IST 615

Bryce Anthony

Shiva Kumar Godipally

Kumar Reddy

Final Project Paper - Deploying Artificial Intelligence Solutions

Due: Friday May 3rd

Submitted: Friday May 3rd

Document Length: 25

Project Background

The primary driving force behind this project was the desire to gain experience with deploying machine learning models. Our frustration stemmed from developing AI solutions that would only be available to a narrow audience of tech savvy individuals. This led us to explore the deployment of AI solutions using cloud infrastructure. In addition to increasing the accessibility of locally developed AI solutions, mastering the deployment process promises to enhance our ability to craft superior solutions and collaborate effectively within teams in our careers. By gaining insights into the decision-making aspects that impact the end-users, we can refine our code and optimize its utility for cloud infrastructure. Effective deployment not only ensures the scalability and adaptability of machine learning solutions but also enables them to deliver actionable insights across various sectors. Ultimately, deploying machine learning models is a critical step in driving tangible outcomes, boosting productivity, and instigating changes across industries via artificial intelligence solutions. In this project we deployed three Machine Learning models that we had previously developed. An exam score predictor using traditional ML to generate score predictions for students using a variety of demographic information, a deep learning model delivering AI powered predictions for regular season NBA matches, and an AI powered web application that brings the power of computer vision and large language models to calorie tracking and nutritional guidance.

Project #1 Exam Score Predictor

Project Description:

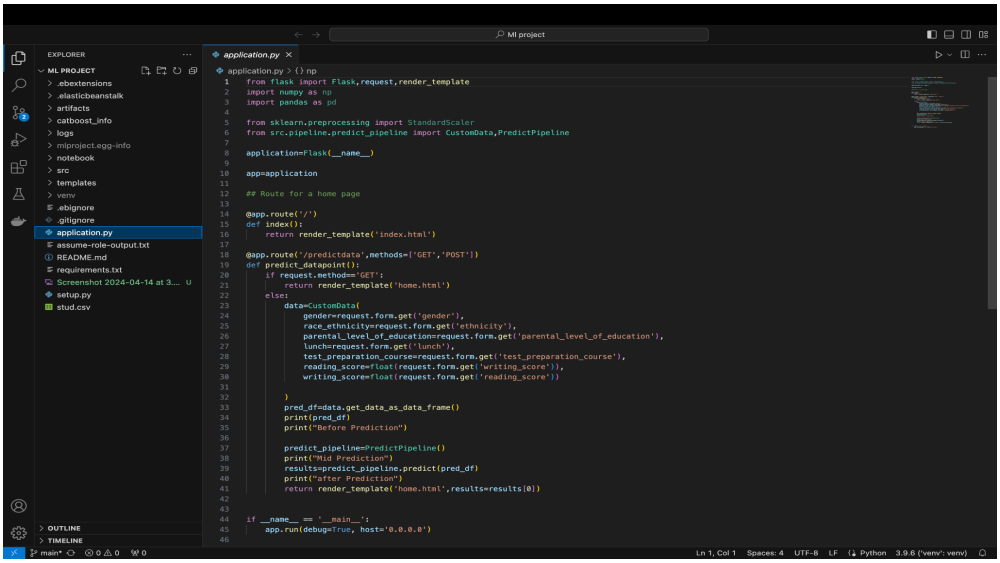
The Student Exam Performance Predictor is a machine learning-based web application designed to forecast student exam outcomes. Developed using Python and Flask, the project focuses on creating a robust and accurate predictive tool that considers multiple factors influencing student performance. This project showcases a commitment to transparent and collaborative development, being fully realized in VS Code and integrated with GitHub. The project serves as a valuable tool for educators and administrators, helping them identify areas for intervention and improvement. It also highlights the potential for technology to enhance educational outcomes through data-driven insights and predictive analytics.

Implementation Overview:

The implementation involved building a custom Python environment from scratch, providing a detailed understanding of the system configurations and dependencies involved in a machine learning workflow. Key components of the project include custom exception handlers and a detailed logging system, which enhance the application's robustness and debugging capabilities.

The web interface, developed using Flask, offers an intuitive user experience and integrates seamlessly with the machine learning model. The model has been trained on diverse datasets to ensure reliability and accuracy in predicting student exam performance. The project was successfully deployed on a host server, demonstrating its viability for real-world applications in educational settings.

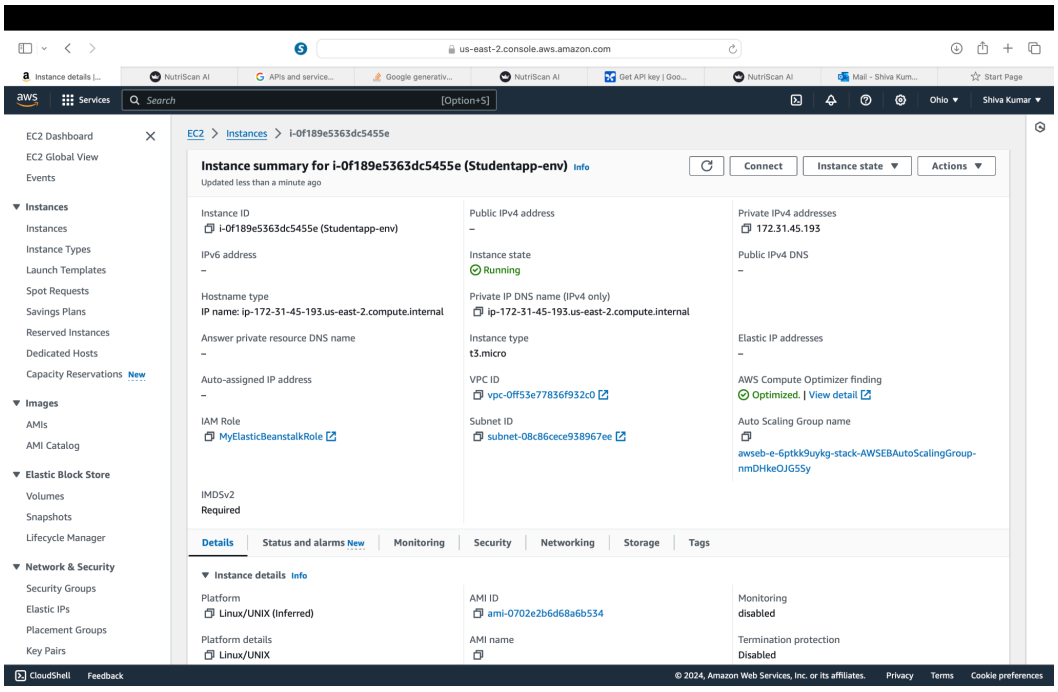
Local Development Environment



The application was developed using Python, with Flask as the web framework, to create an intuitive interface for predicting student exam performance. We implemented key features such as data handling, custom prediction pipelines, and template rendering within the "application.py" file. The local development environment provided a sandbox for us to experiment with different approaches.

