Distribution Plan for Flask Application

Introduction

Our Flask application allows users to interact with a user system made with MongoDB and extract and visualize financial data from public company 10-Q reports using regex and the yfinance, pymupdf, and matplotlib libraries.

Choosing a Containerization Tool

Docker: Docker is the selected containerization tool due to its widespread adoption, comprehensive documentation, and compatibility with major cloud providers like Azure and AWS. Docker containers encapsulate the application and its environment, ensuring consistency across different deployment environments. Anecdotally,

Why Docker?

Portability: Docker containers can run consistently across any machine that has Docker installed, mitigating the "it works on my machine" syndrome.
Scalability and Isolation: Each part of the application can be contained in separate containers and scaled independently.
Community and Ecosystem: Docker has a large community and a plethora of available images from Docker Hub, which can significantly speed up the development and deployment processes.

Containerization Process

Dockerfile Creation: The first step is creating a Dockerfile, which is a blueprint for building a Docker image. This file includes:

- Base image (e.g., python:3.8-slim)
- Dependencies installation (Flask, yfinance, regex, etc.)
- Copying the application source code to the container
- Exposing the application port
- Command to run the application

Building the Image: Using the command docker build, the Docker image is created based on the instructions in the Dockerfile.
Running the Container: Once the image is built, the container can be started using docker run, specifying any required environment variables and ports.

Deployment Plan

AWS Elastic Beanstalk: For deployment, we would choose AWS Elastic Beanstalk due to its ease of use for deploying web applications and automatic scaling and management of the underlying infrastructure.
We also looked into Azure, and while we've heard it has a more seamless user experience AWS seems to be more of a "set and forget" platform that we won't have to worry about later on.

Deployment Steps:

Preparation:

- Test the Docker image in our computers first.
- Prepare a Dockerrun.aws.json file.

Environment Setup:

- Set up an Elastic Beanstalk application and environment through the AWS Management Console.
- Choose the Docker platform as the environment type.

Deployment:

We'd deploy the application by uploading the Docker image.
Configure environment variables such as database URLs or API keys in the Elastic Beanstalk environment management section.

Monitoring and Scaling:

Utilize Elastic Beanstalk's monitoring tools to track application health and performance.
Adjust auto-scaling settings based on demand predictions and actual usage.

Containerizing and deploying the Flask application using Docker and AWS will streamline development and simplify deployment. Our approach also leverages the robust infrastructure of AWS, providing reliability benefits (and making it easier for us to maintain it).

References

Docker Documentation: https://docs.docker.com/
AWS Elastic Beanstalk Documentation: https://docs.aws.amazon.com/elasticbeanstalk/