CSCI 5409 – ADVANCE TOPICS IN CLOUD COMPUTING
GOWRI PRASHANTH KANAGARAJ- B00942544

Sentiment Analysis System

Introduction

In the digital age, where data is the new currency, businesses and organizations generate and collect vast amounts of text data through various channels, including social media, customer reviews, and support tickets. This text data is a goldmine of insights but is often underutilized due to its unstructured nature. To convert this data into actionable insights, there's a critical need for efficient and automated sentiment analysis systems.

Project Description

The project aims to develop a cloud-based sentiment analysis system that can process and analyze text data, whether entered directly or extracted from images, to determine the sentiment behind the words. The system must be scalable to handle fluctuating volumes of data, secure to ensure the confidentiality and integrity of information, and cost-effective to offer a competitive edge. It is designed to analyze the sentiment (positive, negative, or neutral) of a given input, which could be text or an image with text. This real-time analysis can empower businesses to make data-driven decisions, improve customer experience, tailor marketing strategies, and address potential issues before they escalate. It serves not only as a tool for sentiment analysis but also as a strategic asset for reputation management and customer relationship enhancement.

Problem Statement

Organizations struggle to effectively interpret the sentiment of their textual data due to:

The sheer volume of unstructured text data generated daily.

The complexity of natural language, which requires sophisticated computational techniques to understand nuances.

The need for real-time analysis to respond to market trends and customer feedback promptly.

The high cost and complexity of deploying and maintaining sentiment analysis infrastructure.

The sentiment analysis system proposed here seeks to address these challenges by leveraging the power of cloud computing, machine learning, and serverless architectures. It must deliver real-time analysis, scale according to demand, and maintain a high security and compliance standard—all while managing operational costs effectively.

To meet the requirements of the sentiment analysis system, a selection of AWS services was utilized. Below is an overview of the services chosen, along with a comparison to alternative services and the reason for the selection:

Compute – Pick two (2):

• AWS Lambda
• AWS Elastic Beanstalk

Storage – Pick one (1):

• AWS S3

Network – Pick one (1):

• AWS API Gateway

General – Pick two (2):

• Amazon Textract

• Infrastructure as code: cloudformation along with ci/cd pipeline

Service Selection and Comparative Analysis for Sentiment Analysis System

AWS Lambda:

Chosen For: Running code in response to events (e.g., image/text uploads) without managing servers.

Alternative: Azure Functions or Google Cloud Functions.

Rationale: AWS Lambda is deeply integrated with other AWS services, such as S3 and Textract, which facilitates easier setup and management within the AWS ecosystem.

Amazon S3:

Chosen For: Storing images and possibly other documents due to its high durability and availability.

Alternative: Azure Blob Storage or Google Cloud Storage.

Rationale: S3 integrates seamlessly with AWS Lambda and Textract and provides robust security features and fine-grained access controls.

Amazon Textract:

Chosen For: Extracting text from images using OCR technology.

Alternative: Google Cloud Vision API or Microsoft Computer Vision.

Rationale: Textract is specifically tailored for text extraction and is part of the AWS suite, offering better integration and potentially lower latency when used with other AWS services.

Amazon API Gateway:

Chosen For: Creating, publishing, maintaining, monitoring, and securing APIs at any scale.

Alternative: Azure API Management or Google Cloud Endpoints.

Rationale: API Gateway offers native integration with AWS Lambda and other AWS services, making it an efficient choice for managing the APIs necessary for the system.

AWS CloudFormation:

Chosen For: Automated infrastructure provisioning using code.

Alternative: Terraform or Google Cloud Deployment Manager.

Rationale: CloudFormation is an AWS-native service, allowing for a more seamless and cohesive infrastructure-as-code experience within AWS environments.

AWS Elastic Beanstalk:

Chosen For: Deploying and scaling web applications and services.

Alternative: Azure App Service or Google App Engine.

Rationale: Elastic Beanstalk provides an easier deployment process for web applications within AWS and handles much of the management and scaling automatically.

AWS CodeBuild and AWS CodePipeline:

Chosen For: Continuous integration and continuous delivery (CI/CD) services to automate the software release process.

Alternative: Jenkins, CircleCI, or GitLab CI.

Rationale: These AWS services are fully managed, integrate with other AWS services, and remove the need to set up, manage, and scale your own build and release infrastructure.

Reason for Service Selection

The selection of AWS services over alternatives was primarily driven by the following factors:

Integration: AWS services provide tight integration with one another, which simplifies the architecture and streamlines data flow between services.

Management: AWS offers a unified console and set of tools for managing all the services, which can simplify operations and monitoring.

Security: AWS provides comprehensive security and compliance capabilities that adhere to security best practices and various compliance regulations.