By structuring the project effectively and focusing on the essential components during this phase, we laid a solid foundation for subsequent deployment and scaling, ensuring the application met both functional and user experience goals.

EC2 Instance Creation

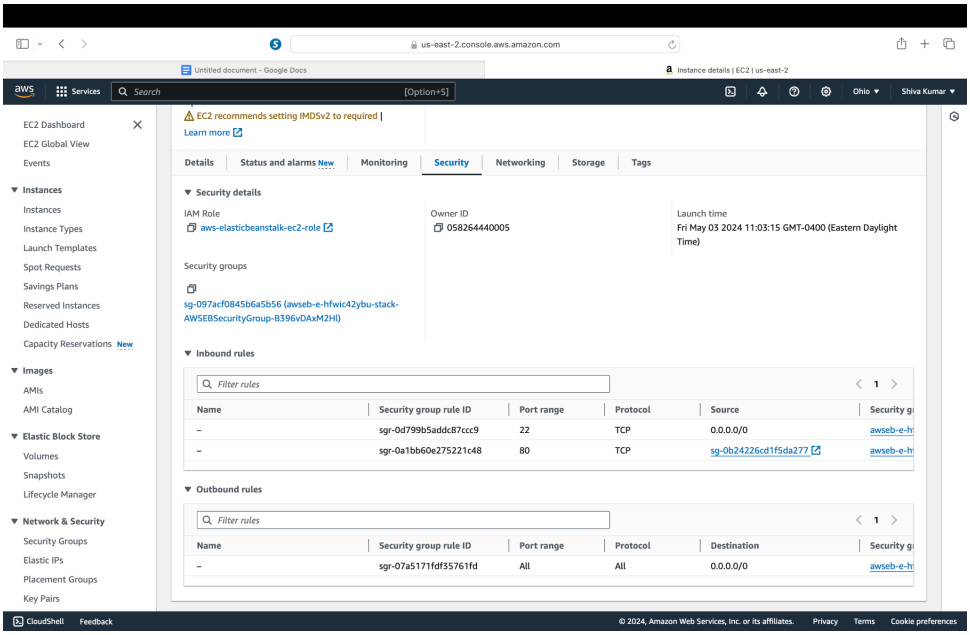


For the deployment of the Student Exam Performance Predictor, we created an EC2 instance within the AWS Management Console. This t3.micro instance provides an optimal balance of computing power and cost-effectiveness for our application. The instance is managed under a specific VPC and subnet, aligning with our network architecture. We used the "MyElasticBeanstalkRole" IAM role to handle permissions, ensuring that the instance could interact securely with other AWS services. The instance state is "running," indicating that our application is currently active and operational.

In this setup, we leveraged AWS Compute Optimizer to ensure that our instance is appropriately configured for performance and cost efficiency. The optimized status suggests that our EC2

environment is well-suited for the workload of the predictive application. This EC2 instance creation was a crucial step in transitioning from local development to a cloud-based deployment, ensuring scalability and reliability for the Student Exam Performance Predictor.

Security Rules Updated



The security rules were updated to allow ssh traffic via port 22 and inbound/outbound http traffic on all ports

The Student Exam Performance Indicator was Deployed

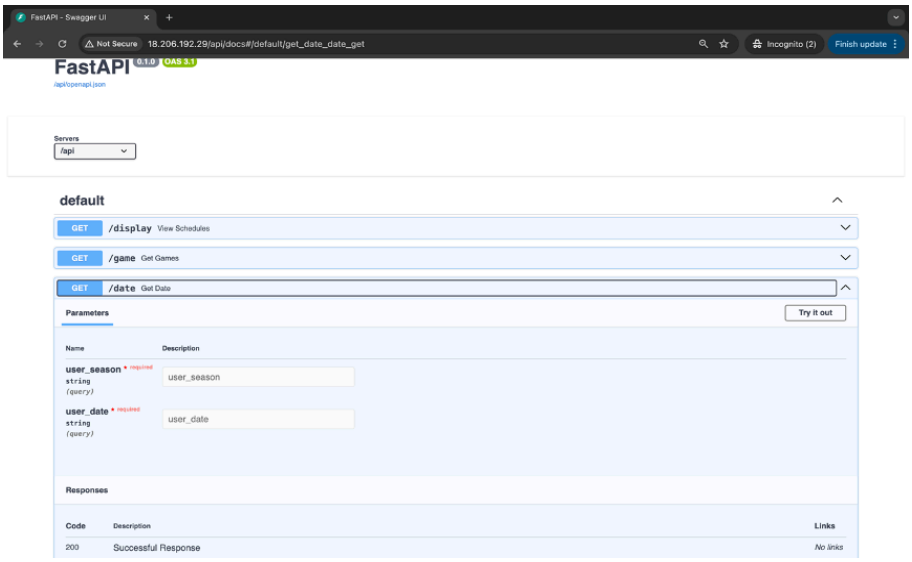
Discussion

AWS Elastic Beanstalk streamlined the deployment process, alleviating much of the heavy lifting. Using the Elastic beanstalk command line interface we were able to create our elastic beanstalk environment and a series of essential resources were initialized for us. This included; the instantiation of an EC2 instance, a load balancer for the EC2 instance, an auto scaling group, an S3 bucket, Amazon CloudWatch alarms, AWS CloudFormation,and security rules that allow the various resources to communicate with each other.

The security group allowed for the passage of HTTP traffic from the load balancer to the EC2 instance. The inclusion of a load balancer served as a critical component, adding an extra layer of security as we weren't directly exposing our instances to the internet. The load balancer security group was established to facilitate inbound traffic on port 80, enabling HTTP traffic from the internet to reach the load balancer securely. The auto-scaling group was integrated to dynamically manage the allocation of resources based on demand fluctuations. A domain name was configured to provide a user-friendly and accessible endpoint for our deployed application, culminating in a robust and scalable deployment architecture. The Amazon S3 bucket served as a central repository for our source code and project directory, facilitating seamless version control and deployment management. Additionally, Amazon CloudWatch alarms provided real-time monitoring capabilities, ensuring the health of our application and AWS CloudFormation orchestrated the connection of these services.

Ultimately this configuration facilitated by AWS Elastic Beanstalk not only streamlined our deployment process but also laid a solid foundation for a robust, scalable, and resilient deployment architecture, empowering us to focus on developing and enhancing our application with confidence.

