COSC349 Assignment 2 – Software in the Cloud

Callum Sullivan – 7234739, Anthony Dong – 2169260

Introduction

In many shared living situations, tenants struggle with the tedious and often complex task of manually tracking and dividing expenses each week. The Flat Finance Tracker is a web application that simplifies financial management for shared accommodations. By streamlining the financial management process, the Flat Finance Tracker promotes transparency among flatmates and creates a more organised financial environment for each tenant. This report will detail the development, implementation and cloud-based deployment of this application. Cloud technology offers key benefits including scalability, accessibility, cost-effectiveness and performance. These advantages enhance the applications' overall reliability and efficiency.

The application utilises a modern web development stack:

- Frontend: HTML, CSS, JavaScript, EJS
- Backend: Node.js & Express.js
- Infrastructure: Amazon EC2 virtual machines for both frontend and backend
- Database: Amazon Relational Database Service (RDS)
- Database Clean Up & Scheduling: Amazon Lambda and Amazon CloudWatch

This technology stack enables a responsive, user-friendly interface while ensuring secure data management and the ability to handle growing user demands. The combination of these technologies and cloud-based services results in a powerful, adaptable solution for managing shared living expenses.

Application Design and use of Virtual Machines

The Flat Finance Tracker leverages multiple AWS cloud services to create a scalable and maintainable architecture. At its core, the system consists of two main Amazon EC2 instances: a tenant interface and a manager interface. This design allows for independent scaling, testing and deployment. The data persistence is handled by an Amazon RDS MySQL database, ensuring reliable and efficient data storage. To maintain the system health and performance over time, the architecture also incorporates an AWS Lambda function for periodic database cleanup.

The TenantVM component is provisioned as an Amazon EC2 instance using Terraform. Its configuration uses Ubuntu AMI (ami-010e83f579f15bba0) and runs on a t2.micro instance type which balances cost-effectiveness with performance for the tenant interface. The TenantVM is associated with two security groups: "allow_web_and_ssh" and "allow_express_backend". These security groups permit inbound traffic on ports 22 (SSH), 80 (HTTP), 443 (HTTPS) and 3000 (tenant API). The deployment process is automated

through a user data script that orchestrates a series of essential steps. This script begins by updating the system and installing all necessary packages, including Node.js, npm and MySQL-client. It then proceeds to clone the application repository, ensuring the latest version of the code is deployed. Following this, the script sets up crucial environment variables for the database connection. During this process, the script utilises the RDS endpoint to configure seamless integration with the backend database. Once these steps are complete, the tenant application becomes available on port 3000. This architecture enables efficient and rapid deployment through its automated setup process, significantly reducing manual configuration errors. The result is a consistent and reliable deployment system that can easily adapt to the evolving needs of the Flat Finance Tracker application.

The ManagerVM shares many architectural similarities with the TenantVM, utilising the same EC2 instance type, AMI and security group configurations. However, several key differences distinguish its role within the Flat Finance Tracker system. The ManagerVM operates on port 3001, catering to administrative functions compared to TenantVMs use of port 3000. The MangerVM is also responsible for initialising the database schema as it waits for the RDS instance to be ready before executing the SQL script to setup the database. Additionally, both VMs set up similar environment variables for database connection although each is tailored to their respective applications. The differences highlight ManagerVM's role as both an administrative interface and a system initializer within the application architecture.

The Flat Finance Tracker utilises an Amazon RDS MySQL instance for data persistence. As defined in the Terraform configuration, this RDS instance runs MySQL 8.9 on a db.t3.micro instance class. The RDS instance is configured with the identifier "mydb", a schema name "myapp", and is set up with admin credentials, allowing secure connections from both tenant and manager interfaces through the associated security groups. Opting for Amazon RDS offers significant advantages over a manually configured MySQL instance on an EC2 server. It provides optimized database operations, ensuring better performance and easier scalability. As the application's demands grow, the database can be scaled both vertically and horizontally with minimal effort. RDS also provides high availability through its Multi-AZ (Availability Zone) deployment feature which maintains a synchronous standby replica in a different AZ for automatic failover. By using RDS, the architecture maintains consistency within the AWS ecosystem, simplifying the overall management and enhancing reliability for this financial application.

