System Architecture of My Flask Application

Disant Upadhyay

September 10, 2023

Contents

1 Introduction

This document furnishes a detailed technical outline of a Flask-based web application. It is structured to provide clarity on the server-side, client-side, and AWS infrastructure components. The overarching goal is to equip developers with the knowledge needed for deployment and regular maintenance.

2 Server-Side (Flask)

The server-side component is built with Flask, a Python micro web framework. Flask promotes agile development, making it apt for web applications.

2.1 Directory Structure

Description:

- `/myflaskapp/`: The main project directory.
- `/myflaskapp/myflaskapp/`: Flask's primary application package.
- *__init__.py*: Initializes Flask, registers blueprints, and incorporates other modules.
- `routes.py`: Houses route definitions, view functions, and core logic.
- `/static/`: Contains static files (e.g., CSS, JS, images) that Flask serves to clients.
- `/templates/`: Contains Jinja2 templates that Flask uses to dynamically produce HTML.
- `config.py`: Manages configurations, such as encrypted database credentials.
- `run.py`: The chief executable to launch the Flask app.
- `requirements.txt`: Enumerates the Python packages essential for the project.

```
/myflaskapp/
|-- /myflaskapp/
|   |-- __init__.py
|   |-- routes.py
|   |-- /static/
|   |-- /templates/
|-- config.py
|-- run.py
|-- requirements.txt
```

2.2 Database Interaction

The Flask application interfaces with a MySQL database on AWS RDS, crucial for persistent data storage and retrieval. For optimum security:

- Database credentials (username, password, and hostname) are encrypted via AWS Key Management Service (KMS).
- The Flask application decrypts these credentials at runtime through KMS.
- AWS security groups restrict direct access to the RDS instance, allowing only the EC2 instance running the Flask app.

3 Client-Side

The client-side offers users a dynamic web interface. It harnesses HTML for structure, CSS for design, and JavaScript for interactivity. The client-side fetches and displays data by communicating with the Flask backend.

3.1 Directory Structure

Description:

- `index.html`: Main entry point of the application.
- `listings.html, contact.html, resources.html, submit.html`: Various web pages delivering different functionalities.
- `styles.css`: Central stylesheet that dictates the application's aesthetics.
- `scripts.js`: Manages dynamic content, AJAX server calls, and other interactive functionalities.

```
/client/
|-- index.html
|-- listings.html
|-- contact.html
|-- resources.html
|-- submit.html
|-- styles.css
|-- scripts.js
|-- newfoundland_cover.jpg
```

4 AWS Infrastructure

The application leans on AWS services for aspects such as hosting, data storage, scalability, and security. The Flask app is hosted on an EC2 instance, the database runs on RDS, and media/static files are stored on S3. AWS IAM

ensures secure resource access, and CloudWatch facilitates extensive monitoring and logging.

4.1 System Architecture Diagram

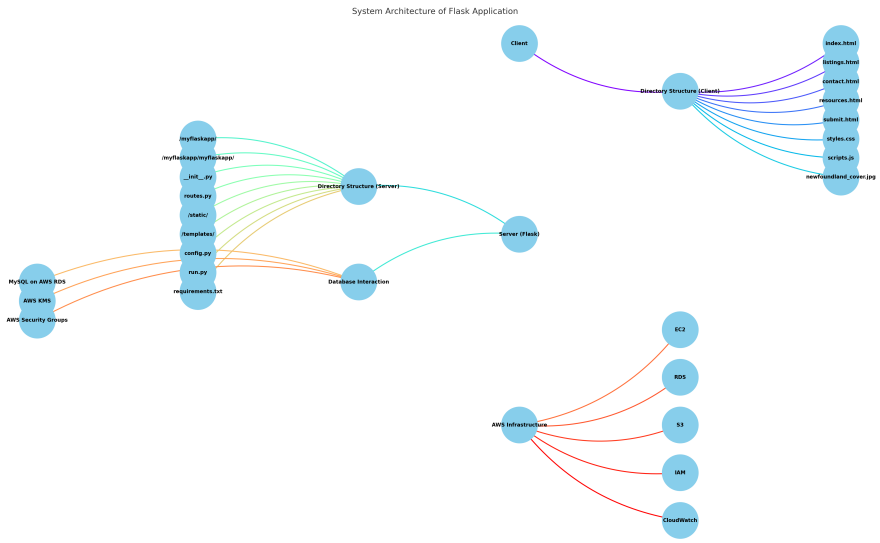

Figure 1: AWS System Architecture, highlighting the interactions between EC2, RDS, S3, and other AWS services. This schematic offers clarity on the flow and relations between different components, benefiting both developers and system architects.

