

Cloud-Based Recommendation System for E-commerce

Group 3:

Michelle Dong

Vidhi Mansharamani

Ziyu Ou

Siyuan Zhu

APAN 5450: Cloud Computing

Professor Philip Shilane

April 28, 2024

- **Business Case**

Executive Summary

The modern e-commerce landscape presents both an opportunity and a challenge, necessitating sophisticated solutions to navigate its breadth. We propose a cloud-based Recommendation System utilizing an AWS technology stack to enhance customer experience and offer a market advantage through personalized shopping journeys.

Problem Statement

In the extensive e-commerce marketplace, customers often feel overwhelmed and seek shopping experiences tailored to their preferences. Current technological setups fail to effectively leverage user data due to scalability challenges, data complexity, and processing power constraints, preventing businesses from delivering sought-after personalized shopping experiences. Due to limitations in their current technological infrastructure, companies need help to utilize the vast amounts of data necessary to provide these customized insights.

Proposed Solution

Our solution is a Recommendation System that exploits the scalability and efficiency of cloud computing and machine learning. It will integrate Amazon's suite of services, including S3 for data storage, DynamoDB for high-performance database management, EC2 for machine learning computations, API Gateway and AWS Lambda for web interfaces, and VPC for fortified networking and secure content delivery. The system will prioritize data privacy and implement strict security protocols to maintain customer trust. Advanced algorithms will be at the system's core, intelligently generating precise recommendations that mirror each user's shopping patterns and preferences.

Objectives

The system aims to analyze user data with high precision, leveraging innovative machine-learning techniques for personalization. Our goal is to refine the shopping experience, making it more intuitive and engaging, and to reduce the time customers spend searching for the right products.

Expected Impact

By offering a tailored selection of products, we anticipate a rise in customer satisfaction and a measurable increase in sales conversions. We aim to see customer satisfaction scores improve by at least 20% in the first year of implementation. Additionally, by accurately meeting customer needs, we expect to cultivate a loyal customer base, thereby enhancing the lifetime value of each customer.

Conclusion

The proposed cloud-based Recommendation System is designed to be an essential method for enriching the customer journey, creating a competitive edge, and driving business growth in the e-commerce sector. This innovative solution will streamline product discovery for consumers and equip businesses with deep insights, enabling them to optimize their offerings and operational efficiencies for maximum return.

