instances.
    - **Configuration**: Placed the EC2 instances within a VPC, utilizing subnets, security groups, and network ACLs (Access Control Lists) to control inbound and outbound traffic. This isolation helps protect your instances from unauthorized access and network threats. [8]

7. Cost Metrics

Analysis of Costs

Up-front Costs:

- **Minimal Initial Costs**: AWS provides a free tier for many services, which can help minimize up-front costs. This includes free usage for services like EC2, S3, and API Gateway within certain limits.
- **Initial Expenses**: The initial costs might include data storage in S3 and the creation of resources. These costs are typically low, especially if you stay within the AWS Free Tier limits.

On-going Costs:

- **EC2 Instance**:
  - Costs are based on the instance type and usage hours. For example, a t2.micro instance, which is eligible for the free tier, might be cost-effective for initial development.
- **S3 Storage**:
  - Charges are based on the amount of data stored and the number of requests (GET, PUT, etc.). Standard storage costs are around $0.023 per GB for the first 50 TB per month.
- **Textract**:
  - The cost is based on the number of pages processed. As of the latest pricing, the first 1,000 pages processed each month are free. Beyond that, it's $0.01 per page for OCR operations. [9]
- **Glue**:
  - AWS Glue charges are based on the amount of data processed and the duration of the jobs. Reducing the number of worker nodes and optimizing job configurations can significantly reduce costs. I have already decreased the worker nodes and adjusted other configurations to make Glue more cost-effective. [10]
- **Elastic Beanstalk**:
  - Costs depend on the underlying EC2 instances and other resources used. This includes charges for instances, load balancers, and any additional services like RDS for databases.

Additional Costs:

- **API Gateway**:
  - Charges are based on the number of API calls. The first 1 million API calls per month are free, and after that, it's $3.50 per million calls. [11]
- **Monitoring and Logging (CloudWatch)**:
  - Costs depend on the amount of data ingested and stored. Basic monitoring is free, but detailed monitoring and additional metrics/log storage incur costs. For example, custom metrics are charged at $0.30 per metric per month. [12]

Cost-saving Alternatives:

1. **Use Spot Instances for EC2**:
    - **Spot Instances** can significantly reduce compute costs by taking advantage of unused EC2 capacity at a reduced price. However, they come with the trade-off of potential termination by AWS if capacity is needed elsewhere.
    - This option is suitable for non-critical workloads where intermittent interruption is acceptable.
2. **Optimize S3 Storage Classes**:
    - Use **S3 Lifecycle Policies** to move data to cheaper storage classes (e.g., S3 Infrequent Access, S3 Glacier) based on access patterns. For example, frequently accessed data can remain in standard storage, while older, less accessed data can be moved to Infrequent Access or Glacier to save costs.
    - Implementing lifecycle policies helps automate the cost optimization process.
3. **Limit Textract Usage**:
    - Use **Amazon Textract** selectively for pages that require OCR. This involves pre-processing documents to identify which pages need text extraction and limiting Textract usage to those pages.
    - By carefully selecting which documents and pages to process, you can control and reduce Textract-related costs.
4. **Optimizing AWS Glue**:
    - **Decrease Worker Nodes**: Reducing the number of worker nodes in AWS Glue jobs helps lower costs. Fewer worker nodes mean less resource consumption and reduced overall expenses.
    - **Configuration Adjustments**: Other configurations, such as job duration limits and data partitioning, can be optimized to ensure efficient data processing without unnecessary resource usage.

Detailed Implementation of Cost-saving Measures:

- **Spot Instances for EC2**:
    - Implement spot instance strategies for workloads that can tolerate interruptions, such as batch processing or development/testing environments.
    - Use AWS Auto Scaling to manage spot instance availability and automatically replace terminated instances with new spot or on-demand instances. [13]
- **S3 Storage Optimization**:
    - Define and apply S3 lifecycle policies to automate the transition of objects between different storage classes based on age or access patterns.
    - Regularly review and adjust lifecycle policies to ensure they align with changing data usage patterns.
- **Textract Usage Limitation**:
    - Implement a pre-processing step to filter out pages that do not require OCR, using techniques such as basic text detection or manual tagging.
    - Schedule Textract processing during periods when it is most cost-effective, avoiding unnecessary real-time processing.
- **AWS Glue Configuration Optimization**:

- Continuously monitor Glue job performance and adjust worker node counts based on actual usage and performance metrics.
- Experiment with different job configurations to find the most cost-effective setup, balancing performance and cost.

By implementing these strategies, we can manage and optimize the costs associated with running your application on AWS. This approach ensures that we maintain a balance between performance and cost-effectiveness, leveraging AWS's flexible pricing models and cost-saving features.

Total Monthly Cost (average)

EC2 Monthly Cost=720hours×$0.0116per hour = $8.35

S3 Storage Cost=100GB×$0.023per GB = $2.30

Total S3 Requests Cost=$0.05+$0.04 = $0.09

Total S3 Monthly Cost=$2.30+$0.09 = $2.39

Textract Monthly Cost=(3,000−1,000)pages×$0.01per page=2,000pages×$0.01 = $20.00

Glue Monthly Cost=100hours×2DPUs×$0.44per DPU-hour = $88.00

Elastic Beanstalk EC2 Cost=2instances×720hours×$0.0116per hour=2×$8.35 = $16.70

API Gateway Cost=$0.00(within free tier)

Total cost = $135.44 ~

References

[1] “Amazon Textract,” Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/textract/ . [Accessed: 01-Jul-2024].

[2] “AWS Glue,” Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/glue/. [Accessed: 07-Jul-2024].

[3] “AWS Fargate,” Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/fargate/. [Accessed: 07-Jul-2024].

[4] “Hybrid cloud,” IBM. [Online]. Available: https://www.ibm.com/topics/cloud-deployment-models . [Accessed: 12-Jul-2024].

[5] “Amazon EKS,” Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/eks/. [Accessed: 03-Aug-2024].

[6] “IaaS, PaaS, and SaaS,” IBM. [Online]. Available: https://www.ibm.com/topics/iaas-paas-saas. [Accessed: 06-Aug-2024].

[7] “Amazon S3 Encryption,” Amazon Web Services, Inc. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/dev/serv-side-encryption.html. [Accessed: 17-Jul-2024].

[8] “Amazon VPC,” Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/vpc/. [Accessed: 18-Jul-2024].

[9] “Amazon Textract Pricing,” Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/textract/pricing/. [Accessed: 02-Aug-2024].

[10] “AWS Glue Pricing,” Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/glue/pricing/. [Accessed: 02-Aug-2024].

[11] “Amazon API Gateway Pricing,” Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/api-gateway/pricing/. [Accessed: 02-Aug-2024].**)**:

[12] “Amazon CloudWatch Pricing,” Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/cloudwatch/pricing/. [Accessed: 02-Aug-2024].

[13] “S3 Spot Instance”, Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/ec2/spot/ . [Accessed: 04-August-2024].