Scalability: AWS services are designed to scale with the application demand automatically, which is crucial for handling varying loads with efficiency.

Ecosystem: By choosing AWS, you can benefit from a vast ecosystem of support, documentation, community, and add-ons.

Expertise: If your team has more experience with AWS, this can reduce the learning curve and speed up development and troubleshooting.

In summary, the AWS suite was chosen for its comprehensive set of integrated services, scalability, security, and the existing expertise within the team. Alternative services were considered but ultimately not selected due to the coherence and synergy provided by staying within the AWS ecosystem.

Deployment Model Description

In the sentiment analysis system, the chosen deployment model is based on a public cloud infrastructure, specifically utilizing AWS (Amazon Web Services) as the cloud service provider. An Automated Continuous Deployment model using AWS services is employed. This model enables code to be deployed automatically to production after passing through stages of automated testing and quality checks.

Components of the Deployment Model:

AWS CodePipeline:

Orchestrates the workflow from code commit to deployment.

Automates the build, test, and deploy phases each time there is a code change, based on the defined rules.

AWS CodeBuild:

Compiles source code, runs tests, and produces ready-to-deploy software packages.

Integrated with CodePipeline to automate the build and test phase.

AWS Elastic Beanstalk:

Automates the deployment of applications, including provisioning of resources like EC2 instances, balancing load, and auto-scaling.

Manages application deployment, from capacity provisioning, load balancing, auto-scaling to application health monitoring.

AWS CloudFormation:

Manages and provisions resources through Infrastructure as Code.

Ensures consistent environment setup for development, testing, and production.

Amazon S3:

Stores build artifacts and serves as a deployment source for CodePipeline and Elastic Beanstalk.

Reason for Choosing Automated Continuous Deployment

Speed and Efficiency: Automated deployments significantly reduce the time to get from code commit to production, enabling a faster response to market changes and user feedback.

Reliability: Automation reduces the risk of human error during deployment. It ensures that every deployment is performed in a consistent manner.

Scalability: The chosen model scales effortlessly with the system's usage patterns and demand without manual intervention.

Repeatability: CloudFormation templates ensure that environments are provisioned consistently every time, which is critical for maintaining stable and predictable release cycles.

Rollback Capabilities: In case of a deployment issue, automated rollbacks are possible to the last known good state, minimizing downtime.

Resource Optimization: By leveraging AWS Elastic Beanstalk, the system automatically manages the underlying resources, optimizing for cost and performance.

The continuous deployment model is ideal for a cloud-based sentiment analysis system where features and updates are expected to be rolled out regularly and reliably. This approach supports agile development practices, enabling frequent and incremental changes to the application without compromising on quality or availability.

Delivery Model Description

For the sentiment analysis system, the chosen delivery model revolves around a Platform as a Service (PaaS) approach, specifically leveraging a Serverless Architecture within the AWS (Amazon Web Services) ecosystem. This delivery model offers a streamlined and efficient way to deploy, manage, and scale applications without the need for infrastructure management tasks. The delivery model chosen for the sentiment analysis system is a Serverless Architecture model utilizing managed services within the AWS ecosystem.

Components of the Delivery Model

AWS API Gateway:

Serves as the entry point for the APIs that trigger the backend services.

Manages traffic, authorization, access control, and monitoring.

AWS Lambda:

Executes code in response to triggers (such as HTTP requests from API Gateway) without the need to manage servers.

Scales automatically by running code in response to each trigger.

Amazon S3:

Provides object storage services that store the incoming data and the artifacts of the system.

Amazon Textract:

Processes the images uploaded to S3 to extract text using OCR technology.

Amazon CloudWatch:

Monitors the system, providing logging and metrics, which enables observing the operational health and performance.

Rationale for Choosing Serverless Architecture

Cost-Effectiveness: Serverless architecture minimizes operational costs as you only pay for the compute time you consume, with no charge when your code is not running.

Simplicity and Speed: Eliminates the need for infrastructure management, speeding up the deployment process and allowing the team to focus more on development rather than operations.

Scalability: Automatically scales the computational resources in response to the incoming request volume without any manual intervention.

High Availability and Fault Tolerance: Managed services like AWS Lambda and S3 are designed for high availability and fault tolerance out of the box.