• Data Description and Procurement

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14318520 entries, 0 to 14318519
Data columns (total 10 columns):
#   Column              Dtype
---  -
0   rating              float64
1   title               object
2   text                object
3   images              object
4   asin                object
5   parent_asin         object
6   user_id             object
7   timestamp           int64
8   helpful_vote        int64
9   verified_purchase   bool
dtypes: bool(1), float64(1), int64(2), object(6)
memory usage: 996.8+ MB
```

The primary data supporting our analysis and subsequent recommendations are sourced from the user-generated content within Amazon.com's '*Grocery_and_Gourmet_Food*' category (see *Appendix A*). This dataset, substantial in size at 5.97GB JSON format(3.7GB at CSV) and comprising 14,318,520 reviews, offers an empirical foundation for our system's algorithms.

The dataset is created to maintain the integrity of the original review submissions, thus reflecting genuine customer experiences and feedback trends. It encompasses a variety of metrics, including star ratings, review texts, images, product identification numbers (ASINs), and customer IDs. This diverse array of data points allows for a multi-faceted customer feedback analysis.

In processing this dataset, we avoid the limitations of analyzing mere samples, which could lead to biased insights. Instead, we employ a reasonable spectrum (5,000,000 reviews) of the data to drive a thorough exploration of customer satisfaction metrics, sentiment valuations, and purchasing patterns. Our analysis includes but is not limited to the following applications:

1. Evaluation of customer satisfaction levels and preferences as indicated by the distribution of star ratings across various products.
2. Sentiment analysis on the corpus of review text, extracting both explicit and implicit customer opinions and detecting trends that may inform future product developments.
3. Association of product ASINs with purchasing and review patterns to unveil high-performing products and identify opportunities for product range enhancement.
4. Predictive modeling using historical data patterns to forecast future purchasing behavior, enabling proactive inventory and marketing strategy adjustments.

The dataset's variables, such as timestamps and helpful vote counts, provide additional insight. For example, the timestamps analysis can reveal seasonality in product popularity. At the same time, the examination of 'helpful' votes on reviews may inform us about the impact of community engagement on purchasing decisions.

Leveraging this data ensures our recommendations are reflective of and responsive to the complex tapestry of consumer behavior and preferences. The insights gleaned from this analysis will be directly translatable into an enhanced, personalized shopping experience for the users of our platform.

- **Cloud Architecture & Security**

- **Recommendation Algorithm**

The backend code sets up a Flask application to deliver personalized product recommendations to users based on their past interactions with items. It leverages Singular Value Decomposition (SVD), a collaborative filtering technique, to analyze user-item interactions and make predictions about which items a user might like.

Firstly, the code initializes the Flask application and imports necessary libraries including boto3 for accessing AWS DynamoDB, pandas for data manipulation, and surprise for the recommendation algorithm. Then, it establishes a route `/recommend` where recommendations will be served upon request.

Upon receiving a request, the application queries an AWS DynamoDB table named 'amzn-reviews' to retrieve user-item interactions. These interactions are then transformed into a pandas DataFrame, where each row represents a user's review of a specific product, including information like user ID, product ID, rating, and other metadata. Before feeding the data into the SVD algorithm, it's preprocessed to fit the Surprise library's requirements. This involves setting up a Reader object to define the rating scale, renaming columns, dropping unnecessary ones, and transforming values where necessary to ensure compatibility with Surprise.

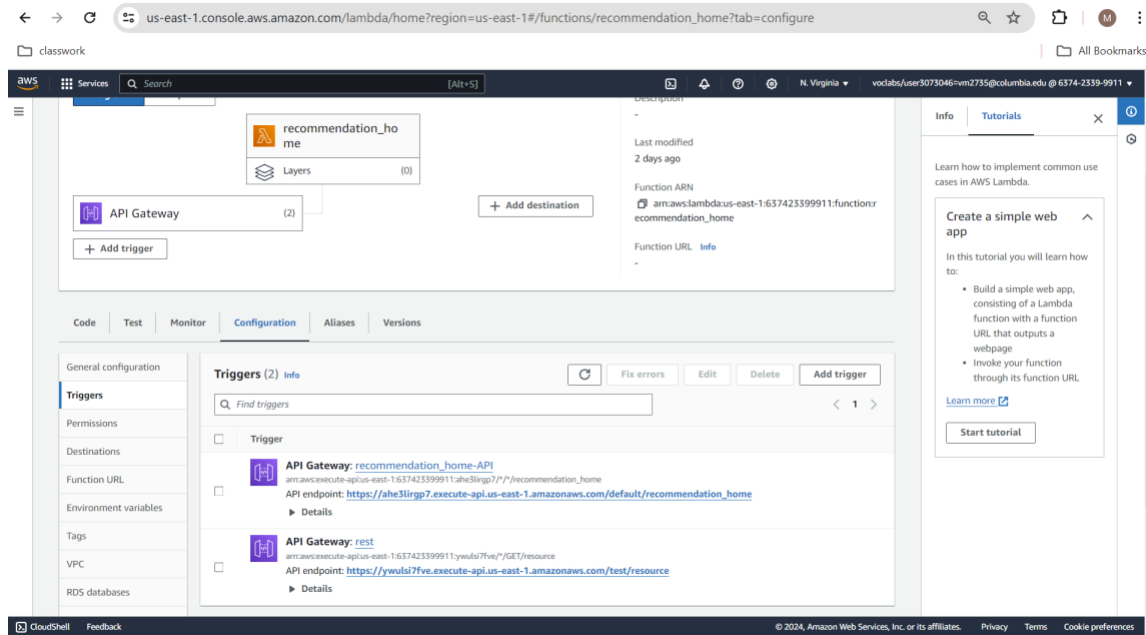
Following data preparation, the model is trained using a train-test split of the data, where a portion of the interactions are reserved for testing the model's accuracy. The SVD algorithm learns patterns from the training data, capturing underlying relationships between users and items. SVD is employed to analyze user-item interactions extracted from an AWS DynamoDB table and make predictions about which items a user might like. The algorithm learns patterns from historical user-item interactions during the training phase, capturing the underlying relationships between users and items. When a user sends a request for recommendations, the trained SVD model predicts ratings for items that the user has not yet interacted with, based on the learned latent factors. Finally, the top-N items with the highest predicted ratings are recommended to the user, providing personalized suggestions tailored to their preferences and past interactions (see *Appendix B* for code).

When a user sends a request to the `/recommend` endpoint with their user ID, the application identifies the items the user hasn't interacted with yet. It then employs the trained SVD model to predict ratings for these unseen items, and selects the top 5 items with the highest predicted ratings as recommendations for the user.

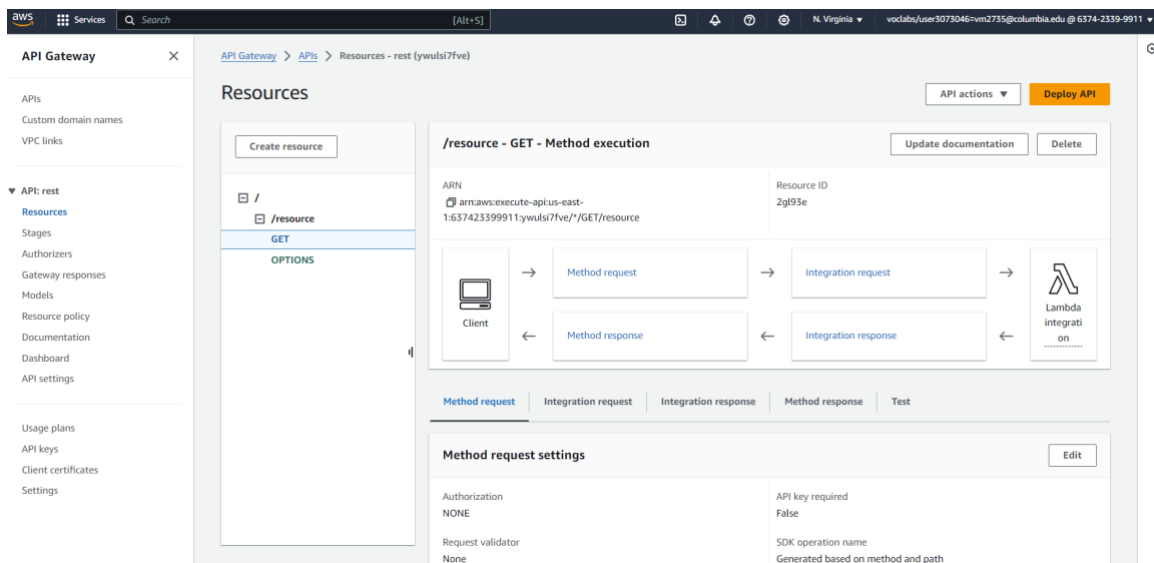
The recommendations are then formatted into JSON and returned as the response, including essential information such as product ID, title, text, and parent ASIN. Finally, the Flask application is run on the specified host and port, making the recommendation service available for users to access and receive personalized product suggestions based on their preferences and past interactions.

○ Service Delivery through Web

