FINAL PROJECT REPORT

IS 698 – Special Topics in Information Systems

CLOUD COMPUTING

Under the Guidance of Samson Oni

Deploying a Scalable AWS Architecture with Infrastructure as Code

" BOOKVERSE "

Submitted By

Prathyusha Harish Kumar

JB24771

GitHub Link: *https://github.com/PrathyushaHarishKumar-JB24771/Bookverse-AWS-Project*

TABLE OF CONTENTS

1. INTRODUCTION

The modern applications can get cloud computing benefits such as scalability, dependability, and automation with almost zero operational overhead. This project BookVerse is a demonstration of the design and implementation power of cloud-based architecture on Amazon Web Services (AWS) based on Infrastructure as Code (IaC) principles. The application is a simple book discovery platform, with the main concern of still the construction of a powerful, secure backend infrastructure rather than a complicated frontend, which is also indispensable.

2. PROJECT OVERVIEW

BookVerse is a cloud-based web application designed to demonstrate a scalable, secure, and automated AWS architecture using Infrastructure as Code (IaC). The goal of this project was not to build a full-featured production application, but to design, deploy, and validate a real-world AWS infrastructure using multiple services working together.

The system was built using a combination of Terraform and AWS CloudFormation, allowing different layers of the infrastructure to be provisioned in a repeatable and controlled manner. Terraform was used to create the core networking layer (VPC, subnets, routing, and security groups), while CloudFormation was used to deploy application-level resources such as EC2, Auto Scaling, Application Load Balancer, RDS, S3, and Lambda.

Application architecture follows best practices by separating public and private resources. Public subnets host the Application Load Balancer and a bastion host for secure administrative access, while private subnets contain the EC2 web servers and the RDS MySQL database. This ensures that sensitive components like the database are never exposed to the public internet.

A static frontend for BookVerse is hosted on Amazon S3 and serves as the user interface for browsing book genres and titles. Backend functionality is demonstrated using AWS Lambda functions. One Lambda function logs file uploads from an S3 bucket into CloudWatch Logs, while another Lambda is exposed through API Gateway to provide a simple HTTP endpoint. In addition, AWS Step Functions were implemented to orchestrate a multi-step “order workflow,” showing how serverless services can be combined to model real application processes.

Interaction with AWS services was demonstrated in multiple ways:

- Through the AWS Management Console for verification
- Through the AWS CLI for resource management
- Through Python Boto3 scripts for programmatic access to EC2, S3, Lambda, and instance metadata

Database connectivity was validated by securely connecting from a bastion host to the RDS MySQL instance and executing Python scripts that query real data stored in the database. This confirms that the backend database is correctly provisioned, secured, and reachable from within the VPC.

Overall, this project demonstrates a complete AWS deployment lifecycle, including design, provisioning, validation, and automation, while following cloud security and scalability best practices. It reflects how modern cloud architecture is built and managed using Infrastructure as Code and managed AWS services.

3. ARCHITECTURE OVERVIEW

The architecture diagram represents the BookVerse AWS cloud infrastructure, where end users access the application through the internet, where traffic is routed to an Application Load Balancer (ALB) deployed inside a Virtual Private Cloud (VPC). The ALB distributes incoming requests across EC2 instances in private subnets, which are managed by an Auto Scaling Group to ensure high availability and fault tolerance.

The VPC is divided into two public subnets and two private subnets across multiple availability zones. The public subnets host the bastion host for secure administrative access and a NAT Gateway for outbound internet access from private resources. The private subnets contain the application EC2 instances and an Amazon RDS MySQL database, which is not publicly accessible and can only be reached internally. An S3 bucket is used for file storage, with AWS Lambda functions logging uploads to CloudWatch Logs. API Gateway and Step Functions orchestrate serverless workflows, coordinating multiple Lambda functions to validate orders, process payments, and complete transactions.

