DALHOUSIE UNIVERSITY

CSCI 5411

Advanced Cloud Architecting

Term Assignment

by

Tijilkumar Parmar
B00953583
tj950701@dal.ca

Submitted to
Prof. Lu Yang
Department of Computer Science
Dalhousie University.

Table of Contents

Superstore Employee Management System (Crumb - Commander)

The Superstore Employee Management System (CrumbCommander) is a robust web application developed using Flask and React, designed to streamline the management of employees, shifts, departments, and associated administrative tasks. This system provides an intuitive interface for managing operational and administrative activities, aiming to enhance the efficiency of retail stores. With its comprehensive features, it supports various user roles, including admin users, department managers, and employees, each with specific functionalities tailored to their needs.

Project Details

- **Original GitHub Link:** Crumb Commander
  I have used my own project for this term assignment as I was passionate about completing this and providing a real solution to the problem faced by my colleagues at my part time job.
- **Hosted Link:** Hosted Application

Purpose and Functionality

Purpose

The primary purpose of CrumbCommander is to facilitate seamless management of employee schedules, attendance, and administrative tasks. It aims to simplify and automate various operational functions within a retail environment, making it easier for administrators, managers, and employees to interact with and manage work-related information. The app focuses a bit more on the Bakery utility, but those are performed locally, the AWS services are majorly used for administrative services.

Functionality

The application was inspired by me coming across the process followed by the managers to manage employee shifts at my part time. The process was manual and hence the weekly schedules were distributed to different departments by handing out printouts. When I joined this store as a part time employee, I could not bear the idea of following these outdated methods being a software engineer. This is when I decided to work on this project, and this term gave me the perfect opportunity to develop and fine-tune the same.

Target Users

CrumbCommander caters to a diverse range of users, including:

- **Admin Users:** Retail store administrators responsible for overseeing the entire system, managing departments, employees, and handling administrative tasks.
- **Employees:** Staff who need to view their attendance and shifts.

Performance Targets

CrumbCommander aims to meet the following performance targets:

- **User Experience:** Ensure a smooth and responsive user interface with minimal latency in data retrieval and updates.
- **Scalability:** Design the system to handle a growing number of users and data without compromising performance.

- **Reliability:** Achieve high system uptime with minimal downtime and efficient error handling.
- **Data Security:** Implement robust security measures to protect sensitive employee and operational data.
- **Accuracy:** Maintain accurate and up-to-date records of employee attendance, shifts, and other operational metrics.

AWS Service Selection

For the CrumbCommander application, I have carefully selected AWS services to fulfill the compute, storage, network, and general requirements, ensuring the efficient operation and scalability of the application.

Compute

Amazon EC2 Instances:

- **Performance:** EC2 instances offer flexible and scalable compute capacity. This allows the Flask application to handle varying workloads efficiently, ensuring smooth operation even during peak usage.
- **Cost:** EC2 pricing is based on instance types and usage, providing cost-effective solutions. By selecting appropriate instance types and leveraging Reserved Instances or Spot Instances, costs can be optimized.
- **Scalability:** Auto Scaling Groups and Scaling Policies automatically adjust the number of EC2 instances based on load, ensuring the application scales up or down as needed, which helps manage costs and performance dynamically.

Database

Amazon RDS with MySQL:

- **Performance:** RDS provides managed MySQL databases with automated backups, patching, and performance tuning. This ensures high availability and performance without manual intervention.
- **Cost:** RDS offers various pricing models, including On-Demand and Reserved Instances, allowing cost management based on the application's needs. Automated backups and snapshots reduce the need for additional backup solutions.
- **Security:** RDS is configured within a dedicated subnet group in the VPC, ensuring isolation from public access and enhancing security. It also supports encryption and other security measures to protect sensitive data.
- **Scalability:** RDS allows for easy scaling of database instances and storage, accommodating growing data needs without significant manual intervention.