We utilized AWS Lambda to create a front end design that securely accesses the API endpoint created by our EC2 instance. We deployed two separate Lambda functions. The first Lambda function accesses the http endpoint created by our EC2 instance (see *Appendix C* for code). This function is connected to the AWS API Gateway to create a second API endpoint that is not http but rather https.

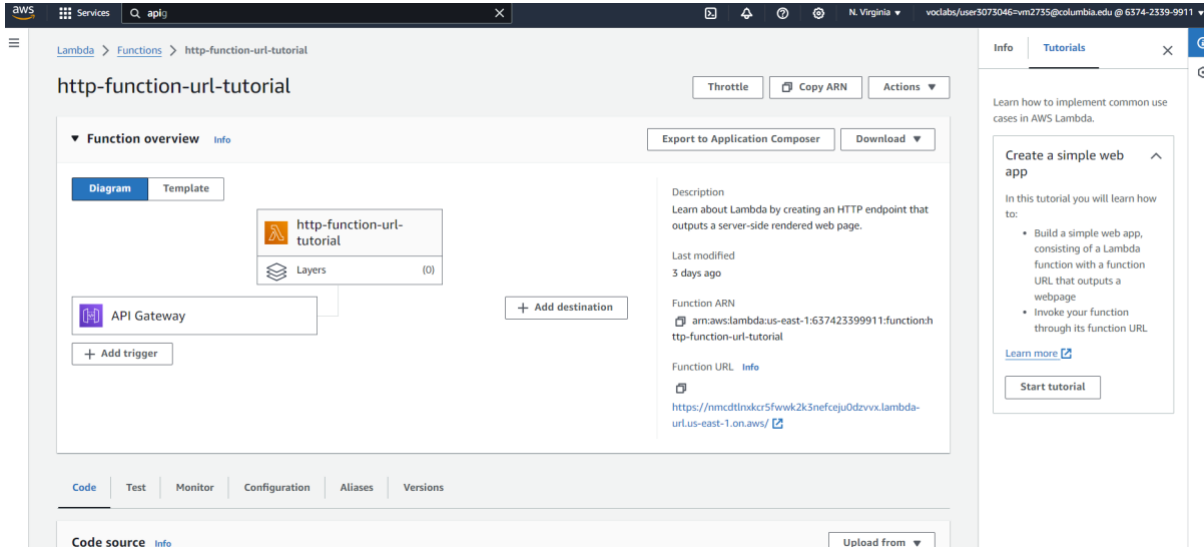


First Lambda Function; “recommendation_home-API” is just a test to make sure the function can open the http ML API. “rest” API creates a https API for this Lambda function.



API Gateway: Creates https API endpoint for first Lambda function. Deployed in the “test” stage as shown in API url.

Our second Lambda function creates a function URL. This Lambda function utilizes HTML and Javascript to create a front end web page for users to input a user_id from our database that displays the five recommendations (see *Appendix D* for code). For the scope of this project, two Lambda functions were necessary, as a Lambda function URL that is https cannot directly access the http API endpoint created by our EC2 instance. This generated a “Mixed Content” error. Using a mix of Lambda and AWS API Gateway we were able to bypass this issue to correctly access and display our data.



Second Lambda Function: Accesses https API made in the first Lambda function, creates https URL.

Beyond the scope of this project, this is not a good practice. For good security there are ways to upgrade our certificate for our EC2 instance to make the endpoint a https domain. AWS has a service called Certificate Manager that can easily do this, however due to the time constraint we created by incurring high costs in the Learner Lab, we chose to internally create a new API endpoint.

○ Networking Configuration

Our cloud-based recommendation system is securely hosted within an Amazon Web Services (AWS) Virtual Private Cloud (VPC), ensuring an isolated environment for all networked components. The VPC is configured across two Availability Zones for enhanced reliability and includes both public and private subnets. Public subnets facilitate access to user-facing services like the Amazon API Gateway, while private subnets protect backend processes such as EC2 instances and AWS Lambda functions. Internet Gateway (IGW) provides internet access, and a NAT Gateway allows secure outbound internet connections from the private subnets without direct exposure.



We've also employed VPC endpoints for direct, private connections to essential AWS services like S3 and DynamoDB, enhancing security and performance. Specifically, security groups are rigorously defined to control inbound and outbound traffic; for instance, EC2 instances are configured to allow only HTTP and SSH traffic, ensuring that all other unauthorized accesses are blocked. Additionally, Network ACLs provide subnet-level traffic filtering to further secure our system against potential threats.

○ Security Configuration

To ensure robust security within our cloud-based recommendation system, we have implemented crucial measures using the features available through our AWS account. Our system's security is primarily upheld by meticulously defined Security Groups and S3 bucket policies. The Security Groups are configured to tightly control access; for EC2 instances, this includes creating rules that only allow HTTP, HTTPS, and SSH traffic, ensuring that these services are safeguarded against unauthorized network access. For S3 buckets, stringent policies are enforced that strictly limit access to the stored data, allowing only authenticated and authorized requests from our application (*Appendix E*). While these measures form the core of our current security framework, we plan to further enhance our security posture by incorporating AWS Identity and Access Management (IAM) roles, Multi-Factor Authentication (MFA), and AWS Key Management Service (KMS) for data encryption, as well as implementing Network Access Control Lists (ACLs) for additional traffic filtering as our account capabilities expand.

● Cost Analysis

AWS Pricing Calculator:

Current Scale Cost				Cost Projection			
Server	Cost/Month	Description	Total	Server	Cost/Month	Description	Total
S3	\$0.09	S3 Standard storage (4 GB per	\$847.08/year	S3	\$1.25	S3 Standard-IA storage (100 GB	\$1,779.36/year
EC2	\$36.87	Tenancy (Shared Instances) Operating system (Linux) Workload (Consistent, Number of instances: 1) Advance EC2 instance (t2.medium) Pricing strategy (On-Demand) EBS Storage amount (30 GB)		EC2	\$94.68	Tenancy (Shared Instances) Operating system (Linux) Workload (Consistent, Number of instances: 2) Advance EC2 instance (t4g.xlarge) Pricing strategy (EC2 Instance Savings Plans: 3yr No Upfront) EBS Storage amount (50 GB)	
VPC NAT gateway	\$33.03	Number of NAT Gateways (1)		VPC NAT gateway	\$37.35	Number of NAT Gateways (1)	
DynamoDB	\$0.60	Uncompressed source file size for Import from Amazon S3 (4 GB)		DynamoDB	\$15.00	Uncompressed source file size for Import from Amazon S3 (100 GB)	

Google Cloud Pricing (GCP) Calculator:

Current Scale Cost					
name	quantity	region	service_id	sku	total_price, USD
Custom Instance Core running in Americas	1460	us-central1	6F81-5844-456A	ACBC-6999-A1C4	48.45966
Custom Instance Ram running in Americas	2920	us-central1	6F81-5844-456A	51E2-59BD-7A6E	12.98232
Storage PD Capacity	30	us-central1	6F81-5844-456A	D973-5D65-BAB2	0
Server Node	730	us-central1	C3BE-24A5-0975	04F7-CAFF-B2CA	474.5
SSD storage	4	us-central1	C3BE-24A5-0975	B307-B7C9-B4CC	0.6333
Backup storage	0	us-central1	C3BE-24A5-0975	E82F-4427-05A9	0
Networking Cloud Nat Gateway Uptime	1	global	E505-1604-58F8	32E2-4EFC-EF9F	1.022
Networking Cloud Nat Data Processing	4	global	E505-1604-58F8	015F-5732-FFF0	0.18
Networking Cloud NAT IP Usage	1	global	E505-1604-58F8	8515-9425-D2CE	3.65
Standard Storage US Regional	4	us-central1	95FF-2EF5-5EA1	E5F0-6A5D-7BAD	0
				Total Price:	541.43/month
					6497.16/year
Cost Projection					
name	quantity	region	service_id	sku	total_price, USD
Commitment v1: Cpu in Americas for 3 Year	2920	us-central1	6F81-5844-456A	BEB5-8DD9-9972	41.537