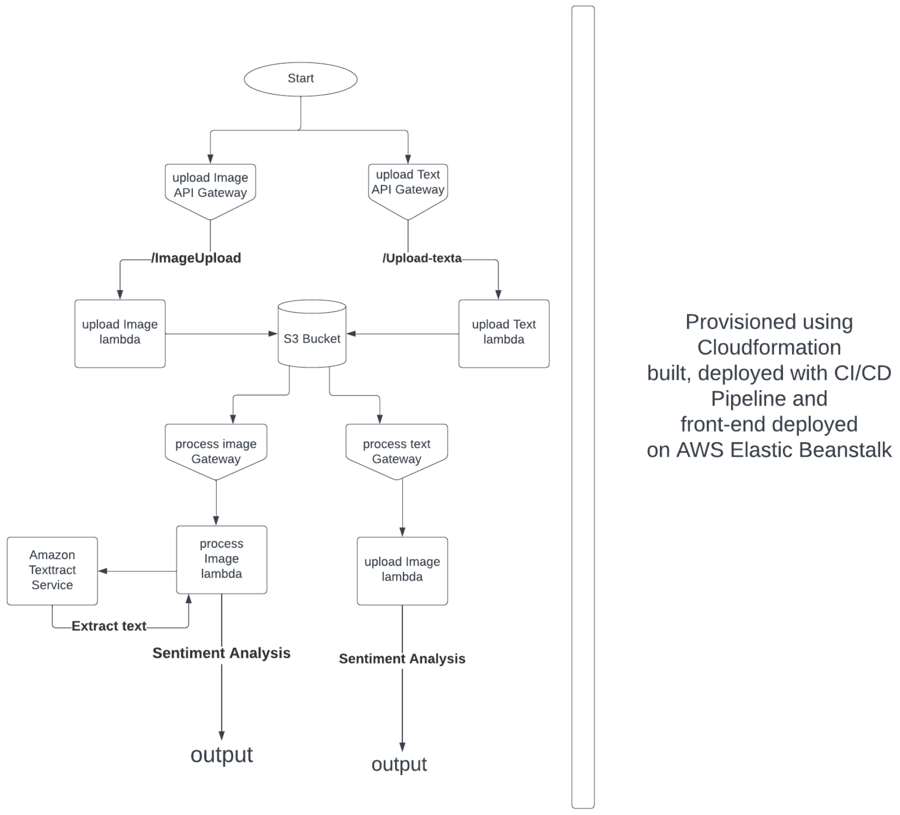
Operational Management: AWS handles the heavy lifting of server and system maintenance, including capacity provisioning, automatic scaling, patching, and security.

Flexibility: Serverless architecture offers the flexibility to quickly update, add, or remove features as business needs change or evolve.

The serverless delivery model aligns with the objectives of building a resilient, scalable, and cost-effective system. It supports the sentiment analysis system's need for handling variable workloads and simplifies the operational overhead. This approach also ensures that the system can be updated with minimal downtime and facilitates rapid feature development and deployment.

System Architecture:

Cloud-based sentiment analysis system designed to process both image and text inputs.



Start: The entry point for the system, where users begin their interaction.

API Gateway:

Upload Image API Gateway: Receives image files uploaded by the user.

Upload Text API Gateway: Receives text data uploaded by the user.

AWS Lambda Functions:

Upload Image Lambda: Triggered by the image upload API, this function is responsible for storing the uploaded image in an Amazon S3 bucket.

Upload Text Lambda: Triggered by the text upload API, this function presumably processes the text data directly, preparing it for sentiment analysis.

Amazon S3 Bucket:

Stores uploaded images before they are processed by Amazon Textract.

Image and Text Processing:

Process Image Gateway: An API endpoint that triggers the Process Image Lambda function.

Process Text Gateway: An API endpoint that likely invokes a Lambda function for the initial processing of the uploaded text.

Amazon Textract:

Utilized by the Process Image Lambda to extract text from the images stored in S3. The extracted text is then presumably sent for sentiment analysis.

Sentiment Analysis:

Conducted by additional Lambda functions (not explicitly shown) that receive the processed text or the text extracted from images.

The sentiment of the text is analyzed, potentially using the TextBlob Python library, as mentioned in your initial explanation.

Output:

The result of the sentiment analysis is outputted. This would typically be a classification of the sentiment as positive, negative, or neutral.

Provisioning and Deployment:

CloudFormation: Used to define and provision the AWS infrastructure resources needed for the system, ensuring consistent and repeatable deployments.

CI/CD Pipeline: Continuous Integration and Continuous Deployment processes are implemented, likely using AWS CodeBuild and AWS CodePipeline, to automate the deployment of updates to the system.

AWS Elastic Beanstalk: Deploys and manages the application's front-end, which interacts with the backend AWS infrastructure. It is responsible for handling the provisioning of EC2 instances and other necessary resources for the web application.

Security Analysis of the Sentiment Analysis System

API Gateway: Serves as the front door to the system's APIs, providing a first layer of defense. It enforces HTTPS to ensure that all data transmitted between the clients and the system is encrypted using TLS (Transport Layer Security). This encryption protects the data from eavesdropping, tampering, and message forgery.

AWS Lambda: When Lambda functions are invoked by API Gateway, the data remains within the secure AWS network, which reduces the exposure to potential transit-based attacks.