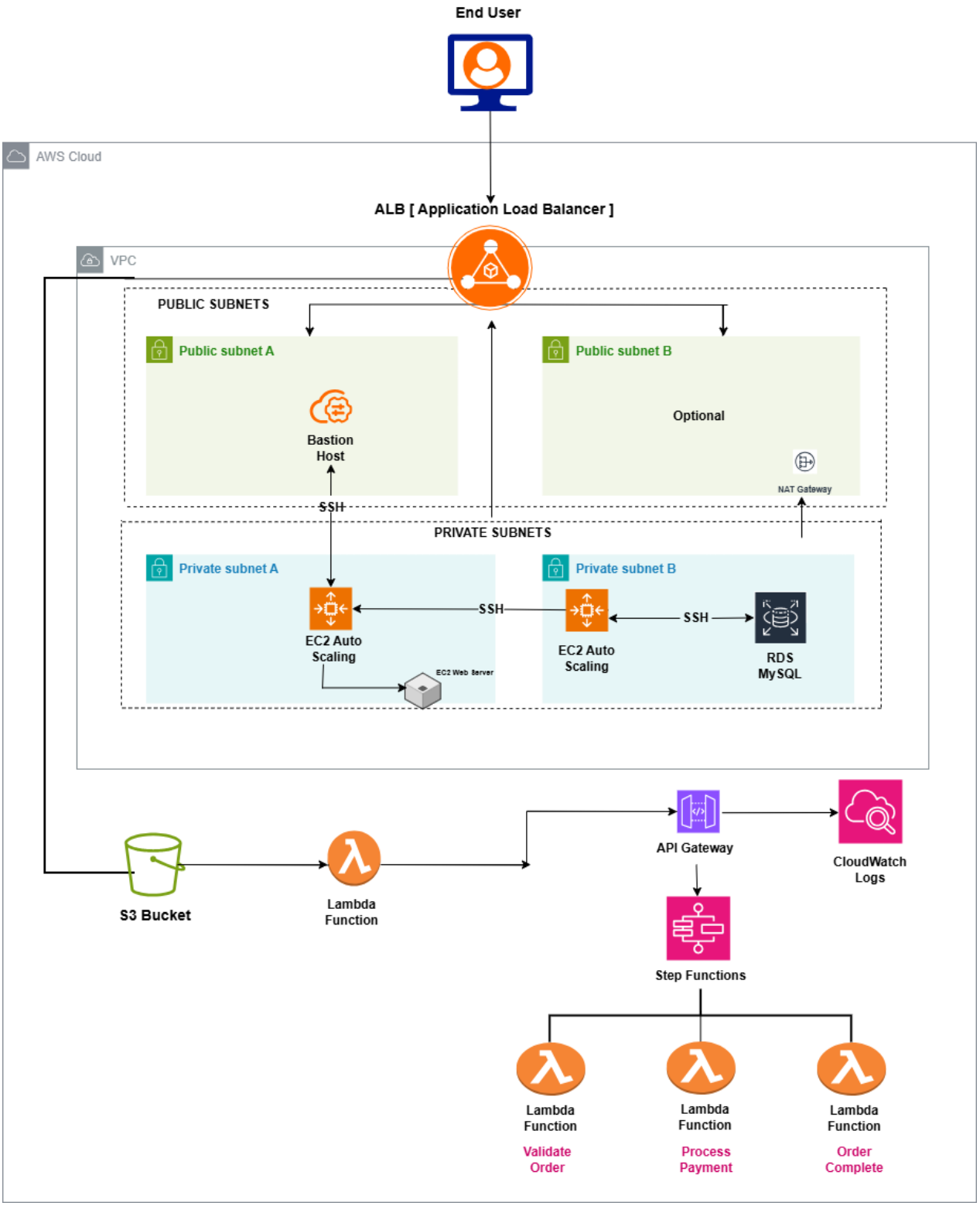


Fig : Architecture Diagram

4. TERRAFORM AND CLOUDFORMATION SCRIPTS

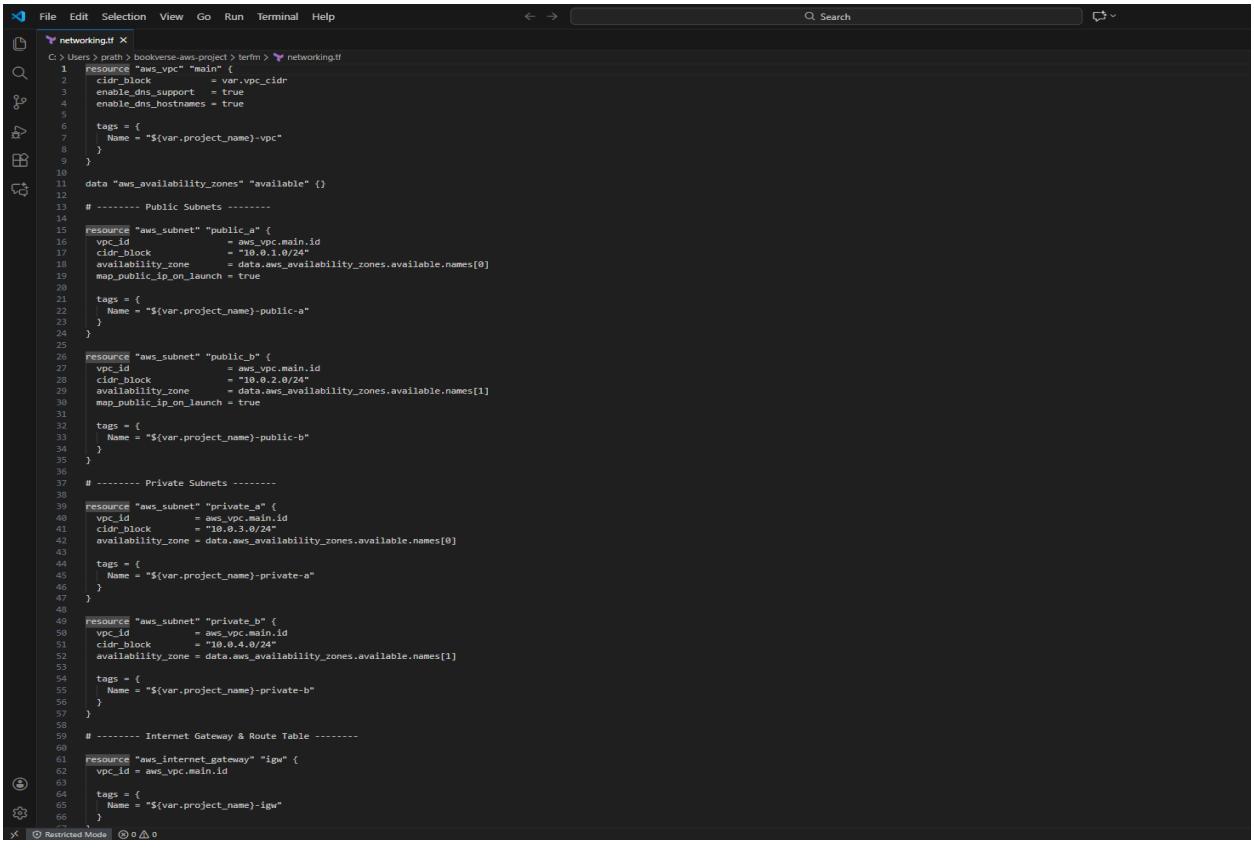
4.1 Terraform – Networking Infrastructure

Terraform was used to provision the core networking layer of the application. The main Terraform configuration file (main.tf) is responsible for creating and managing all foundational AWS network components in a repeatable and automated manner.