Networking

Amazon VPC:

- **Performance:** VPC provides isolated network environments, allowing for controlled and optimized network traffic. Public and private subnets help manage communication between EC2 instances and RDS efficiently.
- **Security:** The VPC setup includes private subnets and a NAT Gateway, which enhances security by isolating sensitive components like the RDS database from direct internet access. Route Tables manage traffic flow, ensuring secure communication.
- **Cost:** Using a VPC with NAT Gateway incurs additional costs, but it is essential for secure networking. The benefits in security and control outweigh the costs for sensitive applications.

Security

Security Groups:

- **Performance:** Security Groups act as virtual firewalls, controlling inbound and outbound traffic to EC2 instances and RDS databases. They ensure that only necessary traffic is allowed, maintaining performance while securing the system.
- **Security:** Security Groups provide fine-grained control over network access, enhancing overall application security. They are configured to allow only required traffic such as SSH, HTTP, and MySQL connections.
- **Cost:** There is no additional cost for Security Groups, making them a cost-effective way to enhance security.

AWS Secrets Manager:

- **Security:** Secrets Manager securely stores sensitive information like database credentials and AWS keys. It provides automatic rotation of secrets and secure access management, reducing the risk of data breaches and unauthorized access.
- **Cost:** Secrets Manager incurs additional costs based on the number of secrets and API calls, but the benefits in security and management justify the expense.

Storage

Amazon S3**:**

- **Performance:** S3 provides scalable and durable storage for static files, media, and backups. Its high availability and redundancy ensure data reliability and quick access. We have used it as the initial position of our lambda function that emails the user's about their newly added shift.
- **Cost:** S3's pricing is based on storage used and data transfer, offering cost-effective solutions for large amounts of data. Features like lifecycle policies help manage storage costs effectively.
- **Scalability:** S3 automatically scales to handle increasing data volumes, making it a suitable choice for growing applications.

Management

AWS CloudFormation:

- **Performance:** CloudFormation automates the deployment and management of infrastructure, ensuring consistent and repeatable setups. This reduces manual configuration errors and improves deployment efficiency.
- **Cost:** There is no additional cost for CloudFormation itself, but it helps manage infrastructure resources efficiently, potentially reducing overall costs through automation.

Comparison of Alternative Services

- **Compute:** Amazon EC2 provides flexible and scalable compute resources, better suited than alternatives like Google Compute Engine due to its extensive instance options and integration capabilities.
- **Database:** Amazon RDS with MySQL offers robust management features and seamless AWS integration, surpassing Google Cloud SQL with its automated backups and strong security.
- **Networking:** Amazon VPC delivers superior network isolation and security, with more flexible configurations compared to Google Cloud or Azure.

- **Security:** AWS Security Groups and Secrets Manager provide comprehensive security and easy integration, outperforming alternatives in managing traffic and sensitive data.
- **Storage:** Amazon S3 excels in scalability and integration for static files and media, compared to Google Cloud Storage or Azure Blob Storage.
- **Management:** AWS CloudFormation offers effective infrastructure management through code, with deep AWS integration, making it preferable to tools like Terraform.

CloudFormation Implementation

For CrumbCommander, I implemented AWS CloudFormation to automate and manage the infrastructure setup:

- **Templates:** Created CloudFormation templates to define resources such as EC2 instances, RDS databases, VPC configurations, and S3 buckets, ensuring consistency and reproducibility across environments.
- **Stacks:** Used CloudFormation stacks to deploy and manage these resources as a single unit, simplifying updates and rollbacks while maintaining infrastructure as code.
- **Automation:** Integrated CloudFormation scripts into CI/CD pipelines, enabling automated infrastructure provisioning with each deployment, which streamlined setup and reduced manual errors.
- **Parameterization:** Utilized parameterized templates to customize deployments based on different environments (development, staging, production), enhancing flexibility and scalability.

Deployment Model

The deployment model for CrumbCommander utilizes a public cloud infrastructure provided by Amazon Web Services (AWS). In this model, all application components and resources are hosted and managed within the AWS cloud environment.