Project #2 NBA Deep Learning match predictions API



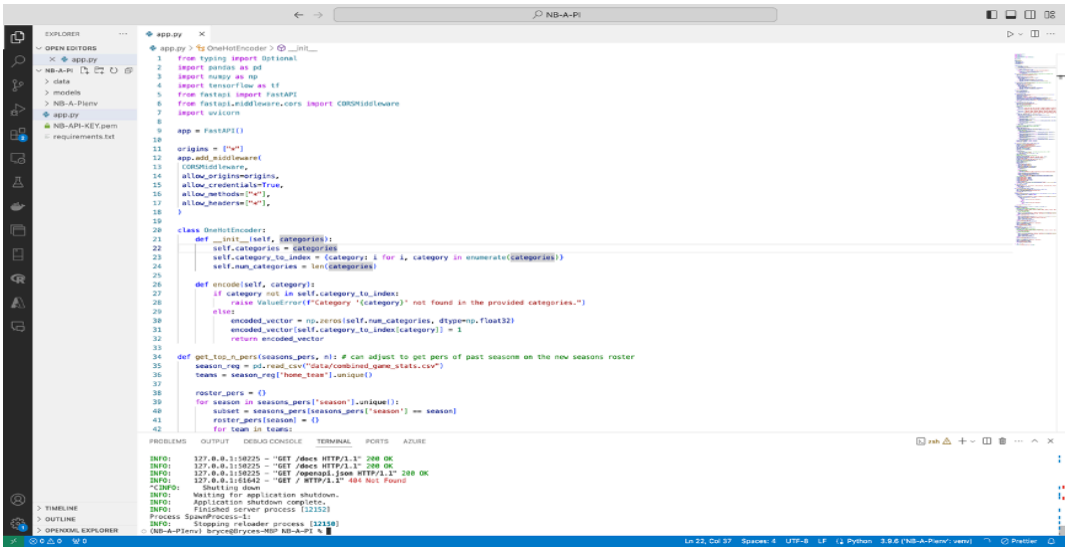
Project description

This Deep Learning model API was crafted with the aim of providing AI-driven insights into NBA game outcomes, accessible to all. Our conviction lies in empowering users with the predictive capabilities of machine learning, enriching the understanding of both dedicated fans and casual observers. With this tool at their disposal, enthusiasts, analysts, pundits, and casual viewers alike can gain deeper insights into individual games and their broader significance within the season. Equipped with such knowledge, they can make more informed predictions, whether for recreational enjoyment or strategic betting purposes. This model was developed locally for the Applied Machine Learning course and, like the other projects in this report, would benefit from being publicly accessible.

Implementation overview

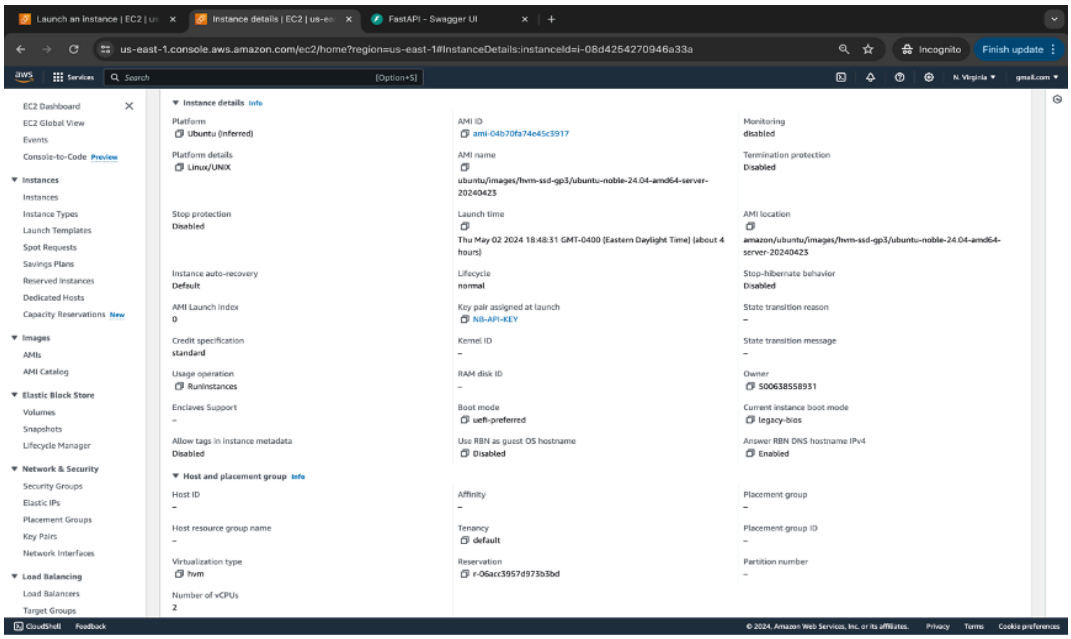
Because there are a variety of use cases for the information being output by the model, we decided that this model would be best deployed as an API. Originally we planned to deploy it via aws lambda since it is serverless, internally handles load balancing and scaling, and only requires users to pay for active computing time, but due to the size of the web Server we decided the best way to deploy the web server would be through the Amazon Elastic Compute Cloud although this meant we would be responsible for the load balancing and scaling features available when using aws lambda.

Local Development environment



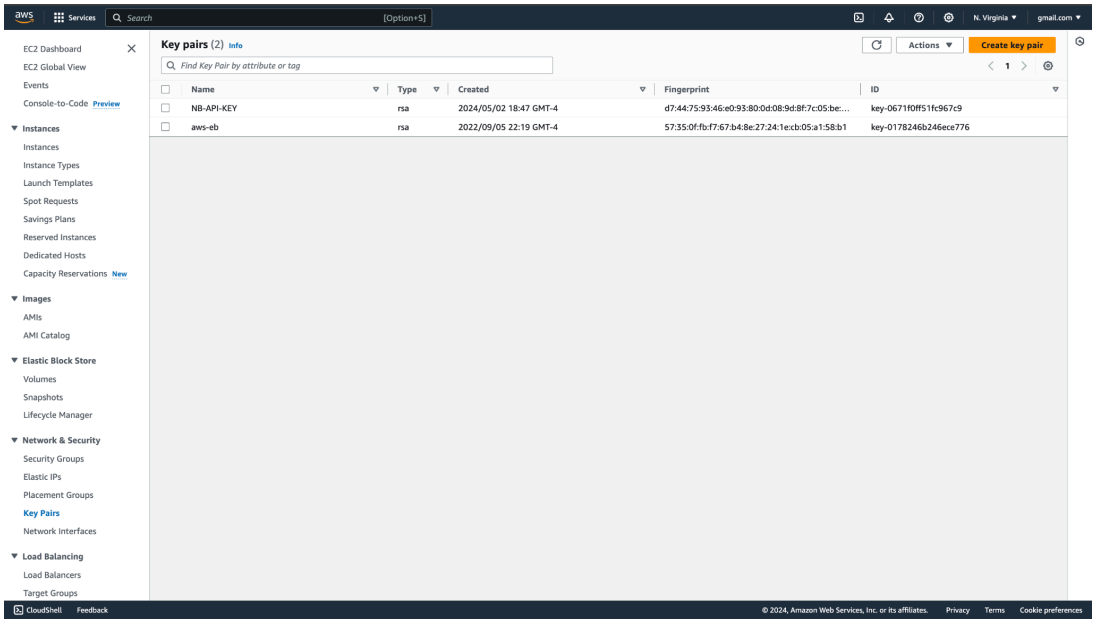
This deep learning model was developed locally and Fast api was used as as the interface for the model. The model was created and trained using TensorFlow running on Python and data scraped from the basketball reference website. Although the functionality of the API was handled in python, for it to become a functional api it was important for it to be hosted in the cloud and be able to concurrently handle multiple requests.

