Feasibility Study

Team Our Daily Bread

Jacob Johnson
Emily Linderman
Tom Lobbestael
Michael Dykema

Supervisor: Joe Fahnestock

Table of Contents

Project Overview

Our Daily Bread (ODB) has requested a recommendation engine built on AWS that will populate content cards to be displayed on web pages to help drive user engagement and generate more awareness of the different content and offerings on their site. Our team's task will be to use ODB's existing data in order to build a machine learning algorithm to generate recommendations based on user data. Those recommendations will be retrieved by an API built by the team, however integration with ODB's website will be taken care of by ODB's web development team. Finally, we will be in charge of setting up monitoring of our API, the pipeline we create in AWS, as well as cost warnings for the various AWS services we will be using.

Technologies and Architecture

Languages

The majority of this project will be developing infrastructure as code (IaC), which will be written in Python. Additionally, Node.js is required as it is used as the backend for AWS services.

Technologies

AWS Command Line Interface

This is a tool that allows interaction with AWS services in any command-line shell. Useful for direct management and testing of services, if necessary this will also allow for development of bash scripts for the various services we will be using.

AWS Cloud Development Kit

The AWS Cloud Development Kit (CDK) is a framework for defining cloud infrastructure in code and provisioning it through AWS CloudFormation. The AWS CDK supports multiple languages, we will be mainly using Python because that is the language that most of us are familiar with.

AWS S3

Amazon Simple Storage Service will be hosting our data lake. As the name implies, it is a service for warehousing/storing data. Costs are relatively low at $0.023 GB/month.

AWS Glue

AWS Glue is a serverless data integration service that makes it easy to discover, prepare, and combine data for analytics, machine learning, and application development. AWS Glue provides both visual and code-based interfaces to make data integration easier. Using this utility will help us monitor our build as it continues.

AWS Lake Formation

AWS Lake Formation is a fully managed service for creating and managing a data lake from existing data. You point Lake Formation to an existing database and give it access credentials and it crawls and reads your data sources to extract technical metadata (like schema definitions). It also creates a searchable catalog to describe this information for users, so they can discover available data sets. It also can perform transformations on data, as well as help clean and prepare your data for analysis.

AWS Lambda

AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers. You can trigger Lambda from over 200 AWS services and software as a service (SaaS) applications, and only pay for what you use. Using this service will aid in the management of the many other AWS services being used.

AWS EC2

AWS Elastic Compute Cloud is a computation platform for different services that perform different calculations such as machine learning, analytics, and more. This will be the service utilized to run SageMaker and build our recommendation engine. Rates for this service are calculated per hour and by amount of data transferred out. Hourly rates vary by instance type used for calculations.

AWS SageMaker

Amazon SageMaker is a fully managed machine learning service. It provides common machine learning algorithms that are optimized to run efficiently against extremely large data in a distributed environment. It also features “SageMaker Studio”, an integrated

machine learning environment where you can build, train, deploy, and analyze your models all in the same application.

AWS CloudWatch

Amazon CloudWatch is a monitoring and observability service built to provide you with data and actionable insights to monitor your applications, respond to system-wide performance changes, and optimize resource utilization. Using this service will allow an easy way to create a dashboard to monitor the health of the pipeline.

DataDog

ODB currently uses DataDog to monitor/test their current website and APIs so will be expanding their testing to include the API that we build for recommendations. This monitoring includes checking for bad requests and uptime.

ASP.Net / AWS API Gateway

We will either be building a standalone API using ASP.Net/C# or utilizing AWS API Gateway to host a HTTP or REST API. From the research we have done, we are leaning towards using API Gateway as the costs are extremely minimal ($1.00-$3.50/million requests), doesn't require separate hosting for ODB, and keeps everything in the same family.