Amazon S3: S3 provides robust options for securing data at rest, including server-side encryption with Amazon S3-managed keys (SSE-S3), AWS Key Management Service (KMS) managed keys (SSE-KMS), or customer-provided keys (SSE-C). Additionally, S3's access control mechanisms, such as bucket policies and IAM policies, restrict unauthorized access to the data stored within.

Amazon Textract: As a fully managed service, Textract handles data at rest security by default. However, it's crucial to ensure that the IAM roles and policies associated with Textract access are configured to follow the principle of least privilege, granting only the necessary permissions.

IAM (Identity and Access Management): The system should utilize IAM to manage access to AWS services and resources securely. This includes setting up IAM roles with fine-grained permissions for Lambda functions and enforcing strict access policies to ensure that only authorized entities can access the necessary resources.

AWS CloudFormation: Ensures that the infrastructure is provisioned with security best practices by defining and deploying all resources with security configurations in code.

Data Backup: Regular backups should be taken for critical data to ensure it can be recovered in case of accidental deletion or corruption.

Disaster Recovery: The system should have a disaster recovery plan in place, utilizing AWS services such as S3 versioning or cross-region replication.

In summary, the project's approach to security encompass a multi-layered strategy that includes encryption, access control, secure data transmission, monitoring, and compliance adherence. It ensures that all aspects of the system, from the API Gateway down to data storage, are configured to meet the highest security standards.

Cost Metrics Analysis for the Sentiment Analysis System

The operating costs of the sentiment analysis system are influenced by several factors and AWS service pricing models. Here's a breakdown of the cost metrics:

Upfront Costs:

Infrastructure Provisioning: AWS CloudFormation has no upfront costs, but there might be costs associated with the resources it provisions.

Development Costs: Initial development, including labor costs for architects, developers, and security specialists.

Ongoing Costs:

AWS Lambda: Priced per 1 million requests and the duration of the code execution time. Costs will increase with the number of invocations and the execution time.

Amazon API Gateway: Charged per million API calls received, plus data transfer costs.

Amazon S3: Costs are based on the amount of data stored per month and the number of GET, PUT, POST, and LIST requests.

Amazon Textract: Priced per page processed for text detection and analysis.

AWS Elastic Beanstalk: Underlying EC2 instances, EBS volumes, and other resources determine the cost.

Data Transfer: Data transfer out of AWS services to the internet can incur costs after the initial free tier limit.

Additional Costs:

Monitoring and Logging: Services like Amazon CloudWatch have free tiers, but detailed monitoring, additional metrics, and log data storage and transfer can incur additional costs.

Backup and Recovery: The costs associated with S3 data backups, including storage and data transfer for the backups.

Cost-Saving Alternatives:

Serverless Architecture: Already a cost-effective approach due to its pay-as-you-go model, ensuring you only pay for the resources you consume.

Reserved Instances: For services like EC2, purchasing Reserved Instances can provide a discount compared to on-demand pricing.

AWS Free Tier: Leveraging AWS Free Tier for development and testing could reduce initial costs.

Scaling Down Resources: Using smaller instances or fewer resources during off-peak times can save costs.

Data Transfer Optimization: Optimizing data transfer by using content delivery networks (CDNs) like Amazon CloudFront to reduce data transfer out costs.

Justification for More Expensive Solutions:

Performance: Choosing a more costly solution may be justified if it results in significantly better performance, reliability, or speed for end-users.

Security and Compliance: Additional costs for enhanced security measures or to ensure compliance with regulations like GDPR, HIPAA, or CCPA are often justified and sometimes non-negotiable.

Scalability: A more expensive architecture might be justified if it provides better scalability to handle growth without performance degradation.

In real-world scenarios, costs can be optimized by closely monitoring the usage patterns and adjusting the provisioned resources accordingly. Continuous assessment and optimization of the architecture can lead to significant cost savings over time.

Future Works:

The current sentiment analysis system sets a strong groundwork for future enhancements. Plans include integrating more sophisticated natural language processing techniques to improve accuracy, expanding language support, and implementing real-time analytics for instantaneous sentiment assessment. User interface improvements, better data privacy measures, and broader integration capabilities are also on the roadmap. Additionally, optimizing scalability and resource management will be a focus to ensure cost-efficiency and performance.

Conclusion:

The system serves as a dynamic tool for extracting actionable insights from unstructured text data, providing businesses with a deeper understanding of customer sentiment. Utilizing a serverless architecture on AWS, the system promises scalability, security, and cost-effectiveness. As the system evolves, it will continue to empower businesses with real-time analytics and a comprehensive suite of text analysis tools, solidifying its role as an essential asset in the data-driven decision-making process.