EC2 Instance Creation



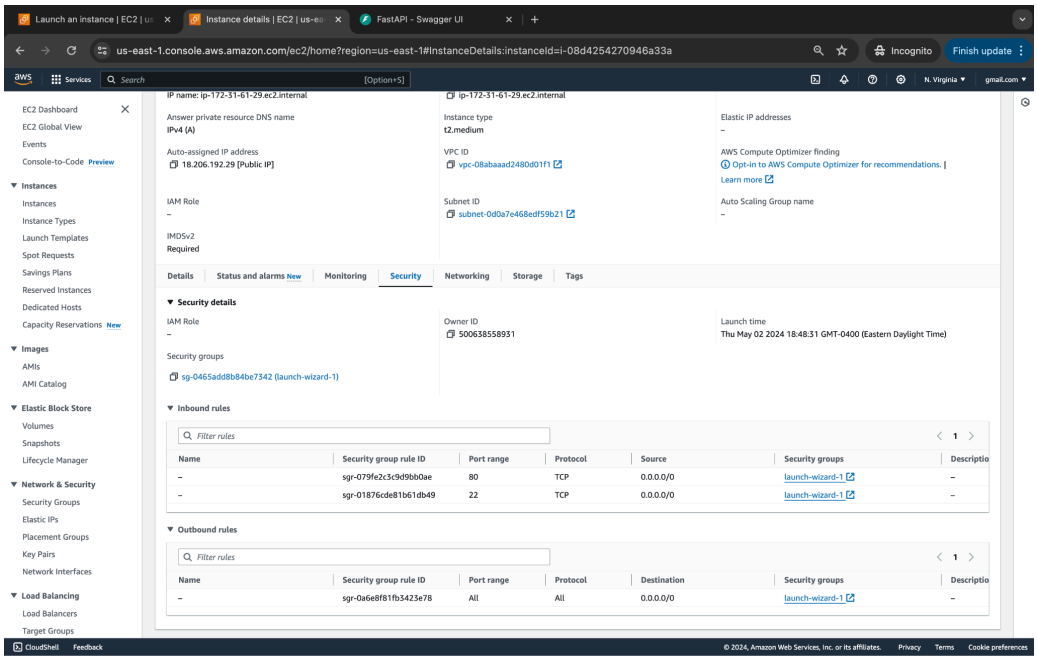
We selected an Ubuntu EC2 instance with 8 gb of storage for this project, this was necessary to host our web server as it required around 3 gb of memory in addition to the 3 gb of underlying software necessary to run the web server.

EC2 Key pair creation



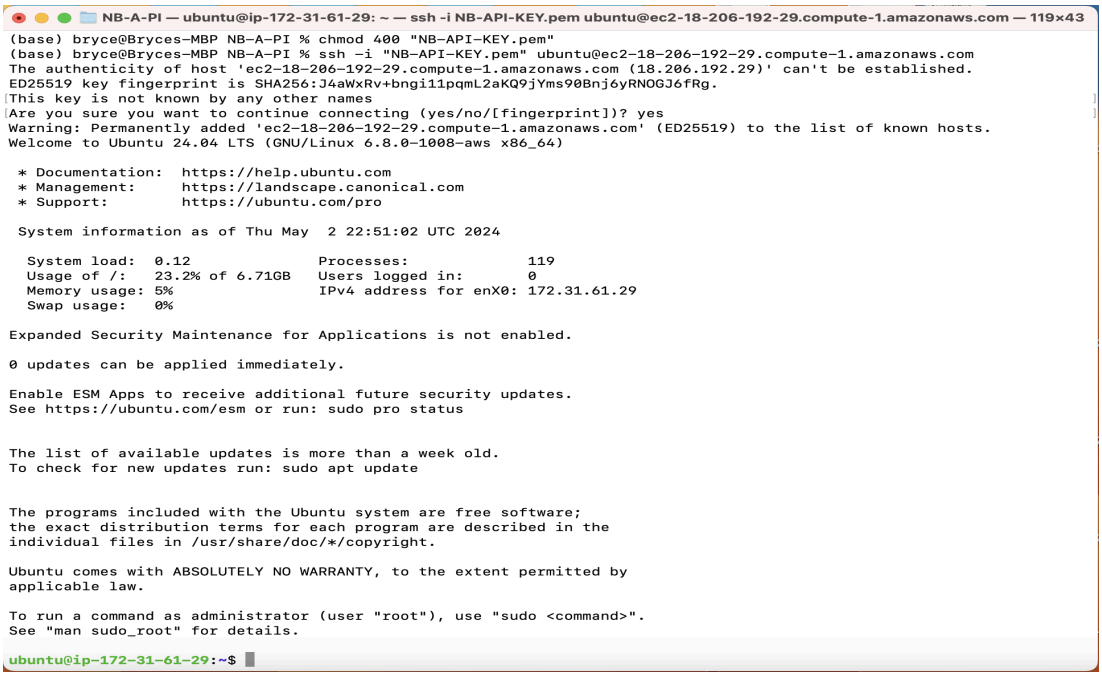
In order to authorize secure connections to the EC2 instance we needed a RSA Key pair and store it on our local machine. This key pair was then used to create secure and authorized connections to the EC2 instance.