Using Terraform, the following resources were created:

- A custom Virtual Private Cloud (VPC) to isolate the BookVerse environment
- Two public subnets across different availability zones to host internet-facing resources
- Two private subnets for internal resources such as EC2 application servers and Amazon RDS
- Route tables and routing rules to enable controlled internet access
- Security groups defining inbound and outbound traffic rules

Terraform was chosen for networking because it allows clear separation of infrastructure layers and makes it easy to reuse or modify the network design without impacting application-level resources.



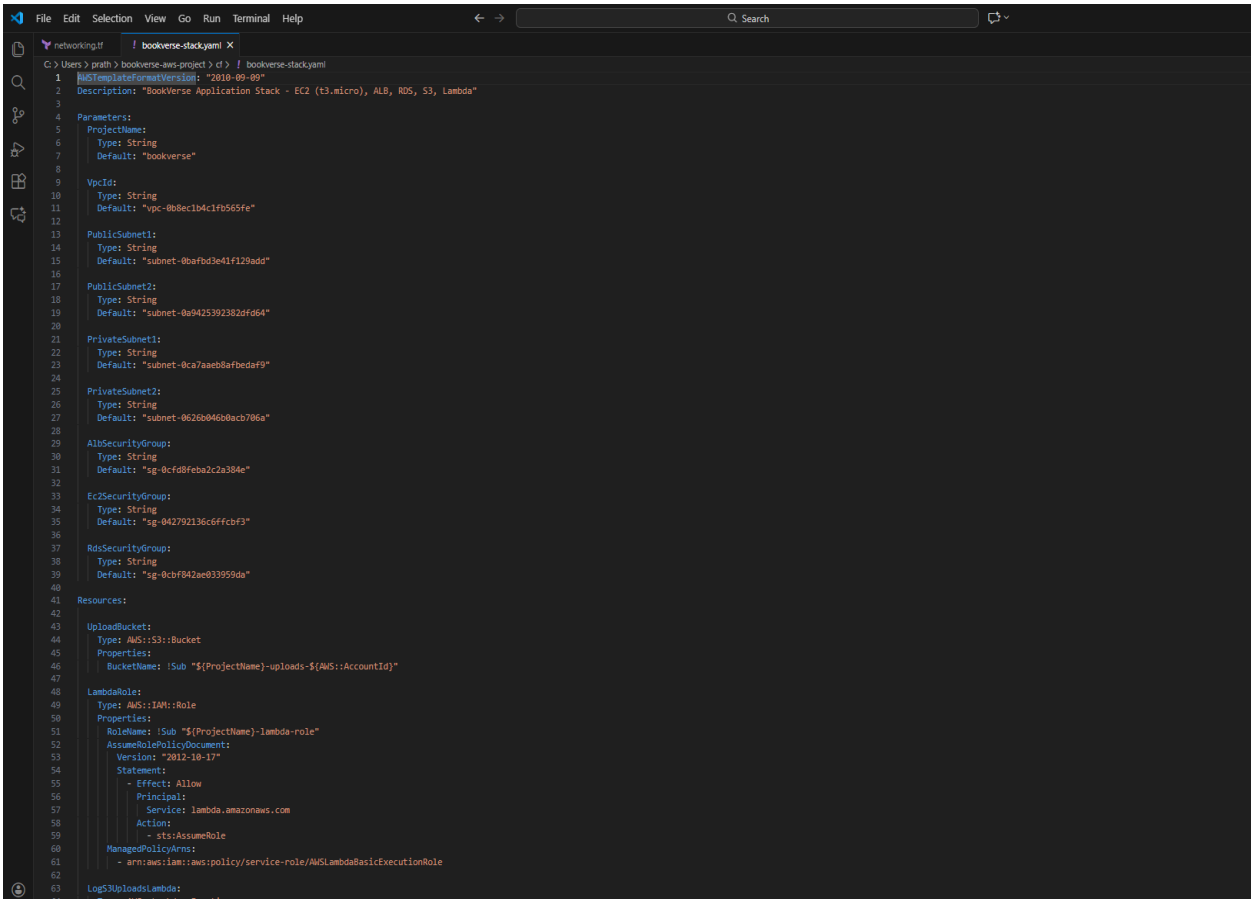
4.2 CloudFormation – Application and Service Deployment

AWS CloudFormation was used to deploy the application stack on top of the Terraform-provisioned network. The CloudFormation YAML template defines higher-level AWS services and their relationships.

The CloudFormation template was used to create:

- EC2 instances running the BookVerse web application
- An Application Load Balancer (ALB) to distribute traffic across instances
- An Auto Scaling Group to provide scalability and fault tolerance
- Amazon RDS (MySQL) as the database backend
- An S3 bucket for file uploads and static assets
- AWS Lambda functions for logging S3 uploads and API processing
- Required IAM roles and permissions

CloudFormation was chosen for this layer because it tightly integrates with AWS services and allows the full application stack to be launched, updated, or removed as a single unit.