- **Reliability and Uptime:** Leveraging AWS ensures 99.999% uptime with its SLA, guaranteeing high availability for CrumbCommander.
- **Scalability:** The public cloud enables effortless scaling of computing resources and storage capacity based on demand, ensuring optimal performance.
- **Global Reach:** With AWS's global infrastructure, CrumbCommander users worldwide experience low latency and high performance.
- **Security:** AWS's comprehensive security features and compliance certifications protect CrumbCommander against unauthorized access and data breaches.
- **Cost-Effectiveness:** AWS's pay-as-you-go pricing model eliminates upfront capital investment, optimizing spending as CrumbCommander scales efficiently.

Delivery Model Description

In deploying CrumbCommander, I have employed a combination of Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) models provided by Amazon Web Services (AWS).

IaaS (Infrastructure as a Service)

- Amazon EC2**:** I utilized Amazon EC2 for flexible cloud server provisioning in CrumbCommander. EC2 hosts the Flask Application.

Reasons for choosing IaaS:

- **Flexibility:** Customize and configure virtual servers and database instances.
- **Control:** Retain control over OS, runtime environment, and infrastructure components.
- **Scalability:** Easily scale up or down based on demand for optimal performance and resource utilization.

PaaS (Platform as a Service)

- Amazon RDS: In addition to IaaS, I have incorporated Amazon RDS (Relational Database Service) within CrumbCommander. Amazon RDS offers managed relational database services, simplifying the setup, operation, and scaling of MySQL databases.

Reasons for Choosing PaaS:

- Simplified development and deployment with abstracted infrastructure.
- Pay-per-use pricing model for cost-efficiency.
- Automatic scaling for enhanced responsiveness and reliability.

CrumbCommander as a SaaS Solution

CrumbCommander embodies the core principles of SaaS, offering users a comprehensive employee management solution accessible through a web browser. Here's how CrumbCommander aligns with the SaaS model:

- **Accessible Collaboration:** Manage employee schedules from anywhere with internet access, facilitating seamless teamwork across distributed teams, regardless of location or time zone.
- **Scalability and Adaptability:** Tailored solutions to suit businesses of any size, ensuring CrumbCommander evolves with your management needs without requiring additional infrastructure investments.
- **Flexible Subscription Model:** Subscription-based pricing allows me to pay for needed features and resources without upfront costs, providing cost predictability and accommodating various budget constraints.
- **Automated Maintenance:** Benefit from automatic updates and maintenance by the SaaS provider, ensuring the platform remains secure, reliable, and up-to-date without burdening users with maintenance tasks.

Final Architecture Overview

The following figure summarizes the architecture followed for this project.

