

Productionization of a Machine Learning Model

Thesis Submitted in Partial Fulfillment
of the Requirements
of the Jay and Jeanie Schottenstein Honors Program

Yeshiva College
Yeshiva University
May 2023

Mark Adelman

Mentor: Professor Dave Feltenberger, Computer Science

Introduction

The ultimate goal of almost any technological project is to deliver it to the end-users efficiently and effectively. This process, known as productionization, is a critical step in the lifecycle of technological development. It marks the transition from a conceptual model or prototype to a usable product. It's not just about creating a working prototype but also ensuring that it's scalable, reliable, and user-friendly. Productionization is the phase where an idea becomes a tangible product, ready to create real-world impacts.

The Importance of Productionization

In the realm of technology, creating a groundbreaking application or developing a new machine-learning model is only half the battle. The other half involves productionizing the technology - ensuring it's accessible, scalable, and useful to its intended audience. A piece of software residing on a developer's local machine or a model sitting in some file system does not contribute value until it is accessible, being used and interacted with.

Without effective productionization, even the most promising technology is rendered moot. It enables innovation to reach the public, making a broader impact and often leading to further innovation.

Productionization Methods

Overview

Local

Local productionization refers to deploying the technology on a local system or machine. Local deployment is often the most straightforward and might be sufficient for the

initial testing of a product (especially when rapid iteration is taking place), but it may lack availability (e.g. what happens when the local Wi-Fi goes down?) and scalability (e.g. can it behave efficiently even when many users are trying to use the application?).

Cloud Providers

Cloud-based productionization, on the other hand, leverages the power of cloud computing to host, distribute, and scale the technology. Cloud providers typically offer robust, scalable infrastructure that can support large-scale technology deployments with ease.

There are several leading cloud providers available, each offering various cloud services that can be used to productionize an application. Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure are among the most popular. These platforms offer services for computing power, storage, database management, and other essential functions needed for productionization.

Our Implementation

In approaching our project, we were deeply committed to treating it as a real-world scenario. Hence, we felt it appropriate to set up a proper deployment of the model using a cloud provider which can reliably deliver our product to our end users.

While different cloud providers might have their strengths and weaknesses, they, for the most part, tend to offer similar types of services and the deployment of an application using one provider can oftentimes be deployed in a parallel fashion using another. Hence, we decided to use the cloud provider we were most accustomed to, which for us was Amazon Web Services.

Application Design

Overview

For most user-facing applications, there exists two disjoint yet interconnected components: the front-end and the back-end.

The front-end is the visible part of an application that users interact with. It constitutes everything the user experiences directly - from the layout of the website or app to how they interact with it, including buttons, images, forms, and other visual elements. Front-end development involves designing and implementing the user interface (UI) and user experience (UX). This is typically achieved using languages such as HTML, CSS, and JavaScript, as well as frameworks like React or Angular. The goal of front-end development is to create an intuitive and easy-to-use interface that effectively translates the user's actions into requests that the back-end can process.

The back-end is where the main logic of the application resides. It's the engine that drives the application and remains hidden from the user. Back-end development involves handling the 'behind-the-scenes' functionality of web applications. This includes data management, data pipelines, and all other functionality necessary for the application to function properly but that doesn't directly concern the user interacting with the application.

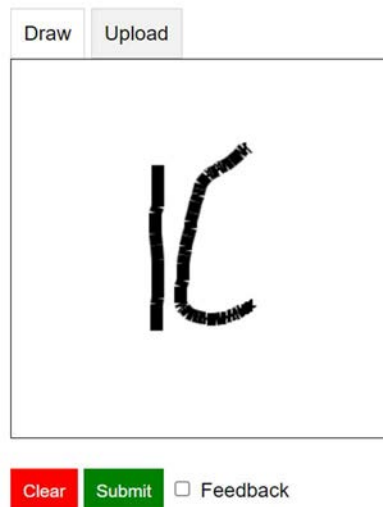
When designing web applications, it's usually best practice to keep the front-end and back-end components separate from one another as much as possible, and to establish a clear distinction between the two components. This enhances the maintainability, scalability, and flexibility of the application. By decoupling the front-end and back-end, each can be developed, tested, updated, and scaled independently of the other.

Front-end Implementation

For our application, we implemented the front-end using AWS EC2, Flask, HTML, and JavaScript. AWS EC2, or Amazon Web Services Elastic Compute Cloud, is a service that provides scalable computing capacity in the AWS cloud. This enables scaling of an application's compute power as demand for the application grows. Flask was used to handle the web server portion of your application. Flask is a web framework written in Python that allows for easily deploying a website. The actual interface that users see and interact with was built using HTML and JavaScript. HTML is the standard language for documents designed to be displayed in a web browser. JavaScript, specifically, is used to make web pages interactive so that they can respond to user actions.