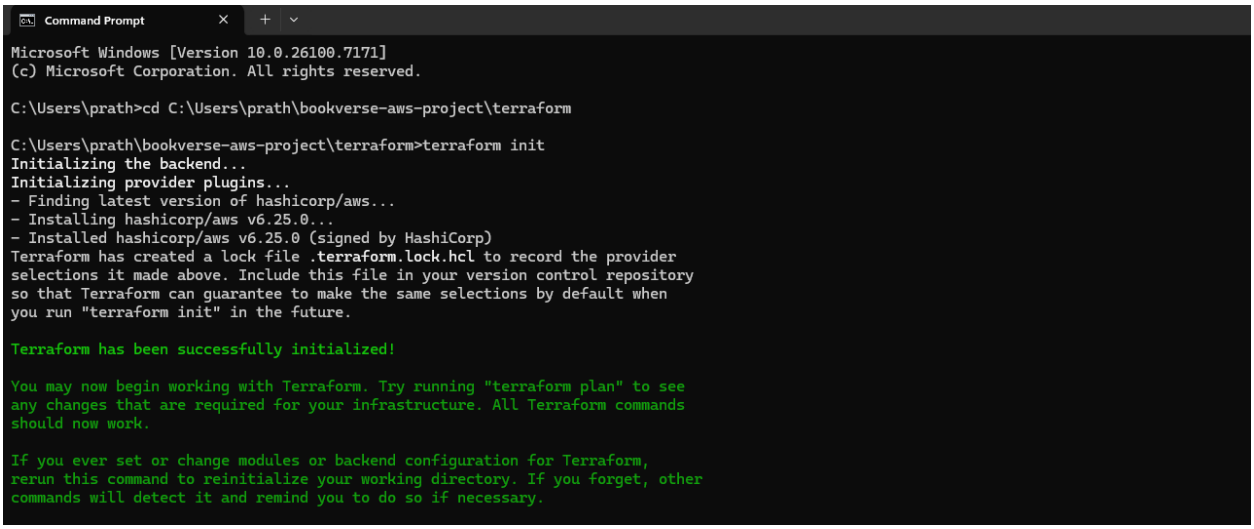
Script Files and Logs

All Terraform files, CloudFormation templates, and supporting scripts are stored in the GitHub repository. Command-line execution logs from Terraform, CloudFormation, AWS CLI, and Python Boto3 scripts are also included as screenshots to demonstrate successful deployment and testing.

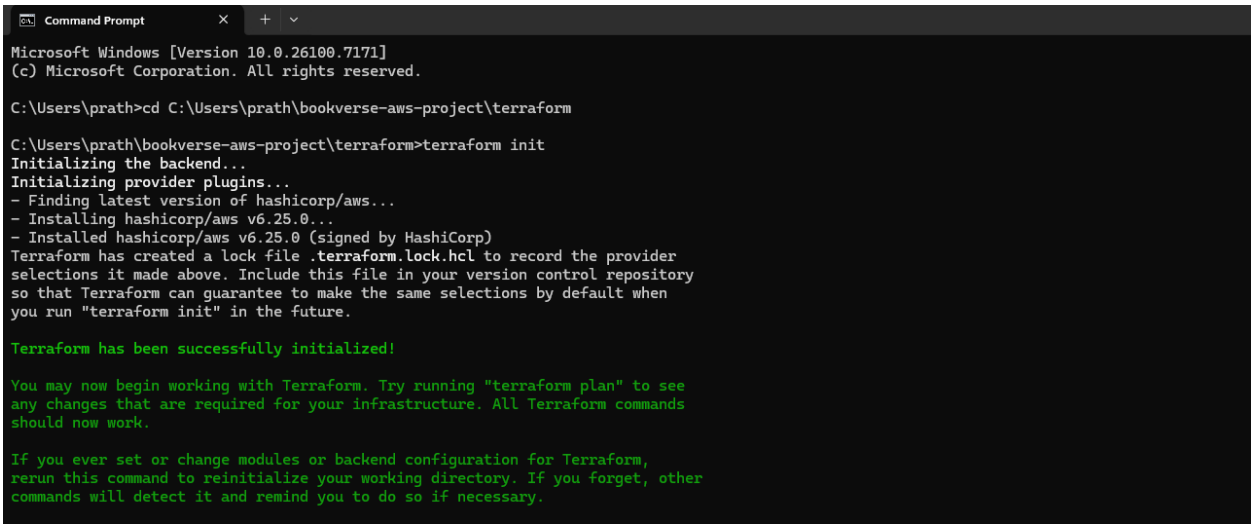
5. STEP-BY-STEP IMPLEMENTATION PROCESS (CONSOLE, CLI, BOTO3)