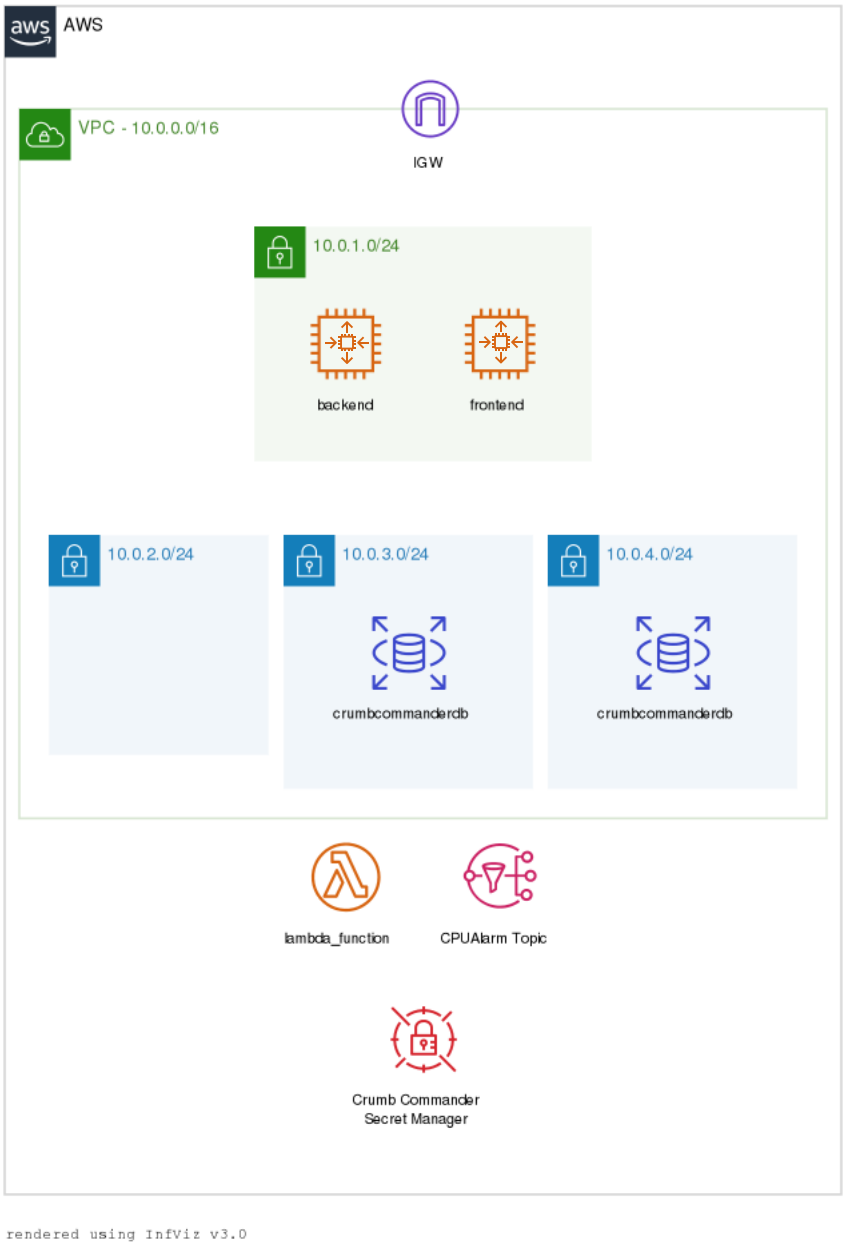

Figure 1: Crumb-Commander's Cloud Architecture

The architecture of CrumbCommander utilizes a combination of AWS services to deliver a scalable, reliable, and efficient solution for employee management.

VPC

Provides isolated networking resources for CrumbCommander, customizable with IP address ranges, subnets, routing tables, and network gateways.

Subnets

Divided into public and private across multiple Availability Zones (AZs), public for internet access (EC2), private for backend services (RDS MySQL).

Internet Gateway
Enables communication between VPC instances and the internet, facilitating outbound traffic from public subnets.

Security Groups
Control inbound/outbound traffic to EC2 and RDS instances based on protocol, port, and IP addresses.

RDS MySQL Database
Provisioned across private subnets for data storage, ensuring security with a subnet group.

EC2 Instance
Hosts Crumb Commander in a public subnet, associated with a public security group allowing HTTP, HTTPS, and SSH traffic.

Secrets Manager
Stores sensitive data like database credentials securely.

CloudWatch Alarms
Monitor EC2 metrics, triggering SNS notifications if CPU utilization exceeds 50%.

Dynamic Scaling Architecture

In CrumbCommander, I have implemented a dynamic scaling architecture to ensure optimal performance and availability:

- **Automatic Scaling:** Configured the system so that when an instance's CPU or memory usage reaches 75%, it automatically triggers the launch of a new instance. This ensures that additional capacity is provided as needed to handle increased traffic.
- **Instance Provisioning:** The configuration of the Flask project with NGINX for application serving and Nginx as a reverse proxy, so each instance is quickly ready to handle requests.

This approach allows me to maintain efficient resource utilization and high system performance under varying loads.

Architecting Principles

In designing the architecture for CrumbCommander, I have adhered to key architecting principles and best practices to ensure that the system is robust, scalable, and maintainable. Here's how my design aligns with these principles:

1. Scalability

- **Auto Scaling Groups:** Implemented Auto Scaling Groups with EC2 instances to handle varying loads by automatically adjusting the number of instances based on traffic. This aligns with the principle of horizontal scaling to meet demand.
- **Amazon RDS Scaling:** Use Amazon RDS to support both vertical scaling (by upgrading instance types) and horizontal scaling (through read replicas) to manage increasing database workloads. This ensures the system remains performant as user demand grows.