Networking rules updated



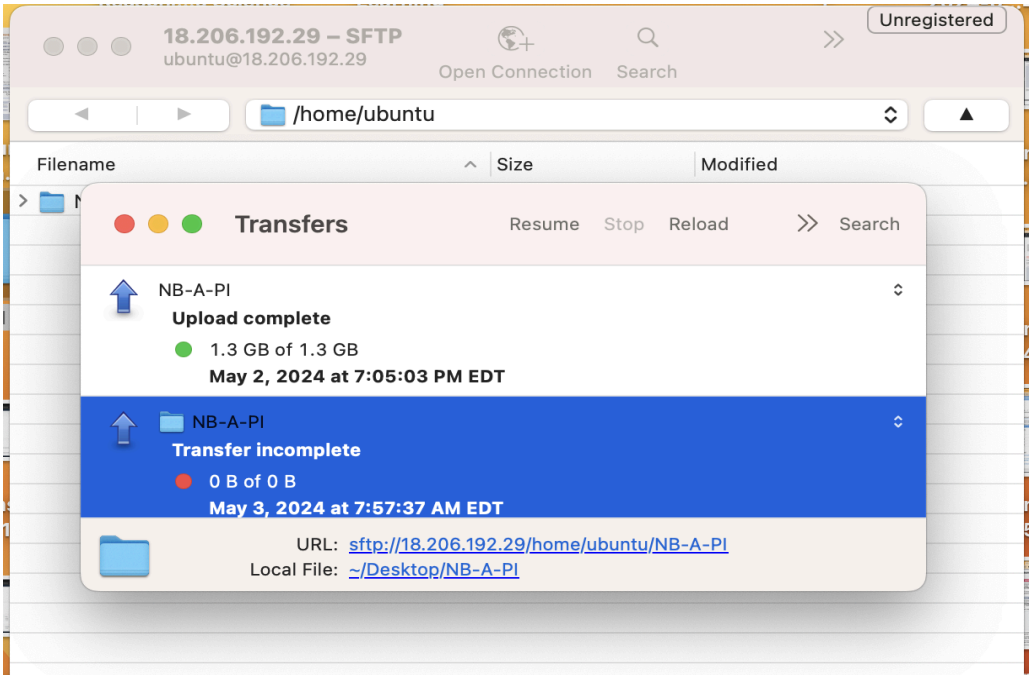
The Security rules of the EC2 instance were updated to allow in and outbound http connections from all ports as well as SSH connections in port 22.

SSH into EC2 Instance



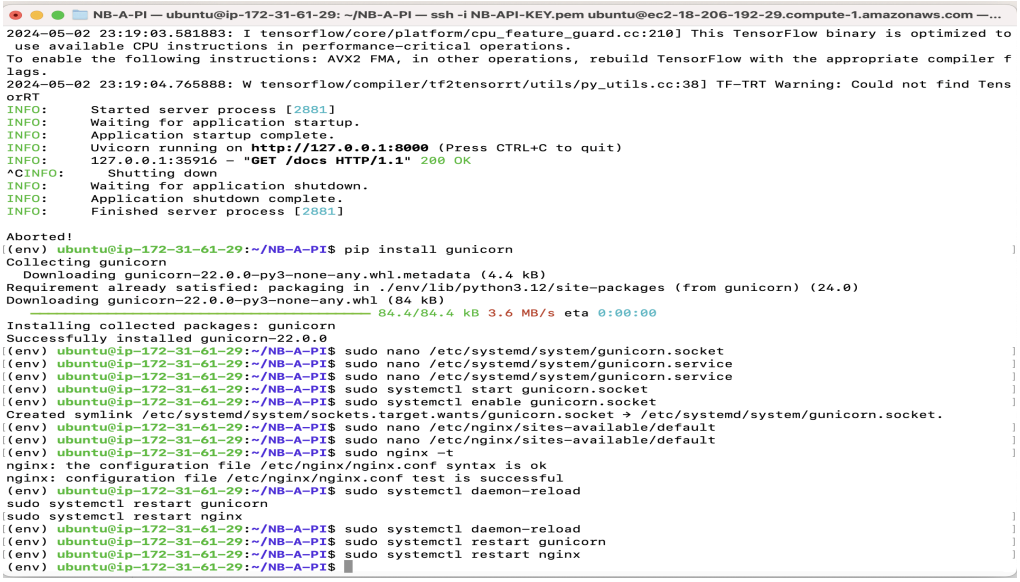
Using the rsa key pair we were able to confirm our credentials by connecting to EC2 instances via ssh.

Cyber Duck SFTP



We were then able to use the same rsa key pair with cyberduck's Secure Shell File Transfer Protocol to transfer the local directory of the API to the EC2 instance to deploy it.

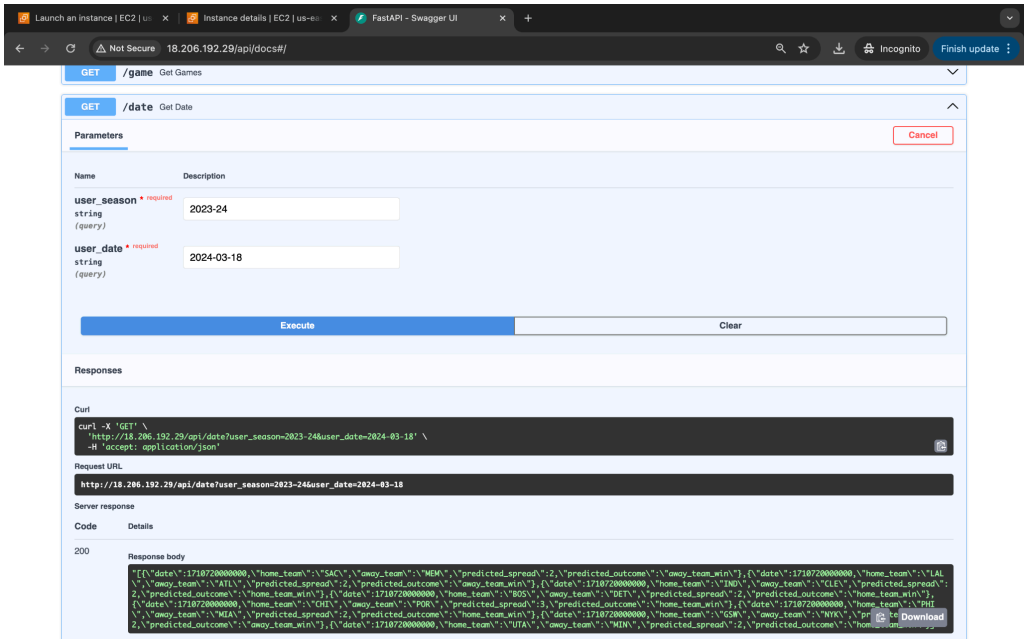
Set up EC2 Instance environment for web server



To make the EC2 instance into a functional API we needed to install NGINX and Uvicorn. NGINX is open source software for web serving, reverse proxying, caching, and load balancing. NGINX was used to handle connections to the server while Uvicorn serves as the application

server running the backend of the API. This included re-configuring our application to accept incoming requests using NGINX as a proxy.