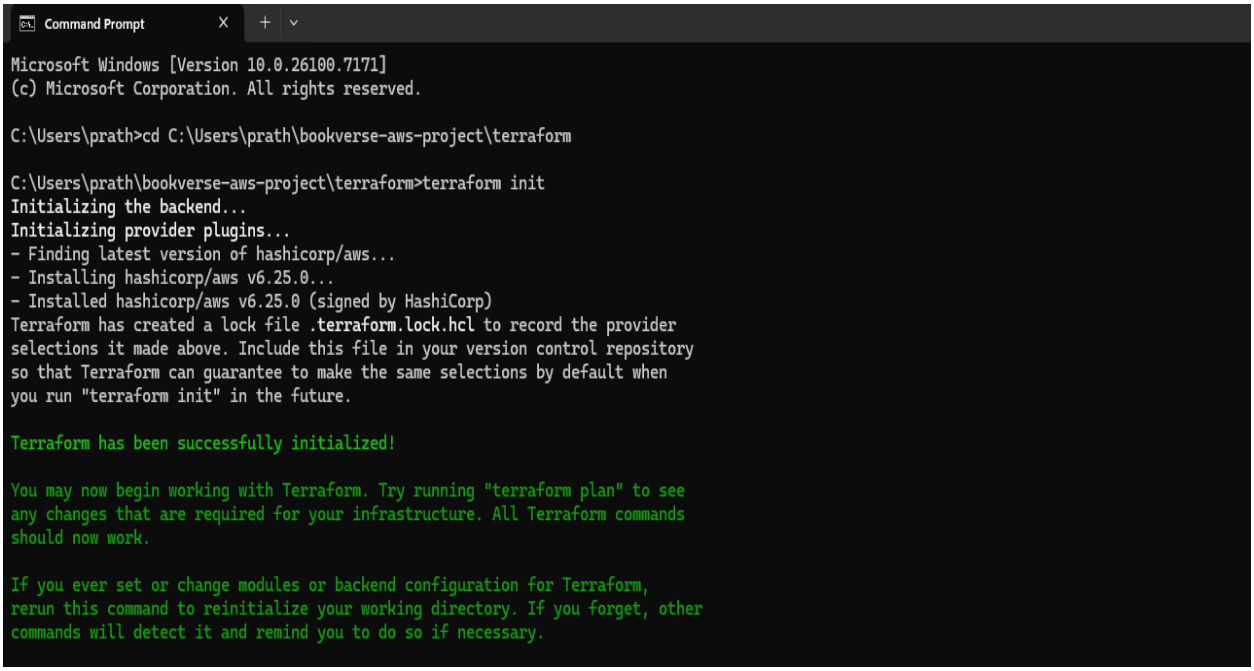
5.1 Networking with Terraform (CLI + Console)

- **Initialize Terraform (CLI) -** Ran *terraform init* in the Terraform directory to download AWS providers and set up the working directory.



- **Review and apply plan (CLI) -** Ran *terraform plan* to preview resources, then *terraform apply* to create:
    1. VPC (bookverse-vpc)
    2. Two public subnets and two private subnets
    3. Internet Gateway, route tables, and security groups.





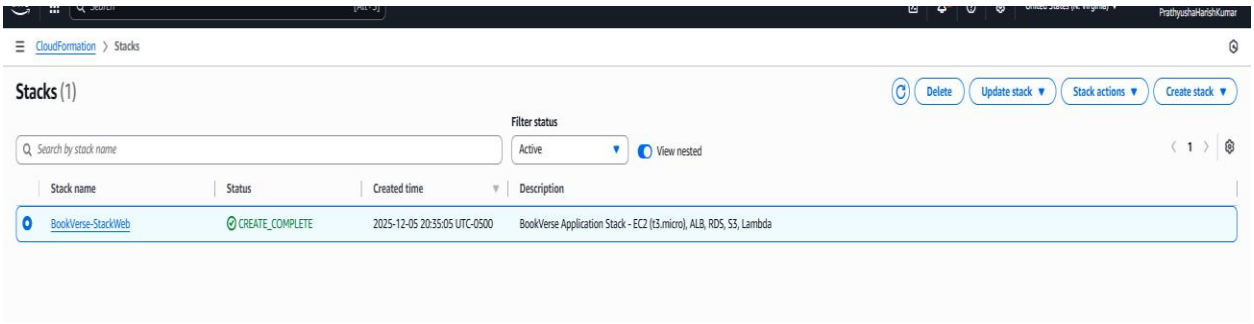
- **Verify networking resources (Console) -** Used the AWS Management Console → VPC to confirm:
    1. Correct VPC ID
    2. Public/private subnets in two AZs
    3. Security groups created by Terraform.

5.2 Deploy Application Stack with CloudFormation (CLI + Console)

- **Deploy CloudFormation stack (CLI)-** From the cf folder, ran *aws cloudformation deploy* with:
    1. Template: bookverse-stack.yaml
    2. Parameters: VpcId, PublicSubnet1/2, PrivateSubnet1/2, SG IDs from Terraform
    3. Capabilities: CAPABILITY_NAMED_IAM
    4. Stack name: CloudFormationStack-BookVerse.