2. Availability and Fault Tolerance

- **High Availability:** Configured EC2 instances and RDS databases within a VPC with private subnets and high-availability features. For RDS, use Multi-AZ deployments to enhance availability and fault tolerance.
- **Automated Backups:** Enabled automated backups and snapshots for RDS, ensuring that data can be restored in case of failure or data loss, which supports effective disaster recovery plans.

3. Security

- **Network Isolation:** Configured the VPC with public and private subnets to isolate sensitive components like the RDS database from direct internet access, enhancing overall security.
- **Security Groups:** Use Security Groups as virtual firewalls to control traffic to and from EC2 instances and RDS databases. This approach adheres to the principle of least privilege by allowing only necessary traffic.
- **Secrets Management:** Rely on AWS Secrets Manager to securely store and manage sensitive information such as database credentials, which reduces the risk of unauthorized access and supports secure management of secrets.

4. Cost Efficiency

- **Right-Sizing Instances:** Select appropriate EC2 instance types and use Auto Scaling to optimize compute costs based on actual demand, preventing over-provisioning. Apart from this, also used CPU alarms which would makes sure to alert me with immediate email whenever the EC2 usage surpasses 50%. This would not allow the instance to infinitely auto-scale up and gives me a chance to understand the usage and allow auto scaling up.
- **Storage Optimization:** Leverage S3's cost-effective pricing model and lifecycle policies to manage data, optimizing storage costs and avoiding unnecessary expenses.
- **Pay-As-You-Go:** Take advantage of AWS services' pay-as-you-go pricing, which allows managing costs effectively based on usage and avoid fixed costs.

5. Performance Efficiency

- **Managed Services:** Use managed services like Amazon RDS and S3, which handle performance tuning and optimization, allowing focusing on improving application performance rather than managing infrastructure.
- **Network Configuration:** Configure the VPC with NAT Gateways and Route Tables to optimize network performance by efficiently managing traffic flow and ensuring secure, reliable communication between components.

6. Automation and Management

- **CloudFormation:** Automate infrastructure deployment with CloudFormation, which ensures consistent and repeatable setups. This reduces manual configuration errors and enhances overall manageability.

7. Modularity and Maintainability

- **Separation of Concerns:** Design the architecture to separate compute, storage, and database components into distinct services, following a modular design principle. This modularity enhances maintainability and allows for independent scaling and management of each component.

- **Documentation and Infrastructure as Code:** Use CloudFormation for infrastructure management and maintain well-documented architecture, which supports ease of updates and overall maintainability.

8. Disaster Recovery and Backup

- **Data Backups:** Ensure robust disaster recovery processes by utilizing RDS automated backups and storing logs and backups in S3. This provides effective data recovery capabilities.
- **Elasticity:** Employ Auto Scaling and managed services to ensure the application can adapt to failures and recover quickly, supporting resilience in case of issues.

By following these principles and best practices, I have created an architecture that balances scalability, availability, security, and cost efficiency, providing a solid foundation for the CrumbCommander application.

Architecture Pillars

In designing the architecture for CrumbCommander, I have focused on leveraging various AWS Well-Architected Framework pillars to create a robust and efficient system. Here's how each pillar is addressed:

1. Operational Excellence

- **CloudFormation:** Utilized CloudFormation to automate the deployment and management of infrastructure components. This approach ensures consistent and repeatable setups, simplifies operations, reduces manual configuration errors, and enhances overall system manageability.

2. Security

- **Amazon VPC:** Configured Amazon VPC with public and private subnets to isolate the RDS database from direct internet access. This setup enhances security by controlling network access and protecting sensitive data.
- **AWS Secrets Manager:** Use AWS Secrets Manager to securely store and manage sensitive information, such as database credentials and AWS keys. This practice ensures the safe handling of secrets and reduces the risk of unauthorized access.
- **Security Groups:** Set up Security Groups to act as virtual firewalls, controlling traffic to and from EC2 instances and RDS databases. This configuration enforces security policies and allows only necessary traffic, enhancing the overall security of the system.

3. Reliability

- **Auto-Scaling:** Implemented Auto-Scaling for EC2 instances to automatically adjust the number of instances based on application load. This ensures that the application remains reliable under varying traffic conditions by dynamically scaling resources as needed.