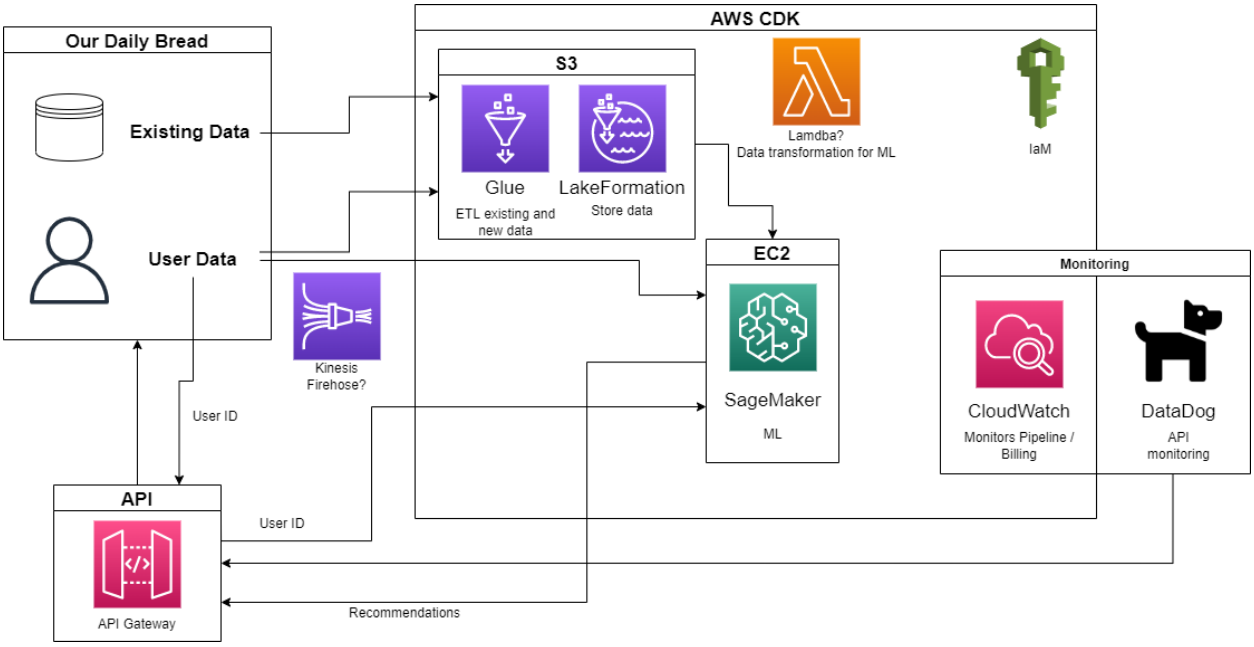
AWS Kinesis Firehose

Kinesis Firehose is an ETL service used to deliver real-time data to data lakes and analytical services. Our team will likely want to import real-time data to our pipeline and the two options are either Glue or Kinesis Firehose. Research has indicated that Firehose may be the preferred solution, but either will work and both will have to be tested further down the line.

AWS Identity and Access Management

This service is responsible for controlling access to the different services and data we will be utilizing. Different users and accounts can be limited to which services they have access to, and can get as fine grained as specific tables and columns in our data lake. It is very important to our clients that security is taken seriously and we limit access to bare minimum for different users. IAM comes free with an AWS account.

Architecture



The above figure is a diagram of what our proposed solution will look like. It describes how all the different technologies and services will work together and gives a rough idea of what the data pipeline will look like. Data will be extracted, transformed, and loaded by Glue in our data lake stored on S3. That data will potentially be transformed by processes on Lambda before being loaded into SageMaker where our recommendation engine will provide results for our API.

Requirements

After our initial research and meetings with Joe, our understanding and anticipation of this project is that it will be more collaborative and linear than most projects as there are certain steps that must be completed before starting on other portions of the project. With this in mind, our team plans on most people working on most areas, but each area will have a lead who is in charge of coordinating and will be the “expert” of that area of the project.

Data ETL and Storage

As stated in the proposal, ODB already has sources of user data and needs them aggregated into one data lake. Jacob will be taking the lead on this part of the project, and to accomplish this we will be using LakeFormation as it is the simplest and most efficient way to set up a data lake from preexisting resources. All we will need is a way to point In using LakeFormation that means that our data lake will be hosted using the Amazon Simple Storage Service, which provides cost efficient storage. LakeFormation uses AWS Glue to provide import workflows from any data source or schema, as well as machine learning transforms.

Machine Learning

Tom will be responsible for taking the lead on the machine learning aspects of the project. AWS SageMaker will be utilized to train the recommendation engine which will allow for extensive training and testing. With SageMaker being an AWS service it will also be much easier to integrate with other technologies that will be used along the data pipeline. We will be implementing an item-item algorithm that recommends content based on what other content the user has viewed.