The second non-EC2 cloud service implemented in this application is an AWS Lambda function for automatic database maintenance which enhances the systems data management and long-term performance. This serverless component named "DbCleanupFunction", runs on Node.js 20.x runtime with 128mb of memory and a 60-second time. Implementing AWS Lambda brings several benefits, such as its cost-effectiveness due to its billing based solely on execution time. AWS Lambda also scales automatically to handle varying workloads, which significantly reduces operation overhead. The Lambda function is equipped with environment variables that secure store database connection details, this enables interaction with the RDS MySQL instance without hardcoding sensitive information. To ensure regular maintenance, a CloudWatch Event Rule triggers the function monthly. This regular checkup provides a consistent database cleanup

without continuous resource consumption. By leveraging this serverless architecture for database maintenance, the Flat Finance Tracker achieves efficient, cost-effective and hands-off management.

Cost Estimation

We estimate the cost to run this web application would be roughly $30 USD a month, or with light usage, roughly $ 35 USD. This mainly comes down to the cost of the two EC2 instances and the RDS database being kept running. An EC2 t2 micro costs $0.0166/hr * 24hrs * 30 days * 2 instances = $16.70 per month. The cost for multiple users on the instance doesn't change as you only pay to keep the instance running. The AWS RDS database costs $0.017/hr * 24hrs * 30 days = $12.24 per month, we then estimate with light data transfer use from users an extra $5 per month which is what gives the variation in price. The lambda use would be covered under the free tier as it is only called periodically one a month.

Configuration and Build Process

The application utilises Terraform and AWS. To run this application, follow this setup process:

1. **Install Terraform**
   Download and install Terraform from the official HashiCorp website

2. **Launch AWS Academy Learner Lab**
   Access you AWS academy account and start your Learner Lab session

3. **Git Clone the repository**
   git clone https://github.com/SullyJR/Bills-Tracker.git
   cd Bills-Tracker

4. **Configure AWS CLI**
   - Run “aws configure” in the terminal
   - Enter the Access Key ID & Secret Access Key found in AWS Details -> AWS CLI -> Show
   - Set the default region to us-east-1

5. **Update the AWS credentials file to match the AWS active session**
   - Run “mate ~/.aws/credentials”
   - Copy and paste the entire AWS CLI credentials file provided by AWS Academy into the textmate file

6. **Initialise Terraform in the terminal**
   “terraform init”

7. **Install Node.js dependencies**
   cd lambda
   npm install
   cd ../tenant
   npm install
   cd ../manager
   npm install
   cd ..

8. **Preview Terraform changes**
   “terraform plan”

9. **Apply Terraform configuration**
   “terraform apply” & type “yes”

After completing these steps, the applications infrastructure should be deployed on AWS and the application should be ready to run after a few minutes. Details of the EC2 and RDS instances are displayed on their corresponding AWS dashboard.

Development Process and Repository Structure

Our process for deploying the Flat Bills Tracker to the cloud was both careful and iterative. We started by getting a single virtual machine (VM) up and running in AWS using Terraform. This helped us get comfortable with infrastructure-as-code and AWS resource management. After the first VM was working, we expanded our Terraform setup to include a second VM, matching our two-service architecture.

The next challenge was connecting our MySQL database using an RDS instance, which turned out to be the toughest part. We faced several issues that required lots of debugging and reconfiguring. Being able to SSH into our VMs was extremely useful, allowing us to make real-time adjustments and troubleshoot directly.

Once the core infrastructure was stable, we focused on making the system easier to maintain. We added a Lambda function triggered by a CloudWatch event rule to clean and archive old bill entries periodically. This keeps the bill dashboard clean while preserving historical data and pre-emptively tackles potential scalability concerns.

Throughout the process, we made sure to keep the repository in a clean and formatted state. This not only helps with organisation and separation of concerns, but it also allows new developers to get up to speed developing in the project much faster.

Conclusion

The Flat Finance Tracker application demonstrates the effective use of cloud technologies to create a robust, scalable and efficient solution for managing shared living expenses. By leveraging AWS services such as EC2, RDS, Lambda and CloudWatch, the application achieves a balance between performance, cost-effectiveness and ease of maintenance. The architectural design utilises separate EC2 instances for tenant and manager interfaces, allowing for independent testing, scaling and deployment. This coupled with the use of RDS for the data persistence, provides a solid foundation for application. The implementation of

the Lambda function for automatic database maintenance further enhances the system's long-term performance. This application demonstrates how cloud technologies can be leveraged to create a scalable, maintainable and cost-effective web application. By using a range of different AWS services, we were able to cohesive yet effective system with a strong foundational base for future enhancements.