4. Performance Efficiency

- **Amazon RDS and S3:** Leveraged Amazon RDS and S3 to provide high-performance, managed database and storage solutions. RDS offers automated performance tuning and scaling, while S3 provides scalable and durable storage. These services enhance performance efficiency by offloading management tasks to AWS.

5. Cost Optimization

- **Auto-Scaling:** Use Auto-Scaling to optimize costs by adjusting the number of EC2 instances based on demand. This prevents over-provisioning and ensures that resources are used efficiently, aligning with cost optimization principles.
- **Amazon S3:** Chose Amazon S3 for cost-effective storage solutions with a pay-as-you-go pricing model. This approach allows for efficient management of storage costs, and lifecycle policies help further reduce expenses by managing data retention.

6. Sustainability

- **Auto-Scaling:** Incorporated Auto-Scaling to ensure that resources are used only when needed. This approach helps reduce the energy footprint associated with idle or over-provisioned resources, contributing to sustainability.
- **Amazon S3 and RDS:** By utilizing Amazon S3 and RDS, supporting environmental sustainability through optimized resource usage and reduced energy consumption compared to on-premises alternatives.

By addressing these AWS pillars, I have ensured that CrumbCommander is secure, reliable, cost-effective, performance-efficient, and sustainable, adhering to best practices for modern cloud-based applications.

Cloud Mechanisms for Crumb Commander

Flask Application (EC2 Instance):

- Hosted the Flask application on an EC2 instance within a public subnet.
- Configured security groups to allow inbound traffic on specific ports required for application functionality (e.g., SSH for administration, custom ports for API communication).

MySQL RDS Database:

- Deployed a MySQL RDS instance within a private subnet to securely store application data.
- Defined a subnet group to ensure the RDS instance is placed within the private subnet.
- Configured security groups to allow inbound traffic only from the Flask application instances, restricting access from the public internet.

VPC (Virtual Private Cloud):

- Created a VPC to isolate and control the networking environment for the application.
- Defined public and private subnets across multiple Availability Zones (AZs) for high availability and fault tolerance.
- Ensured proper routing and internet gateway configuration to enable communication between public and private resources while maintaining security.

Security Groups:

- Defined security groups to control inbound and outbound traffic to EC2 instances, RDS database, and other resources.
- Configured security group rules to allow necessary communication between frontend, backend, and database components while restricting access based on the principle of least privilege.

AWS Secrets Manager:

- Stored sensitive information such as database credentials securely in AWS Secrets Manager.
- Granted necessary permissions to the Flask application to retrieve secrets securely during runtime.

Data Storage:

Data is primarily stored in the MySQL RDS database instance. Here's how data storage is managed within the infrastructure:

MySQL RDS Database:

- The core of the data storage solution is the MySQL RDS database instance.
- The RDS instance is provisioned within a private subnet of the Virtual Private Cloud (VPC) to ensure data isolation and security.
- The database instance stores all application data related to employee accounts, departments, shifts, attendance, and any other relevant information for CrumbCommander.
- Data is organized into tables within the MySQL database, with each table representing a different aspect of the application's data model (e.g., employees, departments, shifts).
- The RDS instance is managed by AWS, providing features such as automated backups, point-in-time recovery, and high availability.

Secrets Manager:

- Sensitive information such as database credentials (e.g., email, password) is stored securely in AWS Secrets Manager.
- Secrets Manager ensures that sensitive data is encrypted at rest and during transit, providing an additional layer of security for access credentials.

System (Product) Deployment

Backend Deployment:

- **NGINX Integration:** To manage incoming user requests, set up NGINX as a reverse proxy. NGINX ensures efficient load balancing and reliable request handling.

Data Security Overview

The application architecture implemented using CloudFormation incorporates several mechanisms to ensure data security at all layers:

- **Network Isolation:** VPC separates resources into public and private subnets, with EC2 instances in public subnets and MySQL in private, restricting external access.
- **Encryption:** Data at rest and in transit encrypted via Amazon RDS for MySQL and HTTPS protocols.
- **Access Control:** Security Groups, IAM roles restrict resource access; granular permissions for EC2.
- **Data Masking and Redaction:** Sensitive data stored securely with AWS Secrets Manager, encrypting and restricting access.