Deployment

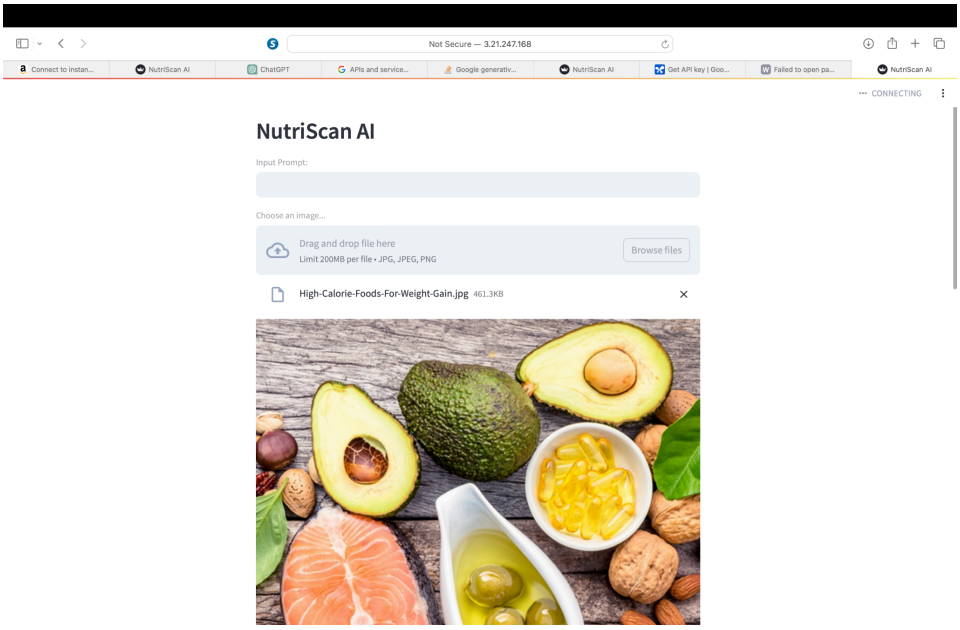


After successfully launching our web server it was officially deployed and we were able to access the API using the public ip address of our instance.

Discussion

Due to the size of the application, other more pre-configured aws services were unavailable to us. Although the EC2 instance isn't super costly, there is no downtime so the costs of the api will steadily increase. While this isn't ideal for an API not expecting lots of web traffic, If our api was expecting lots of web traffic or was to suddenly see a sharp increase in web traffic this could be advantageous because we internally handle load balancing and scaling using NGINX. By managing these aspects internally, we have greater control over resource allocation and can optimize costs based on the specific needs of our application. This setup ensures that we only incur costs proportional to the actual usage and requirements of our API, making it one of the most economical options on AWS considering the capabilities of our application.

Project #3 Nutriscan AI



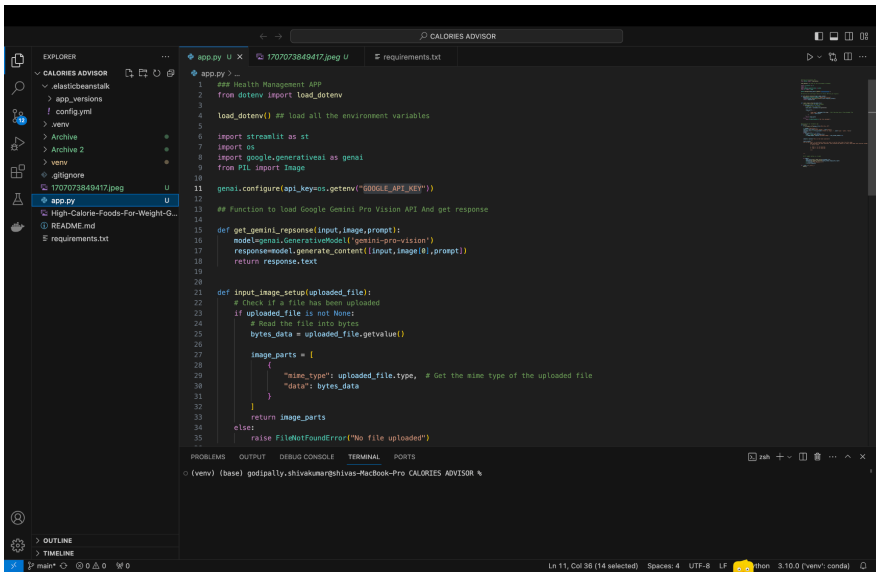
Project description

NutriScan AI is a cutting-edge calorie tracking application that uses advanced image recognition technology to analyze food and estimate calorie content. The app features a virtual dietary assistant powered by Google Gemini Pro, which provides personalized advice on meal optimization, catering to users' health goals. Designed for fitness enthusiasts, tech aficionados, and anyone interested in better managing their diet, NutriScan AI offers a user-friendly experience to analyze food images for calorie content and provide personalized dietary advice utilizing AI-powered insights and tailored guidance.

Implementation Overview

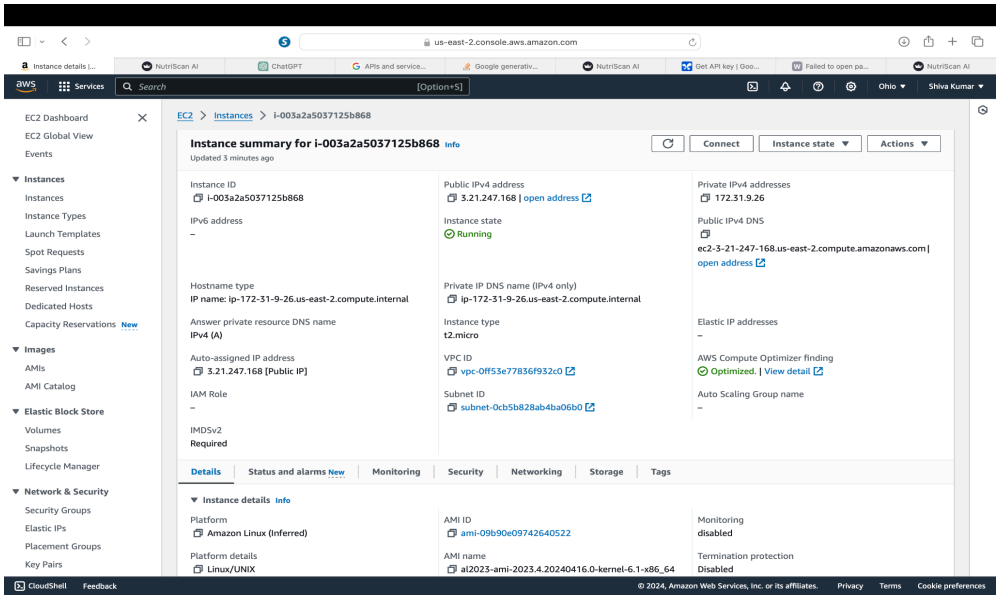
The project involved developing the app using Python, integrating it with Google Gemini Pro's API, and deploying it on an EC2 instance. Our efforts resulted in a holistic solution that bridges the gap between technology, nutrition, and health, enabling users to make informed dietary choices..