Commitment v1: Ram in Americas for 3 Year	11680	us-central1	6F81-5844-456A	5AA0-D2CC-1DA8	22.27376
Storage PD Capacity	50	us-central1	6F81-5844-456A	D973-5D65-BAB2	0.8
Server Node	438	us-central1	C3BE-24A5-0975	04F7-CAFF-B2CA	284.7
SSD storage	60	us-central1	C3BE-24A5-0975	B307-B7C9-B4CC	9.49949
Backup storage	0	us-central1	C3BE-24A5-0975	E82F-4427-05A9	0
Networking Cloud Nat Gateway Uptime	1	global	E505-1604-58F8	32E2-4EFC-EF9F	1.022
Networking Cloud Nat Data Processing	100	global	E505-1604-58F8	015F-5732-FFF0	4.5
Networking Cloud NAT IP Usage	1	global	E505-1604-58F8	8515-9425-D2CE	3.65
Standard Storage US Regional	100	us-central1	95FF-2EF5-5EA1	E5F0-6A5D-7BAD	1.76265
				Total Price:	369.74/month
					4436.88/year

Current Scale Cost						Cost Projection					
Service	AWS Monthly	AWS Yearly	GCP Monthly(Equivalent to AWS Description)	GCP Yearly	Description(AWS)	Service	AWS Monthly	AWS Yearly	GCP Monthly(Equivalent to AWS Description)	GCP Yearly	Description(AWS)
S3/Cloud Storage	\$0.09	-	\$0	-	4 GB Standard storage	S3/Cloud Storage	\$1.25 \$3 Standard-IA storage	-	\$1.76	-	100 GB Standard-IA storage
EC2/ Instances (Compute Engine)	\$36.87	-	\$61.44	-	t2.medium instance, 30 GB SSD gp2	EC2/ Instances (Compute Engine)	\$94.68 EC2 Instance Savings Plans: 3yr No Upfront	-	\$64.61 (Committed use discount options: 3 years)	-	t4g.xlarge instance projection, 50 GB SSD gp2
NAT Gateway	\$33.03	-	\$4.85	-	1 NAT Gateway	NAT Gateway	\$37.35	-	\$9.17	-	1 NAT Gateway
DynamoDB/Cloud Bigtable	\$0.60	-	\$475.13	-	4 GB data import	DynamoDB/Cloud Bigtable	\$15	-	\$294.2 (Committed use discount options: 3 years)	-	100 GB data import
Total AWS/GCP	-	\$847.08	-	-	-	Total AWS	-	\$1,779.36	-	-	-
GCP Equivalent	-	-	-	\$6,497.16	-	GCP Equivalent	-	-	-	\$4,436.88	-

➡ AWS vs Google Cloud – Current Scale Cost Analysis:

1. Data Storage Cost (S3/Cloud Storage): Google Cloud's Cloud Storage cost is minimal for a small amount of storage (4 GB), but if the storage needs are scaled up to 100 GB, Amazon's S3 offers a slightly lower price annually.
2. Data Processing Cost (DynamoDB/Cloud Bigtable): The largest cost difference is in database services, with AWS's DynamoDB being significantly cheaper (approximately 800 times) than GCP's Cloud Bigtable.
3. Machine Learning Processing Cost/ Compute Cost(EC2/Instances): Google Cloud charges seem higher (approximately 2 times) because it separates compute and storage (EBS), while AWS charges for EC2 include both compute and storage (EBS).

4. Networking and Content Delivery Cost (NAT Gateway): Google Cloud's total price for NAT Gateway services is cheaper (approximately 7 times) than AWS's offering.

Overall, the yearly comparison suggests that AWS offers a more cost-effective solution for database services (DynamoDB) and is competitive for storage when scaled to higher capacities (100 GB).

However, GCP offers substantial savings in networking (NAT Gateway) and might also be cost-effective if their machine learning/compute costs include additional services or features not present in AWS's offering. To make a fully informed decision, we must consider our specific use-case requirements, including performance, scalability, and additional features beyond cost.

➡ Cost Optimizations:

1. Reserved Instances/Savings Plans

- AWS: Committing to EC2 Instance Savings Plans could lower the costs significantly compared to on-demand prices.
- GCP: Applying Committed Use Discount Options for Compute Engine instances can yield substantial savings. Choosing a 3-year discount option costs only \$3 more than the no-commitment option(month-to-month payments), suggesting that longer-term use could lead to greater cost efficiencies. Therefore, exploring Committed Use Discount Options may further optimize costs on GCP.

2. Storage Optimization

- AWS: Implement S3 lifecycle policies to move data to cheaper storage classes like S3 Infrequent Access.
- GCP: Use Object Lifecycle Management to automatically transition objects to Nearline, Coldline, or Archive storage classes based on the age of the data.

3. NAT Gateway Usage

- AWS: To reduce costs associated with the NAT Gateway, consider using VPC Endpoints for AWS services, which allows private connections between the VPC and AWS services without using an Internet gateway, NAT devices, or a VPN connection.
- GCP: Evaluate using Google Cloud's Interconnect or Cloud VPN to minimize data processing costs through the NAT Gateway.

4. Scale to Fit

- Analyze usage patterns regularly and scale resources to match actual demand. This will prevent paying for unused capacity on both AWS and GCP.

5. Efficient Database Use

- AWS DynamoDB: The cost can be optimized by adjusting the provisioned read/write capacity to match actual usage patterns or by using DynamoDB Auto Scaling. Also, structuring data and choosing the right primary key can reduce read/write operations.
- GCP Cloud Bigtable: Since Bigtable pricing is based on node hours, optimize the instance size and node count according to the load. Use Bigtable's auto-scaling feature to adjust resources automatically.

6. Cost Monitoring and Management Tools

- AWS: Leverage AWS Cost Explorer to track and manage AWS spending.
- GCP: Use GCP's Cost Management tools to monitor and optimize expenses.

7. Commitment Use

- For AWS and GCP, try to predict consistent usage over a longer period and consider committing to 1 or 3-year plans, which offer better rates than month-to-month payments.

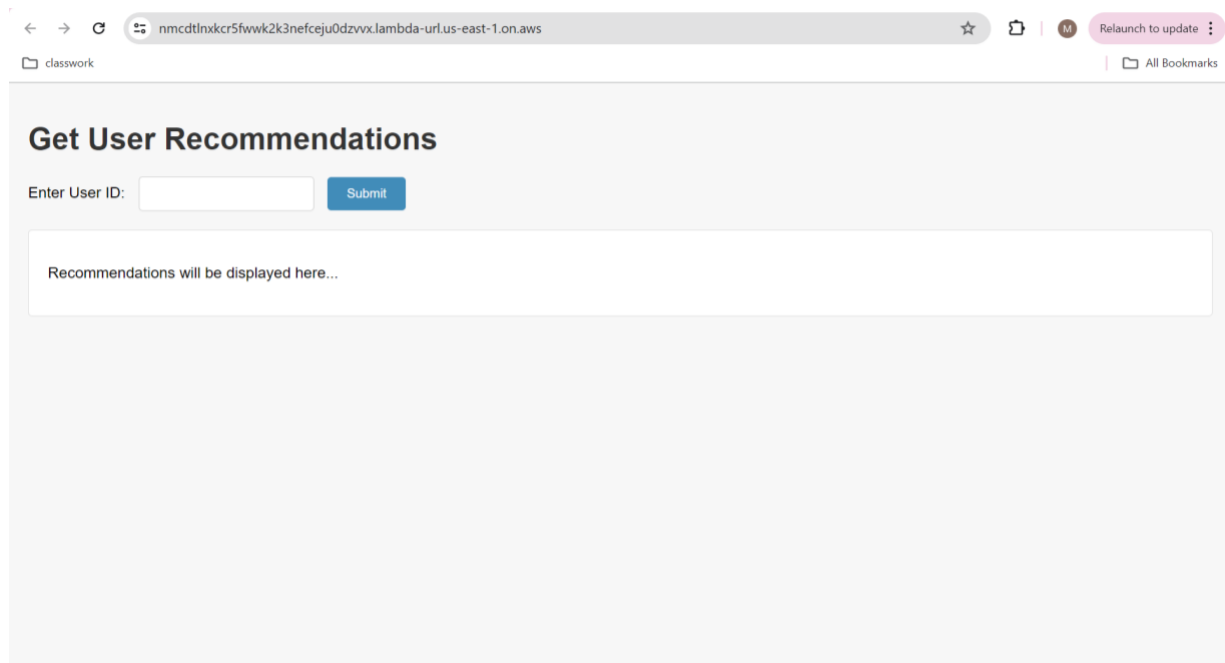
8. Additional Strategies

- Right Sizing: Use the right size for the instances on AWS and GCP to match the workload demands without overprovisioning.

Based on the cost data provided, it's evident that AWS currently appears to be more cost-effective at the scale assessed. However, for a long-term projection over 3 years, if GCP's Committed Use Discount Options are leveraged, it may offer competitive pricing, which could be appealing for sustained use. Despite this, AWS's wide array of services and mature ecosystem may present added value that transcends cost considerations. It's essential to weigh the broad service offerings, established infrastructure, and potential for enterprise agreements that AWS can provide.