Network Security

- **VPC:** Segregated network with public/private subnets.

- **Security Groups:** Firewalls for EC2/RDS, strict traffic rules.
- **Private Database Subnet:** MySQL isolated from internet access.

Encryption

- **Data Encryption (Rest):** Amazon RDS encrypts MySQL data at rest, ensuring confidentiality and blocking unauthorized access.
- **HTTPS Protocol:** Enforced for user-app communication, securing data transmission via SSL/TLS encryption, avoiding eavesdropping.

Access Control

- **IAM Roles:** Implemented least privilege access for EC2, limiting actions to necessary resources.
- **Secrets Management:** Leveraged AWS Secrets Manager for secure storage of sensitive data, with encrypted secrets and fine-grained access control.

Cloud Mechanisms and Cost Considerations

MySQL RDS Database Instance

- The MySQL RDS database instance serves as the backbone of our application's data storage and management.
- Costs associated with the RDS instance primarily depend on instance type, storage size, and provisioned IOPS.
- Continuous monitoring of database usage metrics, such as CPU utilization, storage capacity, and data transfer rates, is essential to identify potential cost spikes or inefficiencies.
- Proactive monitoring allows for cost-effective resource management.

Future Evolution of Application

Continuing the development of CrumbCommander could involve several feature enhancements and improvements. Here are some potential features to consider for future development and the corresponding cloud mechanisms to implement them:

Mobile Application Development

- Develop mobile applications for iOS and Android platforms to access CrumbCommander on the go.
- Utilize AWS Amplify for building and deploying mobile applications with serverless backends.
- Introduce automated testing pipelines to ensure code quality and reliability.

Advanced Analytics and Reporting

- Provide insights into employee performance, attendance rates, and shift management efficiency.
- Utilize AWS Athena for SQL querying of data stored in S3 for analytics.
- Implement scheduled data exports to Amazon Redshift for data warehousing and advanced analytics.

By integrating these features, CrumbCommander will further enhance its functionality and provide a more comprehensive management solution for retail stores.

References:

[1] AWS, AWS Documentation, *Amazon*. [Online]. Available: https://docs.aws.amazon.com/ [Accessed on: Aug 1, 2024].

[2] AWS Academy, *AWS Academy*, [Online]. Available: https://aws.amazon.com/training/awsacademy/ [Accessed on: Aug 1, 2024]

[3] Amazon Web Services, "Amazon EC2 Auto Scaling," *Amazon.* [Online]. Available: https://aws.amazon.com/ec2/autoscaling/. [Accessed on: Aug 1, 2024].

[4] Amazon Web Services, "Amazon RDS," *Amazon.* [Online]. Available: https://aws.amazon.com/rds/. [Accessed on: Aug 1, 2024].

[5] Amazon Web Services, "Amazon VPC," *Amazon.* [Online]. Available: https://aws.amazon.com/vpc/. [Accessed on: Aug 1, 2024].

[6] Amazon Web Services, "AWS Secrets Manager," *Amazon.* [Online]. Available: https://aws.amazon.com/secrets-manager/. [Accessed on: Aug 1, 2024].

[7] Amazon Web Services, "Amazon S3," *Amazon.* [Online]. Available: https://aws.amazon.com/s3/. [Accessed on: Aug 1, 2024].

[8] Amazon Web Services, "AWS CloudFormation," *Amazon.* [Online]. Available: https://aws.amazon.com/cloudformation/. [Accessed on: Aug 1, 2024].

[9] Amazon Web Services, "Amazon RDS Multi-AZ Deployments," *Amazon.* [Online]. Available: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.MultiAZ.html. [Accessed on: Aug 1, 2024].

[10] Amazon Web Services, "AWS Security Groups," *Amazon.* [Online]. Available: https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_SecurityGroups.html. [Accessed on: Aug 1, 2024].

[11] Amazon Web Services, "AWS Cost Management," *Amazon.* [Online]. Available: https://aws.amazon.com/aws-cost-management/. [Accessed on: Aug 1, 2024].