Local Development Environment



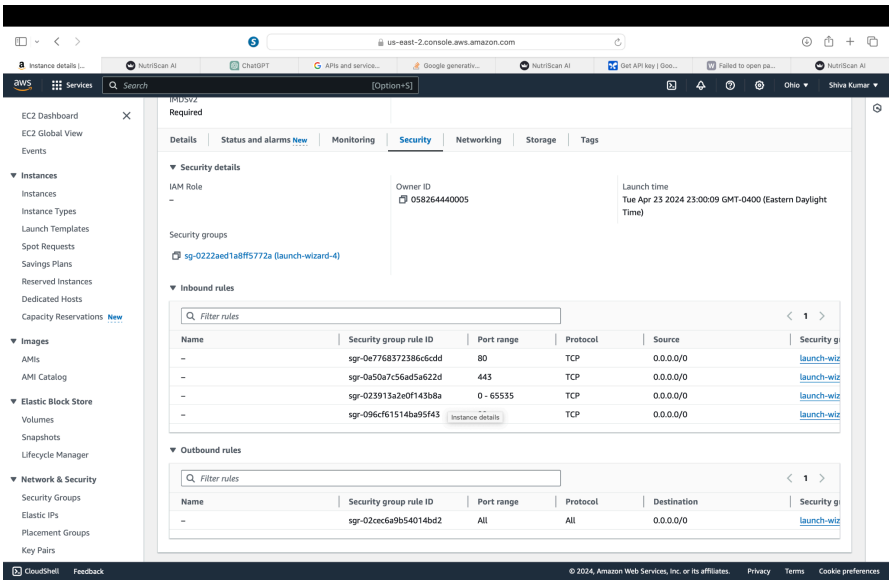
We used Python for development, leveraging libraries like Streamlit to build an intuitive user interface. The application incorporated image recognition and AI technologies. We integrated Google Gemini Pro's API to power the virtual dietary assistant.

EC2 Instance Creation



In setting up our cloud infrastructure, we created a t2.micro Amazon EC2 instance. This instance type was chosen for cost efficiency and functionality.

Security rules updated



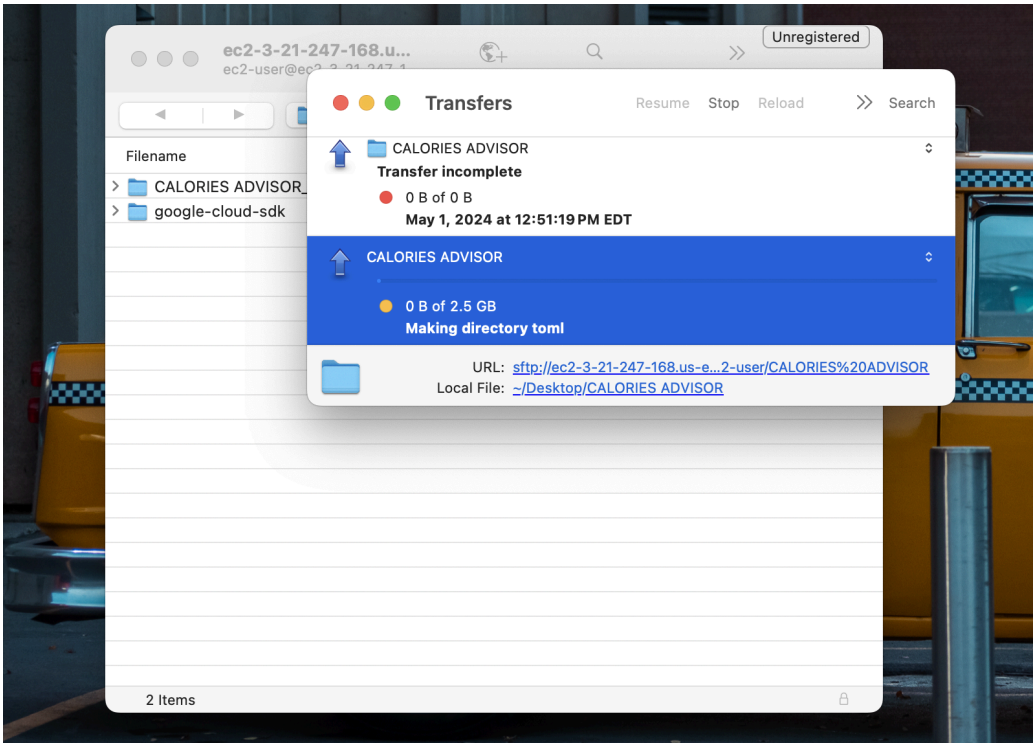
The security configuration for our EC2 instance involved updating both inbound and outbound rules to meet our specific needs. The instance was associated with a security group created during the setup wizard, and we defined multiple inbound rules to allow traffic over common web ports (80 and 443) as well as a broader range of TCP ports. This ensured our instance could function as a web server while also being flexible enough to accommodate other services. For outbound rules, we enabled unrestricted access, ensuring the instance could communicate with any necessary external services. These configurations provided the right balance between functionality and security, supporting our application's needs while protecting it from unauthorized access.

SSH into EC2 Instance

To further secure and manage our EC2 instance, we utilized SSH to establish a secure shell connection. This approach allowed us to remotely access the instance's command line,

facilitating direct control over configurations, software installations, and system maintenance. Using SSH, we authenticated with a private key, ensuring secure access to the instance. This setup provided us with the flexibility to manage the instance efficiently while maintaining strong security practices, essential for protecting sensitive data and ensuring the smooth operation of our cloud-based applications

CyberDuck SFTP



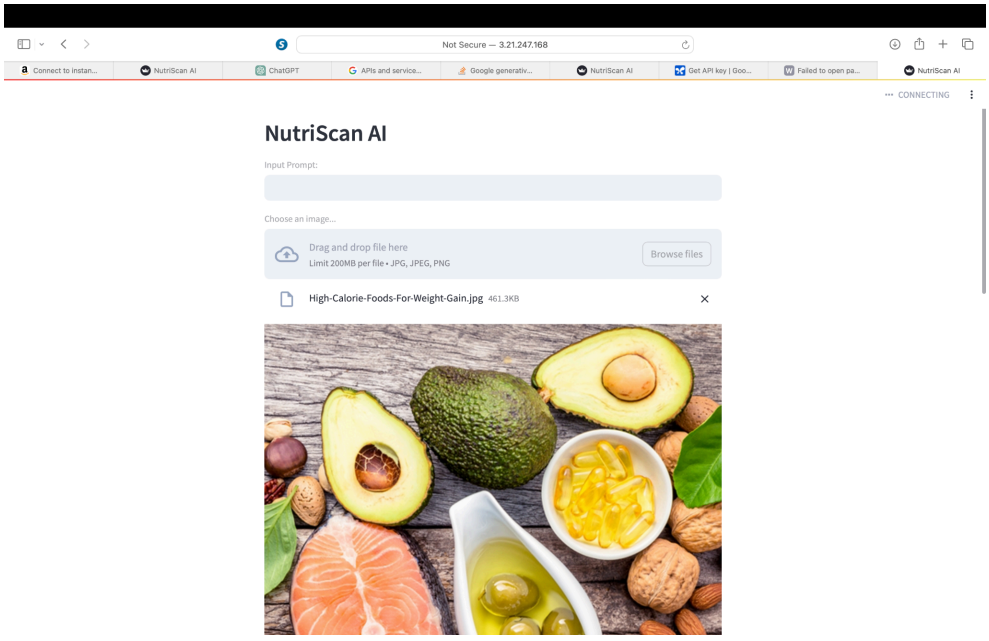
In addition to using SSH, we leveraged Cyberduck for secure file transfer using SFTP. This allowed us to efficiently manage and transfer files between our local environment and the EC2 instance. With Cyberduck, we benefited from an intuitive graphical interface, simplifying the process of uploading and downloading files. Using this method, we securely managed project files, application assets, and logs, enhancing our workflow while maintaining robust security measures. This approach was vital for ensuring seamless deployment and management of our cloud-based applications.