Accessing Recommendations (API)

Michael will be taking the lead in developing the API as he has a significant amount of experience working with .Net APIs. The original proposal for this project included a standalone API that would be hosted by ODB and likely use the .Net framework. After meeting with Joe however, ODB has no preference on the technology the team decides to utilize. While doing research on the services AWS has to offer, our team stumbled across AWS API Gateway, a service for creating APIs for AWS services. With API Gateway, one can create either a HTTP or a REST API and at this point we are not sure which one will be more appropriate. It appears that a HTTP API would be preferable as it is lighter-weight and more cost effective so it is the route we will plan on using for now. Should our needs require a REST API, making that change shouldn’t be too difficult. By using API Gateway we eliminate the need for ODB to dedicate their own resources to hosting it, and keeps our different technologies closer together.

Monitoring

The creation of monitoring software will be Emily's responsibility. There will be two tools used for monitoring the pipeline, AWS CloudWatch and DataDog. CloudWatch will allow ODB to set alarms that will go off if the pipeline suddenly receives too much traffic which could lead to a high AWS bill. DataDog on the other hand will be used to monitor more day to day pipeline health.

Security

Security and access management will be a top priority for all over the course of this project, but Michael will be taking the lead on security. This will mean the he is responsible for ensuring everyone pays attention to the roles and access of their accounts, that no passwords or account information is stored on Git in any way, and that sensitive or identifying information is not stored, or if it shall become necessary, that it is kept as secure and private as possible.

Sprints and Deliverables

Sprint 1

- Create data lake
- Populate data lake with existing user data
- Real-time data stream into data lake
- Configure accounts and security access
- Initial research and setup of monitoring dashboard

Sprint 2

- Research implementation of chosen recommendation algorithm
- Train and test recommendation engine
- Create API

Sprint 3

- Prepare for integration with ODB website
- Create alarms to monitor pipeline health
- Set up monitoring software for API

Sprint 4

- Fine tuning of recommendation engine
- Bug fixes (If necessary)

Sprint 5

- Fine tuning of recommendation engine
- Documentation
- Polishing

Sprint 6

- Final Report for ODB on cost and expected value generated
- Submission of project
- Presentation

Policies and Standards

Code Repository/Branch Organization

Our team will utilize Git Flow as our standard for branching and managing our repository. With this model, we will utilize two permanent branches, *master* and *develop*. *Master* will contain production ready code and will be the most stable version of our project. *Develop* will contain the most recent features and improvements that are awaiting release. Along with this we will have three types of temporary branches. *Feature* branches will be used to work on features that are actively being developed for future updates. After completion these will be merged back into the *develop* branch signifying readiness of that feature for release. We will also have *release* branches that are for last minute fine tuning to the current *develop* branch before merging to *master*. This type of branch will include things such as metadata for the version that is about to be merged with *master* as well as small bug fixes. The final type of branch that is to be implemented is the *hotfix* branch. These will be used to fix bugs or other issues that are found within the master branch . After the problem has been fixed these branches will be merged into both the *develop* and *master* branches.

Documentation and Styling

All Python code shall follow the conventions listed in the Style Guide for Python. Comments should be present on all methods detailing the functionality along with any returns and arguments. Since the goal of this project is to have our infrastructure continue to be used and managed by ODB after the semester ends, liberal

documentation is needed and required. This includes but is not limited to documentation for security and account permissions, how to read and use the dashboard, and details of recommendation engine training.

SCRUM Methodologies

We will be using ZenHub to manage our sprints and keep track of progress. Our project will be broken down into 6 sprints, each lasting 2 weeks. Progress will be tracked over the lifetime of the project and during each sprint using burndown charts. At the end of each sprint we will either perform a written report or presentation on the progress made during each sprint. Additionally, at the end of each sprint code must be complete and functional and a final sprint commit will be made tagged “sprint{n}” with the number of the sprint. Our sprint boards will use the standard categories for ZenHub.

Integrations

We are anticipating the MVP of this project to be ready for integration with the production ODB site within about 6 weeks. Initial talks with Joe have indicated that with a little bit of pre-planning, once our side is ready it can be integrated within a week or so. This will involve our API being tested, reviewed and content cards being designed. With our final 3 sprints we hope to keep improving our machine learning algorithm and providing better recommendations and generating more traffic. Our plan is to integrate these improvements at the end of each sprint.

Testing

In addition to testing our code such as our API, testing will involve comparing our recommendations against each other in order to compare results and improve our algorithm(s). Most likely this will involve doing a form of A/B testing, where users will randomly be assigned different algorithms and results can be compared that way. Measurables have not yet been determined, but some discussion has been had and has brought forth number of clicks, length of stay on the website, and so on.