On the webpage, users can upload an image of a character, or draw a character on a canvas and then submit the image/drawing to be classified. Below is a screenshot from the webpage:

Hebrew Script Letter Classifier



Draw Upload

Clear Submit Feedback

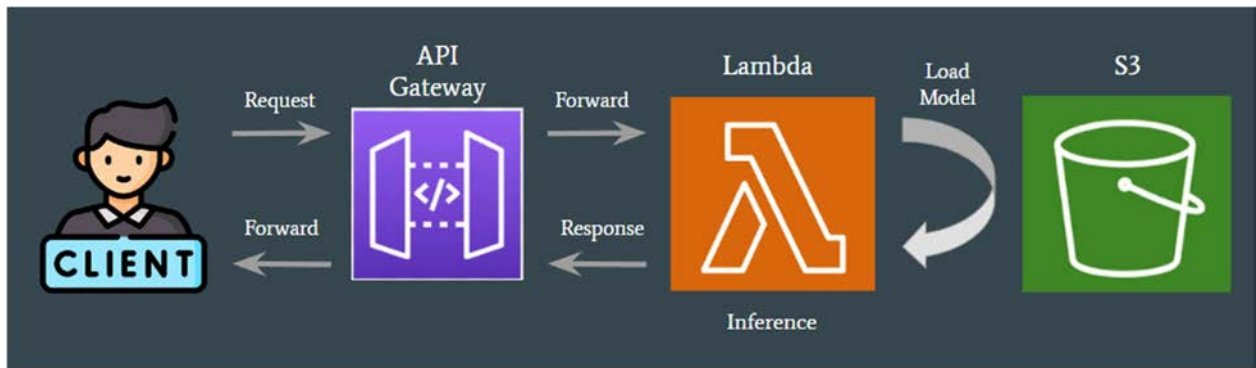
Back-end Implementation

AWS API Gateway, Lambda, and S3

For our first attempt at implementing the back-end for our application, we attempted to use Amazon Gateway, AWS Lambda, and Amazon S3 to create a “serverless architecture.”

Amazon Gateway is a service that receives incoming requests and guides them to where they are supposed to go. AWS Lambda is a service that can run code without managing the servers on which the code is to be run. Amazon S3 is a storage system that houses and organizes persistent data. In an ideal scenario, these three services can be utilized to create a "serverless architecture," which is essentially a setup that allows us to run our application without the need for us to maintain physical servers.

Our vision was to have the Amazon API Gateway manage all the requests coming from a client (usually sent from the front-end), triggering specific AWS Lambda functions that would handle these requests and issue a response (i.e. letter classification), and use Amazon S3 for storing the model that Lambda would use to make the classification. From a productization standpoint this approach would have offered us many benefits, like the high scalability of a serverless architecture, efficient cost structure where we only pay for the computing time we actually consume, and reduced operational overhead since we wouldn't have to manage any servers. Below is a depiction of such an architecture:



However, our attempt at implementing this approach did not work due to Lambda's lack of support for 3rd party libraries that were necessary for the model we were using.

AWS Sagemaker

Our next approach was to use AWS SageMaker to deploy our machine learning model and provide access via an API. SageMaker is a cloud-based service that aids in the building, training, and deployment of machine learning models.

However, we encountered complexities and difficulties in the process of setting up and managing SageMaker. These challenges were compounded by our plans for continuous iterations and improvements to our model. With each modification or upgrade to the model, we would have been required to repeatedly set up and configure the SageMaker environment. As a result, we decided to pivot away from SageMaker and instead sought a more manageable solution.

AWS EC2

Ultimately, we decided to implement the back-end on an EC2 instance. We used Flask to build an API which gets invoked by the front-end when a user submits an image to be classified. The same EC2 server was also used to host the front-end of our application. By

keeping both the front-end and back-end on the same server, we were able to minimize latency, as both the client-facing portion of our application and the server-side logic were housed in the same place. This setup eliminated the need for data to travel across different networks, making our application faster and more responsive (as well as more cost efficient).

However, keeping in mind scalability and best practices for application design, we also ensured that the back-end and front-end components were isolated from one another such that they can function independently if ever needed. This decoupling allows us to have the flexibility to move either component to a different server in case the demand on our single EC2 instance becomes too high. This way, we can scale our application effectively, ensuring that it continues to perform well even as user demand increases.

Model Improvement

It is important to understand that productionization is not the end but rather the beginning of an ongoing process of improvement and optimization. Once a product is live, it can be used to gather valuable data. This data can be collected and analyzed to understand the product's effectiveness and identify any shortcomings or areas for improvement. This iterative process can inform decisions about updates and enhancements to a project.

For our application, we decided to leverage the interface as a way to gather more data for the model. When using the website users can toggle “feedback” mode. When toggled, users will be prompted to verify whether the classification the model makes is correct or not and can supply the correct classification when the model does indeed make a misclassification. This data is then saved and can be used to further train and improve the model.