COSC349 Assignment 2: Deploying to AWS

James Robiony-Rogers (5793901) & Corban Surtees (4658948)

October 7, 2024

Contents

1 Introduction

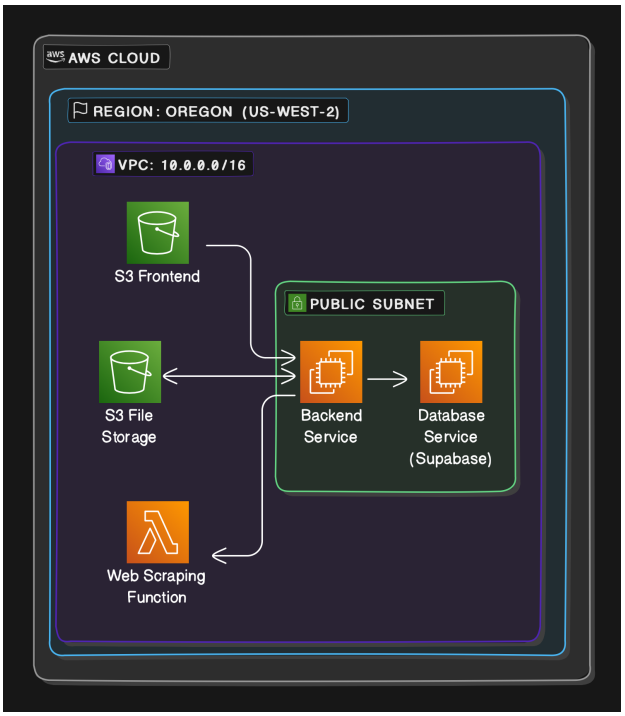
This report details the design, development and deployment of TailorWrite, a job application tracker, to the Amazon Web Services (AWS) cloud. TailorWrite was developed with modern cloud virtualisation practices and is designed to help students and job seekers manage their job search by recording key information about their applications, such as job roles, companies, statuses, and dates of submission. The project demonstrates the use of infrastructure as code (IaC) to automate the deployment of the application to AWS.

The application uses a range of AWS services to deploy the application to AWS. These including:

- Elastic Compute Cloud (EC2) for hosting the application.
- Simple Storage Service (S3) for storing static assets.
- AWS Lambda for a serverless function.

2 Cloud Architecture

The cloud architecture used to deploy TailorWrite involves several components deployed on AWS. The backend, hosted on an EC2 instance, manages the core business logic and interactions between the other services.



AWS Cloud Architecture Diagram

The cloud architecture dipected in the diagram to the right shows the deployment of TailorWrite to AWS. It is deployed in the Oregon (us-west-2) region and utilises multiple AWS services to host the application. The Virtual Private Cloud (VPC) is the core networking composnent of this architecture, providing an isolated environment where the services are hosted. The VPC uses the default CIDR block of `10.0.0.0/16` to allocate IP addresses. Supabase, also running on an EC2 instance, handles the application's PostgreSQL database, user authentication and provides an administration interface for managing the database. A web scraping function, deployed on AWS Lambda, retrieves job listing data, which is used to auto-populate the application tracking form for users. The frontend is hosted on Amazon S3, where the user interface is served to users, while file storage for job applications and related documents is also managed through S3, though a separate bucket.

Communication between components follows a clear structure. The frontend interacts with the backend via the public Elastic IP of the EC2 instance. The backend then handles interactions with the Supabase database using it's public Elastic IP. The python library `boto3` is used to interact with file storage service implemented ith S3 and the web scraping Lambda function.

EC2 was chosen for both the backend and Supabase to allow greater control over server configurations and resources in addition to maintaining our current implementation via Docker and Docker Compose. S3 was selected for it's cost effective file storage and ability to host static web files, while Lambda provided serverless compute power to execute the web scraping function on demand, avoiding the need for constantly allocated resources.