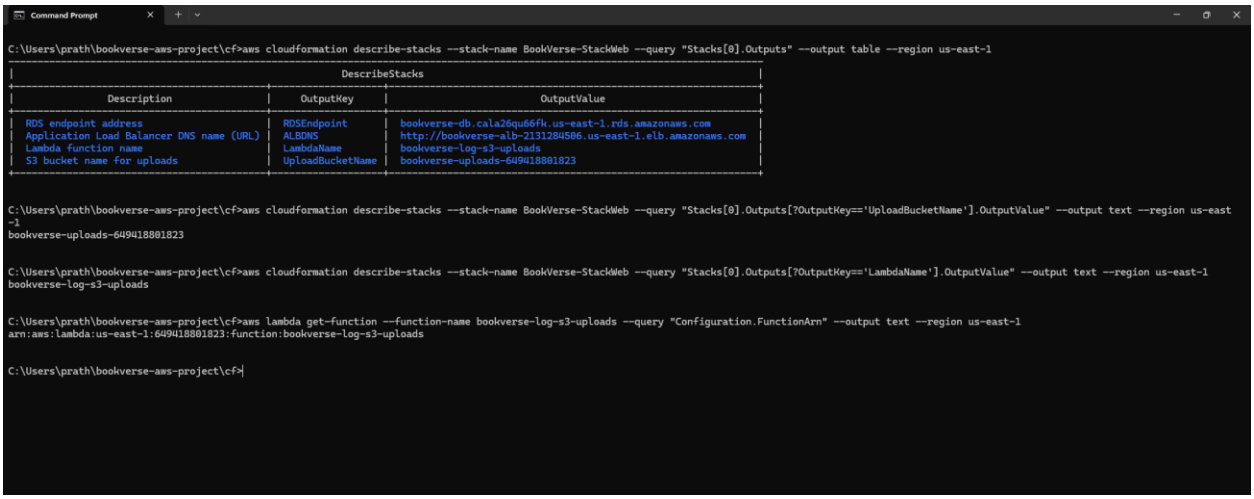
```
C:\Users\prath\bookverse-aws-project\cloudformation>cd C:\Users\prath\bookverse-aws-project\cloudformation

C:\Users\prath\bookverse-aws-project\cloudformation>dir
 Volume in drive C is Windows
 Volume Serial Number is EED3-89E9

 Directory of C:\Users\prath\bookverse-aws-project\cloudformation

12/05/2025  10:30 AM    <DIR>          .
12/02/2025  04:17 PM    <DIR>          ..
12/02/2025  03:33 PM                 0 .gitkeep
12/05/2025  10:30 AM             3,250 webapp.yaml
               2 File(s)          3,250 bytes
               2 Dir(s)  812,376,682,496 bytes free
```

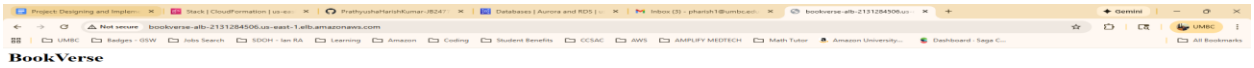
- **Validate stack creation** (CLI) - Used *aws cloudformation describe-stacks* to confirm status *CREATE_COMPLETE.*



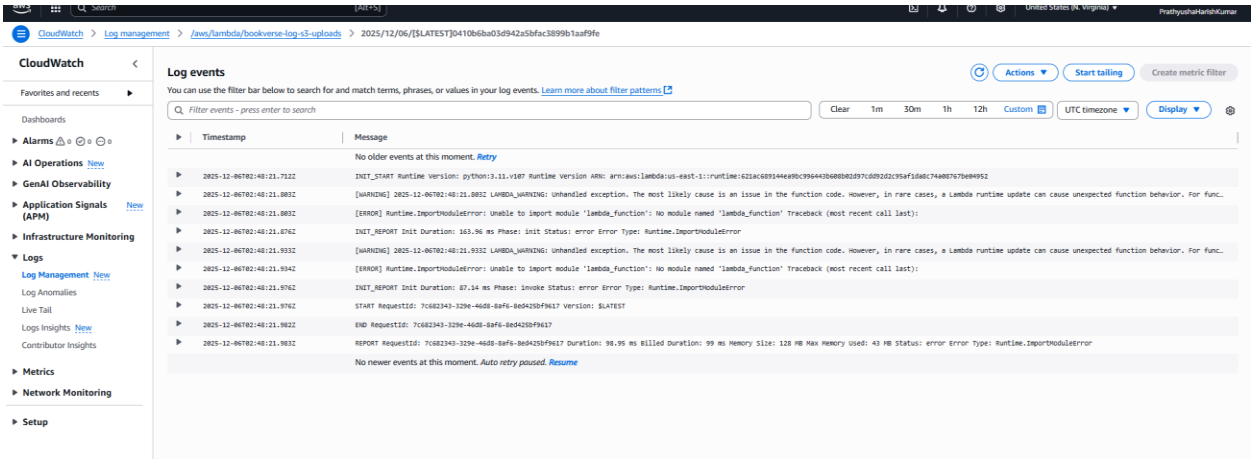
Retrieved outputs:

1. ALB DNS (ALBDNS)
2. RDS endpoint (RDSEndpoint)
3. Upload bucket name (UploadBucketName)
4. Lambda name (LambdaName).



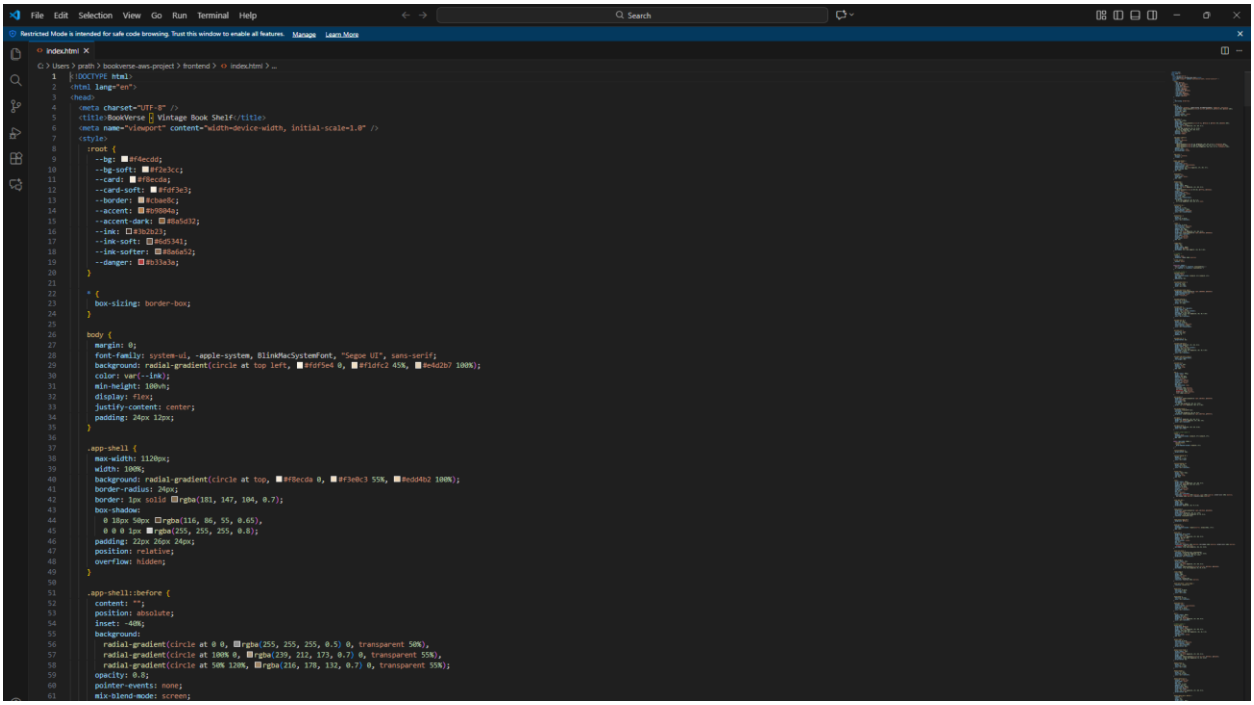


- **Inspect resources** (Console)
    1. EC2 console: Saw the app instance created by the Auto Scaling Group.
    2. RDS console: Verified MySQL instance running in private subnets.
    3. S3 console: Confirmed the uploads bucket.
    4. Lambda console: Verified *bookverse-log-s3-uploads* function.



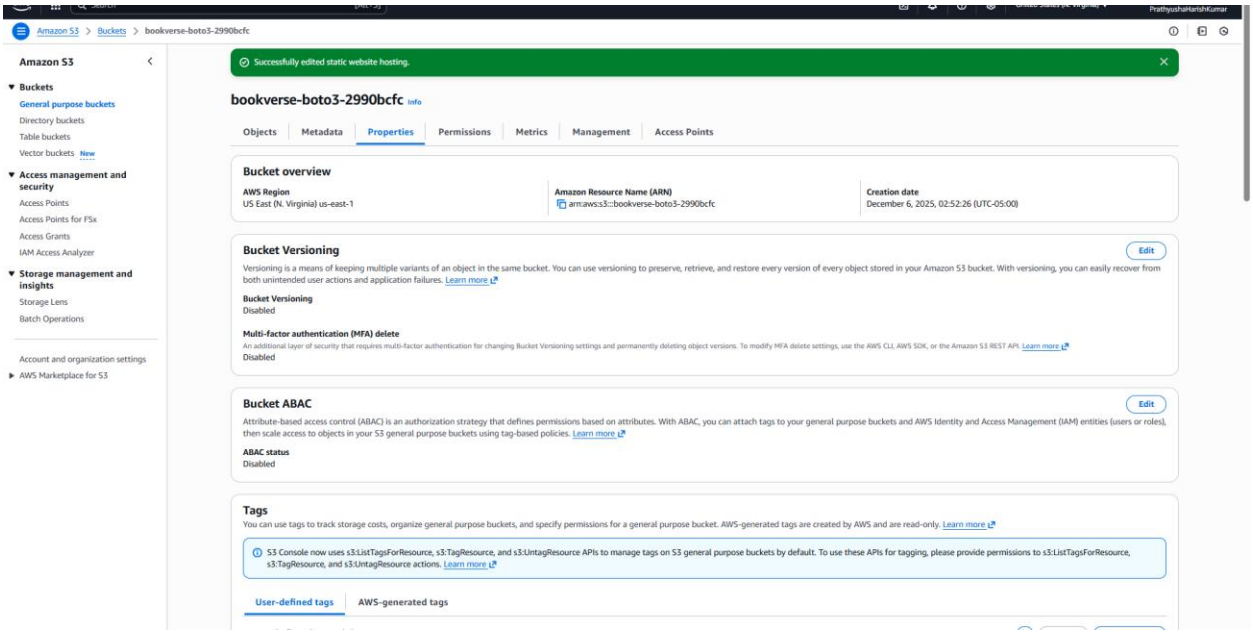
5.3 Static Web Frontend Deployment (CLI + Console)

- **Prepare frontend (Local)**
- Created a multi-page BookVerse index.html with:
    1. Landing page (“Welcome to BookVerse” + Explore button)
    2. Genre filter page (Romance / Fiction / Non-Fiction)
    3. Book details with price and “Buy” option
    4. Simple checkout form (name, email, phone, Apple Pay placeholder).



- **Upload frontend to S3 (CLI)**

  From the frontend directory, ran: *aws s3 cp index.html s3://bookverse-boto3-2990bcfc/index.html --region us-east-1*.