5 Setup and Deployment

5.1 Local Deployment

For a local setup, follow these steps:

1. Clone the repository.
2. Ensure a modern web browser (e.g., Chrome, Firefox) is installed.
3. Navigate to the `client` directory.
4. Open `index.html` in your browser. This method loads the application locally, without the server-side functionality.

5.2 Server Deployment

For full server deployment:

1. Procure a domain and a server.
2. Install and set up a web server (like Nginx or Apache) to manage requests and direct them to the Flask application.
3. Transfer the `client` directory to the server's web root or integrate it within the Flask application's `templates` folder.
4. Configure the server to manage static files.
5. Open your domain in a web browser to view the application.

5.3 AWS Deployment

For AWS deployment:

1. Launch an EC2 instance for the Flask app, ensuring necessary ports (typically 80, 443, and 22 for SSH) are open.
2. Set up RDS for MySQL and ensure its security group settings permit the EC2 instance to establish a connection.
3. Use S3 for storing media and larger static files.
4. Employ KMS to encrypt and decrypt sensitive database credentials.
5. Use CloudWatch for comprehensive application monitoring and logging.
6. Implement IAM to control and manage AWS resource access meticulously.

6 Detailed Deployment Steps on EC2

6.1 Setting up the EC2 Instance

To set up the EC2 instance:

1. Log into the AWS Management Console.
2. Navigate to the EC2 dashboard.
3. Launch a new instance. For this application, a Linux-based instance, like Amazon Linux 2 or Ubuntu, is recommended.
4. While configuring the instance, make sure to attach an IAM role that has permissions to access RDS and S3.
5. Configure the security group to allow inbound traffic on port 80 (HTTP), 443 (HTTPS), and 22 (SSH).

6. Once the instance is running, connect to it using SSH. Use the provided command for reference:

```
chmod 400 path_to_key.pem
ssh -i "path_to_key.pem" ec2-user@your_ec2_ip
```

7. Once connected, update the system packages:

```
sudo apt update && sudo apt upgrade
```

6.2 Setting up the Flask Application on EC2

Upon SSH-ing into the EC2 instance:

1. Install the required system packages:

```
sudo apt install python3-pip python3-venv git
```

2. Clone the application repository:

```
git clone [repository_url]
```

3. Navigate to the project directory:

```
cd /path/to/repository
```

4. Follow the steps detailed in the "Setting up the Development Environment" section to prepare the application.

5. Use Gunicorn or a similar WSGI server to serve the Flask application and set it up as a system service to ensure it runs continuously.

6.3 Setting up Nginx as a Reverse Proxy on EC2

Nginx will handle incoming HTTP requests and forward them to the Flask application. This setup ensures better performance and the ability to handle multiple simultaneous connections.

1. Install Nginx:

```
sudo apt install nginx
```

2. Remove the default configuration and create a new one for the Flask application.
3. Ensure the new configuration forwards requests to the WSGI server (e.g., Gunicorn) running the Flask app.
4. Restart Nginx to apply the changes:

```
sudo systemctl restart nginx
```

6.4 Securing the Application with SSL on EC2

SSL encryption ensures data transmitted between the client and server is secure.

1. Install Certbot:

```
sudo apt install certbot python-certbot-nginx
```

2. Request and install an SSL certificate for your domain:

```
sudo certbot --nginx
```

3. Ensure the certificate auto-renews:

```
sudo certbot renew --dry-run
```

7 Ongoing Maintenance and Monitoring

7.1 Routine Maintenance

1. Frequently backup the RDS database.
2. Periodically update all software and packages to leverage security patches and improvements.
3. Monitor server resources and upscale the EC2 instance or RDS as required.

7.2 Monitoring with CloudWatch

1. Configure CloudWatch for the EC2 instance and RDS for detailed logging.
2. Monitor EC2 instance metrics to track CPU, memory usage, and network traffic.
3. Set CloudWatch Alarms for notifications on any anomalies or resource constraints.

8 Conclusion

This comprehensive guide equips developers with a step-by-step blueprint to deploy and maintain the Flask application. By adhering to these steps, developers can ensure a robust, scalable, and secure deployment. With continuous monitoring and regular maintenance, the application's stability and performance are guaranteed.