When selecting a cloud provider, the decision should incorporate a comprehensive evaluation of each service's features, performance, and quality of support. Cost is a significant factor, but it should be balanced with these other considerations to determine which platform offers the best overall value for the project's specific requirements.

● Implementation & Demo



The screenshot shows a web browser window with the address bar displaying a URL ending in ".on.aws". The page has a header "classwork" and a "Relaunch to update" button. The main content area is titled "Get User Recommendations" and contains a form with the label "Enter User ID:" followed by a text input field and a "Submit" button. Below the form is a large rectangular box with the placeholder text "Recommendations will be displayed here...".

User inputs user_id and is presented with the top five recommendations based off of the recommendation score generated by the EC2 instance.

← → ↺ nmcctlxkcr5fwk2k3nefcej0dzvxx.lambda-url.us-east-1.on.aws ☆ 📁 19 Relaunch to update ⋮

classwork 📁 All Bookmarks

Get User Recommendations

Enter User ID:

Title	Parent Product ID	Product ID	Score	Review
The best hair treatment	B0BTTNFBZD	B00DS842HS	4.34	I bought this with the intention of making my own eczema cream, but my skin didn't like the mix of ingredients I used. Instead, I have been using this in my hair as a mask. I apply it all throughout my dry hair and comb it through with a wide-toothed comb to make sure all strands are coated, and then wrap it into a bun. I let it sit for at least twenty minutes, then shampoo and condition as usual. It makes my hair incredibly silky, shiny, and healthy. I also use it on my face after my nightly skin care routine over top of my moisturizer and it really helps with skin elasticity. It hasn't clogged my pores at all, and that's saying something because my skin is very sensitive. Oh, and I've also melted it and mixed it with a little flour, milk, and seasonings as a chicken batter, and rubbed it on steaks with salt and pepper before cooking! And it goes great in smoothies. It really is a multipurpose staple.
Long lasting energy, very focused & mellowed mood. No reason to make this stuff up.	B0B5G1SXX9	B074TQ24SY	4.33	Just ordered mine through Amazon excellent product. I'm an over the road truck driver bout 4yrs, & ive started drinking green tea powder "culinary blend" last few weeks. I did in fact get past the earthy taste which all i do is get a small bottle of juice, or already green tea & add in the powder then shake it up to taste. I can Agree that out of all the caffeine products on the market nothing has worked for me. Only thing that has come close to keeping me up on sluggish days, or nights were drinking mtn dew soda. Well, at the end of the day I knew drinking all that soda was not good for me so I really needed a different approach. After receiving the ceremonial blend green tea in the mail I was pretty excited. After seeing all the reviews I knew this had to be very good stuff. Day 1, I noticed same if not better energy level, then came along the focused & mellowed mood which I really need. Only negative side I can remotely lay out there is that my stomach had some uneasy feeling. I trusted it's judgement after all it's green tea & sometimes it can run through ya. But besides the energy & focused mellowed mood. I want to try to raise my metabolism & get rid of my lower stomach overhang. I'm kinda average in stomach size which really isn't that bad but I'm pushing for a flatter stomach and work on my health being on the road. Hopefully this product will stay around for my health alternative.

● Trials and Attempts

The original plan was to utilize Amazon Personalize, a prominent AWS service specifically designed for constructing recommendation systems. Amazon Personalize streamlines the recommendation process by creating distinct campaigns tailored to the dataset, thus enabling seamless recommendation generation. Unfortunately, due to constraints with IAM roles within the account, we encountered challenges in attaching policies to the existing lab role or creating new IAM roles. We then attempted to leverage the Factorization Machines algorithm via Amazon SageMaker, successfully training the model but facing obstacles in registering it. Consequently, we opted to deploy the recommendation system on an EC2 instance, circumventing the IAM role limitations and model registration issues. Despite requiring more manual setup compared to managed services, this approach enabled us to deliver personalized product recommendations to users based on their interactions, providing a viable solution amidst the encountered challenges.

Amidst our progress in utilizing Amazon Sagemaker, we quickly realized that our budget had been exhausted after running our machine learning models over the course of just one evening. Due to the Learner Lab updating the budget 8-12 hours after use, we were over budget and unable to continue our work. Luckily we were able to implement our model through an EC2 instance with minimal budget constraints on a separate student account. In the future we will be more careful with budget use with AWS and have realized how costly it can be to train models through AWS without prior knowledge of its cost.

● Conclusion

In conclusion, our project has not only achieved its objective of enhancing user shopping experiences through personalized recommendations but has also laid the groundwork for future enhancements. As cloud technologies evolve and our AWS account capabilities expand, we look forward to implementing additional security measures and optimizing our architecture to new heights. The success of this project stands as a testament to the efficacy of cloud computing in solving complex business challenges and transforming the landscape of e-commerce.

● Appendix

Appendix A

<https://amazon-reviews-2023.github.io/index.html>

- Our specific dataset centers on [review](#) for “Grocery_and_Gourmet_Food”

Appendix B - code.py, ran in EC2 instance to generate recommendations and API endpoint with flask.