Set up EC2 Instance environment for web server

```
Requirement already satisfied: pyasn1==0.5.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 37)) (0.5.1)
Requirement already satisfied: pyasn1-modules==0.3.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 38)) (0.3.0)
Requirement already satisfied: pydeck==0.8.1b0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 39)) (0.8.1b0)
Requirement already satisfied: Pygments==2.17.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 40)) (2.17.2)
Requirement already satisfied: python-dateutil==2.8.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 41)) (2.8.2)
Requirement already satisfied: python-dotenv==1.0.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 42)) (1.0.1)
Requirement already satisfied: pytz==2024.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 43)) (2024.1)
Requirement already satisfied: referencing==0.33.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 44)) (0.33.0)
Requirement already satisfied: requests==2.31.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 45)) (2.31.0)
Requirement already satisfied: rich==13.7.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 46)) (13.7.0)
Requirement already satisfied: rpds-py==0.18.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 47)) (0.18.0)
Requirement already satisfied: rsa==4.9 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 48)) (4.9)
Requirement already satisfied: scikit-learn==1.4.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 49)) (1.4.2)
Requirement already satisfied: scipy==1.13.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 50)) (1.13.0)
Requirement already satisfied: six==1.16.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 51)) (1.16.0)
Requirement already satisfied: smmap==5.0.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 52)) (5.0.1)
Requirement already satisfied: streamlit==1.31.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 53)) (1.31.1)
Requirement already satisfied: tenacity==8.2.3 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 54)) (8.2.3)
Requirement already satisfied: threadpoolctl==3.4.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 55)) (3.4.0)
Requirement already satisfied: toml==0.10.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 56)) (0.10.2)
Requirement already satisfied: toolz==0.12.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 57)) (0.12.1)
Requirement already satisfied: tornado==6.4 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 58)) (6.4)
Requirement already satisfied: tqdm==4.66.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 59)) (4.66.2)
Requirement already satisfied: typing_extensions==4.9.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 60)) (4.9.0)
Requirement already satisfied: tzdata==2024.1 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 61)) (2024.1)
Requirement already satisfied: tzlocal==5.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 62)) (5.2)
Requirement already satisfied: urllib3==2.2.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 63)) (2.2.0)
Requirement already satisfied: validators==0.22.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 64)) (0.22.0)
Requirement already satisfied: Werkzeug==3.0.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 65)) (3.0.2)
Requirement already satisfied: youtube-transcript-api==0.6.2 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 66)) (0.6.2)
Requirement already satisfied: zipp==3.17.0 in /home/ec2-user/.local/lib/python3.9/site-packages (from -r requirements.txt (line 67)) (3.17.0)
Requirement already satisfied: watchdog>=2.1.5 in /home/ec2-user/.local/lib/python3.9/site-packages (from streamlit==1.31.1->-r requirements.txt (line 53)) (4.0.0)
[ec2-user@ip-172-31-9-26 CALORIES ADVISOR]$ streamlit run app.py

Collecting usage statistics. To deactivate, set browser.gatherUsageStats to False.


  You can now view your Streamlit app in your browser.

  Network URL: http://172.31.9.26:8501
  External URL: http://3.21.247.168:8501

2024-05-01 16:47:59.885 Uncaught app exception
```

To set up the EC2 instance as a web server, we first ensured that the necessary software dependencies used in the development environment were installed. The screenshot shows a successful installation of Python packages from a requirements file, which was crucial for our application. Using the terminal, we installed and verified the presence of key packages such as streamlit, which serves as the backbone for our web application. Next, we ran the web server, using Streamlit to host our application. The output provided network URLs, including both internal and external addresses, allowing us to access the web application securely.

Deployment



The deployment of NutriScan AI on our EC2 instance involved configuring a secure environment for web hosting and integrating our machine learning model. The screenshot shows the user interface, where users can upload images of their meals to analyze calorie content.

Discussion

The development and deployment of NutriScan AI on an EC2 instance underscored the importance of integrating advanced technology with user-centric design in achieving impactful solutions. Throughout this project, we navigated through several critical phases, from setting up a secure cloud environment to integrating machine learning for dietary analysis and establishing a user-friendly interface using Streamlit.

Deploying this application as an EC2 instance allows for full control over server environments for customization and flexibility in software and configuration choices. Scalability is readily achievable, enabling seamless handling of fluctuating traffic demands while maintaining performance. Additionally, the pay-as-you-go pricing model enhances cost-effectiveness, particularly for applications with variable usage patterns. Managing EC2 instances entails technical expertise and ongoing maintenance tasks, potentially increasing operational overhead. Scalability configuration complexities may arise, and improper scaling can affect performance and cost-efficiency. Moreover, reliance on a single EC2 instance introduces a potential single point of failure, while ensuring security requires meticulous attention to access controls and firewall configurations to mitigate risks of unauthorized access and breaches

Conclusion

This project emphasized the value of leveraging cloud technologies and machine learning to address a variety of real-world challenges. By focusing on both the technical and user experience aspects this project underscored the significance of utilizing cloud technologies and machine learning to tackle real-world challenges effectively. By addressing both technical intricacies and user experience considerations, we developed a comprehensive solution that not only delivers AI-powered solutions to consumers and diverse applications but also enhances our teamwork capabilities. Our focus on the deployment aspect of cloud technologies and machine learning provided invaluable insights into efficient deployment processes, allowing us to understand best practices and their impact on project success. This experience not only expanded our knowledge of cloud infrastructure but also equipped us with practical skills essential for deploying scalable and impactful AI-powered applications collaboratively within teams.