3 Deployment via Infrastructure as Code

The infrastructure of TailorWrite was automated using Terraform, an Infrastructure as Code (IaC) tool. Terraform scripts were used to provision the virtual networking components, security groups, Elastic IP addresses and upload the user interface's static build to S3, in addition to deploying the core services outlined above. Although most deployment steps are automated, the frontend build process remains manual prior to running the Terraform scripts. This is a limitation of the current implementation and could be improved in future iterations.

A detailed README file in the project repository offers step-by-step instructions for setting up the development environment and deploying the application. This documentation ensures that new developers can quickly onboard and contribute to the project.

4 User Interaction & Application Usage

Users can access TailorWrite via the frontend hosted in an S3 bucket. This provides a simple and intuitive interface for users to manage their job applications. After logging-in users can track their the status and details of their job applications and upload documents such as resumes, cover letters or on-boarding documents. The two minute screen recording demonstrates key functionalities such as navigating the job application tracker, using the web scraping feature to auto-populate forms, and uploading documents.

5 Estimated Running Costs

Using the AWS Pricing Calculator, we have estimated the monthly running costs of the TailorWrite application when the system is idle and when the system is in light use.

Estimated Costs When Idle

Service	AWS Service	Upfront Cost	Monthly Cost
Backend	EC2 (t2.micro)		$8.47
Supabase Database	EC2 (t3.small)		$15.18
Frontend	S3		$0.02
File Storage	S3	$0.26	$0.23
Web Scraping	Lambda	Free Tire Usage	$0.00
Total Monthly Cost			$23.90
Total Yearly Cost			$287.06

An instance of the AWS Cost Pricing calculator used to estimate the costs can be accessed here.

Estimated Costs When In Light Use

Service	AWS Service	Upfront Cost	Monthly Cost
Backend	EC2 (t2.micro)		$8.47
Supabase Database	EC2 (t3.small)		$15.18
Frontend	S3		$0.04
File Storage	S3	$0.26	$0.27
Web Scraping	Lambda	Free Tire Usage	$0.00

Total Monthly Cost	$23.96
Total Yearly Cost	$287.78

Instances of the AWS Cost Pricing calculator used to estimate the costs can when the system is idle can be accessed here, and when the system is in light use can be accessed here.

6 Development Process

Development of TailorWrite followed an incremental approach, with the commit history in the Git repository reflecting each stage of feature development and debugging. As this was a group project, we used a feature branch workflow to manage the development process. This structure ensured clarity and traceability throughout the development cycle.

Many challenges arose during development. Configuring security groups for the Ec2 instances proved difficult initially, as ingress and egress rules needed to be configured correctly to allow communication between EC2 instances, in addition to allowing SSH access for debugging. While provisioning an API Gateway to invoke the Lambda function was initially planned, issues with IAM permissions surfaced when attempting to invoke the Lambda function via API Gateway leading us to pivot away from API Gateway towards Lambda's function URL. This however, posed a new challenge as we were now facing CORS (Cross Origin Resource Sharing) issues when trying to invoke the Lambda function from the frontend. After hours of debugging, we decided to pivot for a third time to invoke the Lambda function directly via our backend (using the `boto3` library), which resolved the issue.

7 Project Links

Below are links to the project demo video and the GitHub repository:

- Repository: https://github.com/JamesRobionyRogers/Assignment02-CloudDeployment
- Project demo video: sdsd

8 Attribution

This project was developed by James Robiony-Rogers and Corban Surtees for COSC349 at the University of Otago. The project would not have been made possible without various open-source projects, libraries and websites. These include:

Development Tools

- Frontend: TailwindCSS components https://tailwindui.com/components
- Frontend: Material Tailwind https://material-tailwind.com/
- Frontend: Preline UI https://preline.co
- Backend: Flask https://flask.palletsprojects.com/en/2.0.x/
- Database: Supabase https://supabase.io/

Deployment Tools

- Terraform https://www.terraform.io/
- AWS https://aws.amazon.com/
- Docker https://www.docker.com/

- Docker Compose https://docs.docker.com/compose/
- Spacelift https://spacelift.io