```
from flask import Flask
app = Flask(__name__)

@app.route('/recommend', methods=['GET'])
def recommend():
    # Initialize the DynamoDB client
    dynamodb = boto3.client('dynamodb', region_name='us-east-1')

    # Query the DynamoDB table for 20 rows of data
    response = dynamodb.scan(
        TableName='amazon-reviews',
        Limit=5000000
    )

    # Extract the items from the response
    df = pd.DataFrame(response.get('Items', []))

    reader = Reader(rating_scale=(1, 5))

    df.drop(['id', 'timestamp'], axis=1, inplace=True)
    df = df.rename(columns={'asin': 'product_id'})
    df = df.rename(columns={'event_type': 'title'})
    # Extracting values from dictionary objects in DataFrame
    df['user_id'] = df['user_id'].apply(lambda x: x['S'])
    df['product_id'] = df['product_id'].apply(lambda x: x['S'])
    df['rating'] = df['rating'].apply(lambda x: float(x['S']))
    # Add similar lines for other fields you want to extract and transform
    df['parent_asin'] = df['parent_asin'].apply(lambda x: x['S'])
    df['title'] = df['title'].apply(lambda x: codecs.decode(x['S'], 'unicode_escape'), convert_dtype=False)
    df['text'] = df['text'].apply(lambda x: x['S'].replace('<br />', ''))

    data = Dataset.load_from_df(df[['user_id', 'product_id', 'rating']], reader)

    # Split the data into train and test sets
    trainset, testset = train_test_split(data, test_size=0.2)
    # Train the SVD algorithm
    algo = SVD()
    algo.fit(trainset)

    user_id = request.args.get('user_id')
    # Get all unique item ids
    all_product_ids = set(df['product_id'].unique())
    # Get item ids already reviewed by the user
    reviewed_product_ids = set(df[df['user_id'] == user_id]['product_id'].unique())
    # Get item ids not reviewed by the user
    not_reviewed_product_ids = all_product_ids - reviewed_product_ids

    # Make predictions for items not reviewed by the user
    user_predictions = algo.test([(user_id, product_id, 0) for product_id in not_reviewed_product_ids])

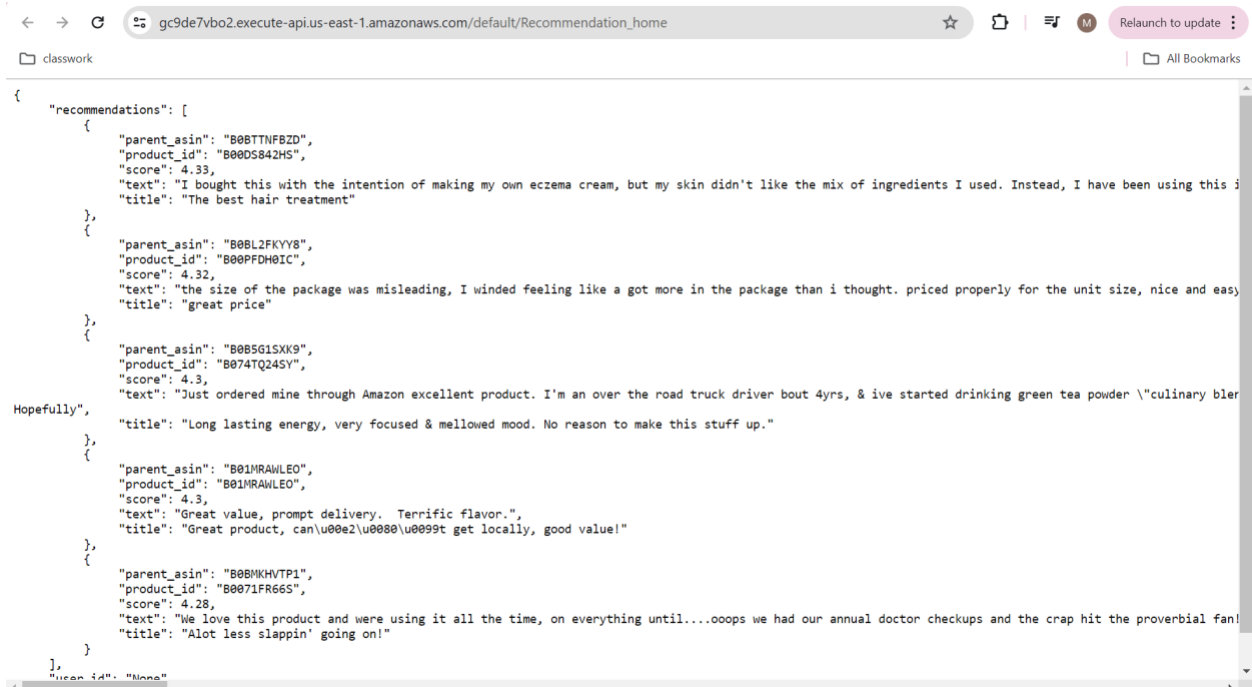
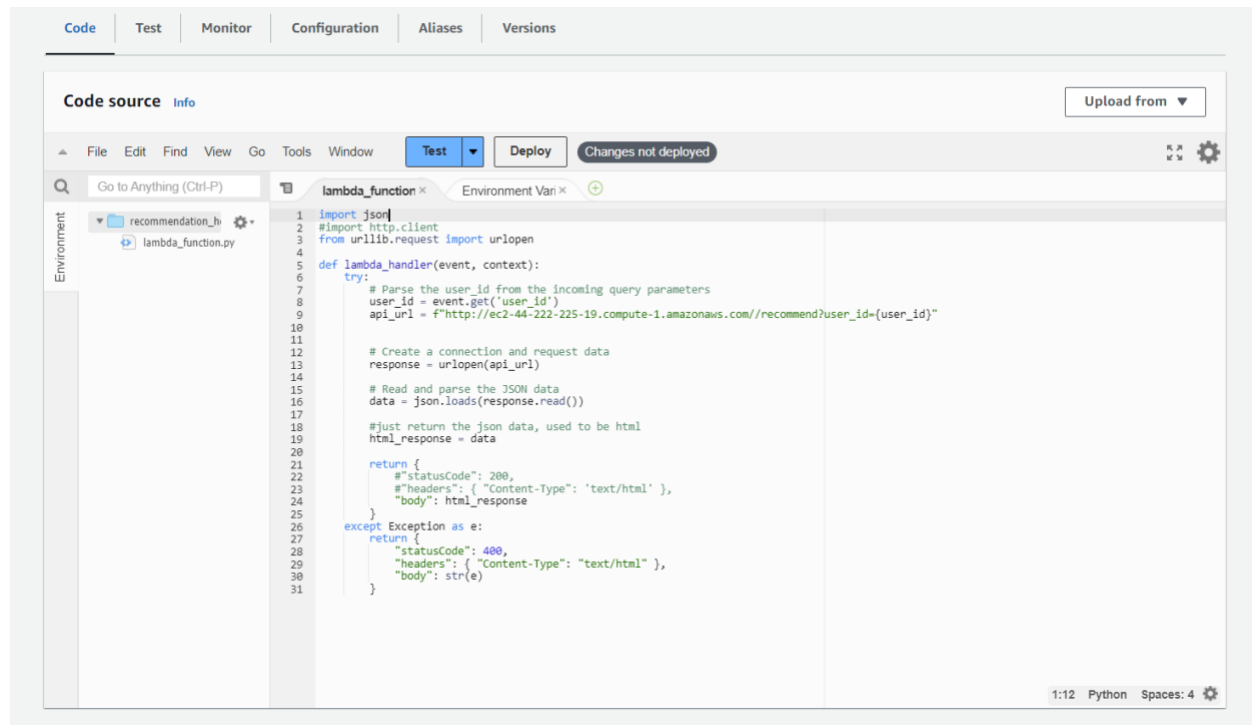
    # Get top 5 recommendations for the user with recommendation score
    top_n = sorted(user_predictions, key=lambda x: x.est, reverse=True)[:5]

    recommendations = []
    for pred in top_n:
        product_info = df[df['product_id'] == pred.id].iloc[0] # Get item info from DataFrame
        recommendation = {
            "product_id": pred.id,
            "score": round(pred.est, 2),
            "title": product_info['title'],
            "text": product_info['text'],
            "parent_asin": product_info['parent_asin']
        }
        recommendations.append(recommendation)

    return jsonify({"user_id": user_id, "recommendations": recommendations})

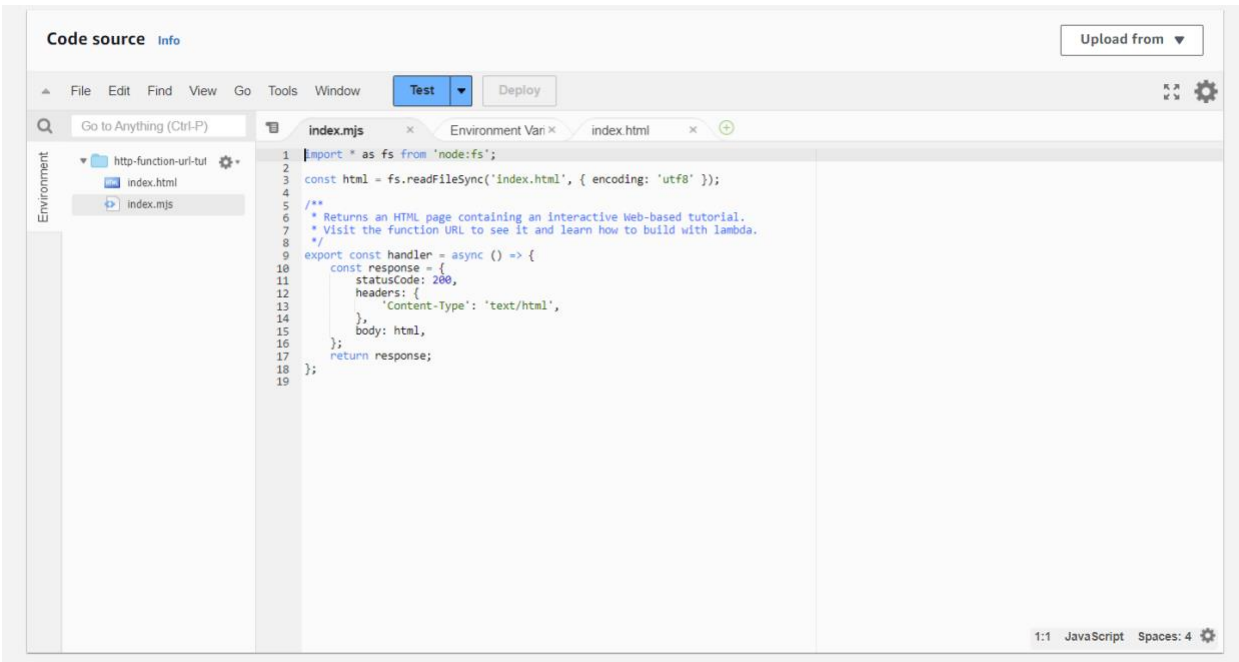
app.run(host='0.0.0.0', port=80)
```

Appendix C - First Lambda Function code (accesses http ML API endpoint) and returned JSON

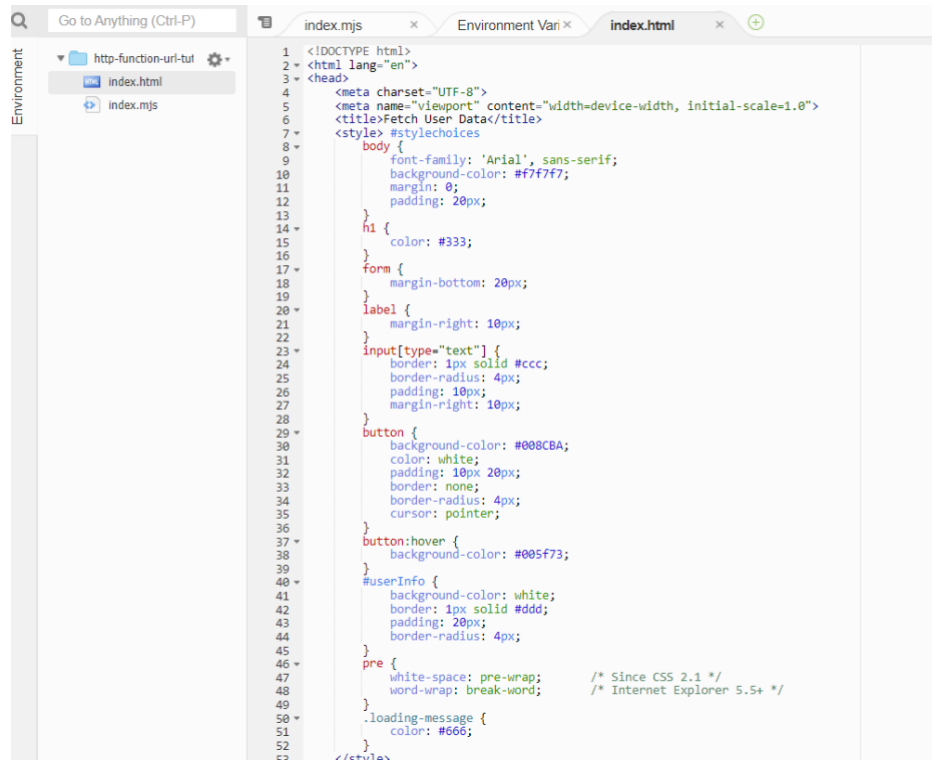


Appendix D - Second Lambda Function code (2 files; access https API endpoint and creates https URL)

index.mjs:



index.html: contains stylistic choices; `<script>` is written in javascript and accesses the second API endpoint and displays the JSON accordingly



```

Go to Anything (Ctrl-P)
index.html
index.mjs
Environment Var
index.html

24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108

</style>
</head>
<body>
<h1>Get User Recommendations</h1>
<form id="userForm">
  <label for="userId">Enter User ID:</label>
  <input type="text" id="userId" name="userId" required>
  <button type="submit">Submit</button>
</form>
<div id="userInfo">
  <p>Recommendations will be displayed here...</p>
</div>
<script>
  document.getElementById('userForm').addEventListener('submit', function(event) {
    event.preventDefault(); // Prevent the default form submission behavior
    userId = document.getElementById('userId').value;
    apiUrl = 'https://ywuis17fve.execute-api.us-east-1.amazonaws.com/test/resource?${encodeURIComponent(userId)}';
    fetch(apiUrl)
      .then(response => {
        if (!response.ok) {
          throw new Error('Failed to fetch data: ' + response.statusText);
        }
        return response.json();
      })
      .then(data => {
        // Start building the table HTML
        var tableHTML = "<table>";
        tableHTML += "<tr><th>Title</th><th>Parent Product ID</th><th>Product ID</th><th>Score</th><th>Review</th></tr>";
        // Loop through each recommendation and create a table row
        data.body.recommendations.forEach(function(rec) {
          tableHTML += "<tr>";
          tableHTML += "<td>${rec.title}</td>";
          tableHTML += "<td>${rec.parent_asin}</td>";
          tableHTML += "<td>${rec.product_id}</td>";
          tableHTML += "<td>${rec.score}</td>";
          tableHTML += "<td>${rec.text}</td>";
          tableHTML += "</tr>";
        });
        // Close the table HTML
        tableHTML += "</table>";
        // Set the table HTML inside the userInfo div
        document.getElementById('userInfo').innerHTML = tableHTML;
      })
      .catch(error => {
        document.getElementById('userInfo').textContent = error.message;
        console.error(error);
      });
  });
</script>
</body>
</html>

```

Appendix E - Security configurations

sg-0d08fdf47ce0bea77 - recommender_ec2
Actions

Details

Security group name recommender_ec2	Security group ID sg-0d08fdf47ce0bea77	Description Allow access to HTTP and SSH	VPC ID vpc-0bedaf2ddf491e34f
Owner 058264340496	Inbound rules count 3 Permission entries	Outbound rules count 1 Permission entry	

Inbound rules
Outbound rules
Tags

Inbound rules (3)
Manage tags
Edit inbound rules

Search
1

	Name	Security group rule...	IP version	Type	Protocol
	-	sgr-00866e4695153a8af	IPv4	HTTP	TCP
	-	sgr-0cb20aa12e9b517...	IPv4	HTTPS	TCP
	-	SecurityGroup sgr-0e5bfd44a27f86de1	IPv4	SSH	TCP

Permissions overview

Access

[View analyzer for us-east-1](#)

Block public access (bucket settings) Edit

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

Block *all* public access

✔ On

► Individual Block Public Access settings for this bucket

Bucket policy Edit Delete

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. [Learn more](#)



Public access is blocked because Block Public Access settings are turned on for this bucket