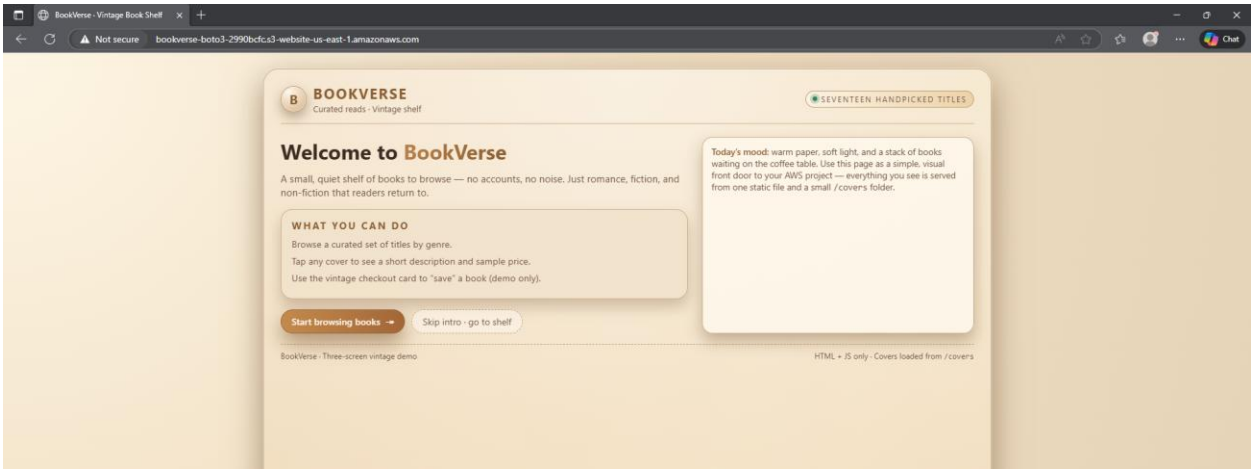


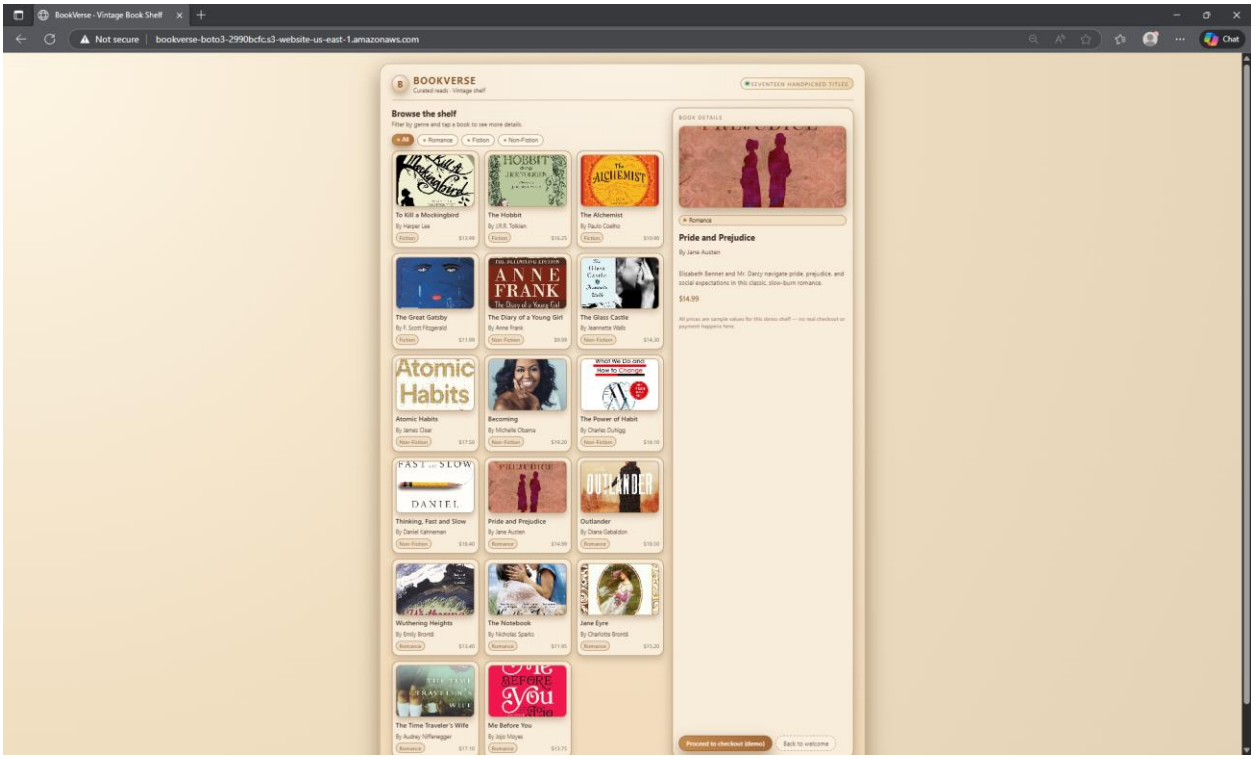
- **Enable static website hosting (Console)**
    1. Opened S3 bucket *bookverse-boto3-2990bcfc.*
    2. Enabled Static website hosting with index.html as the index document.
    3. Noted the website endpoint: *http://bookverse-boto3-2990bcfc.s3-website-us-east-1.amazonaws.com/.*
- **Test static site (Browser)**

  Opened the S3 website URL to confirm the interactive BookVerse UI loads correctly.

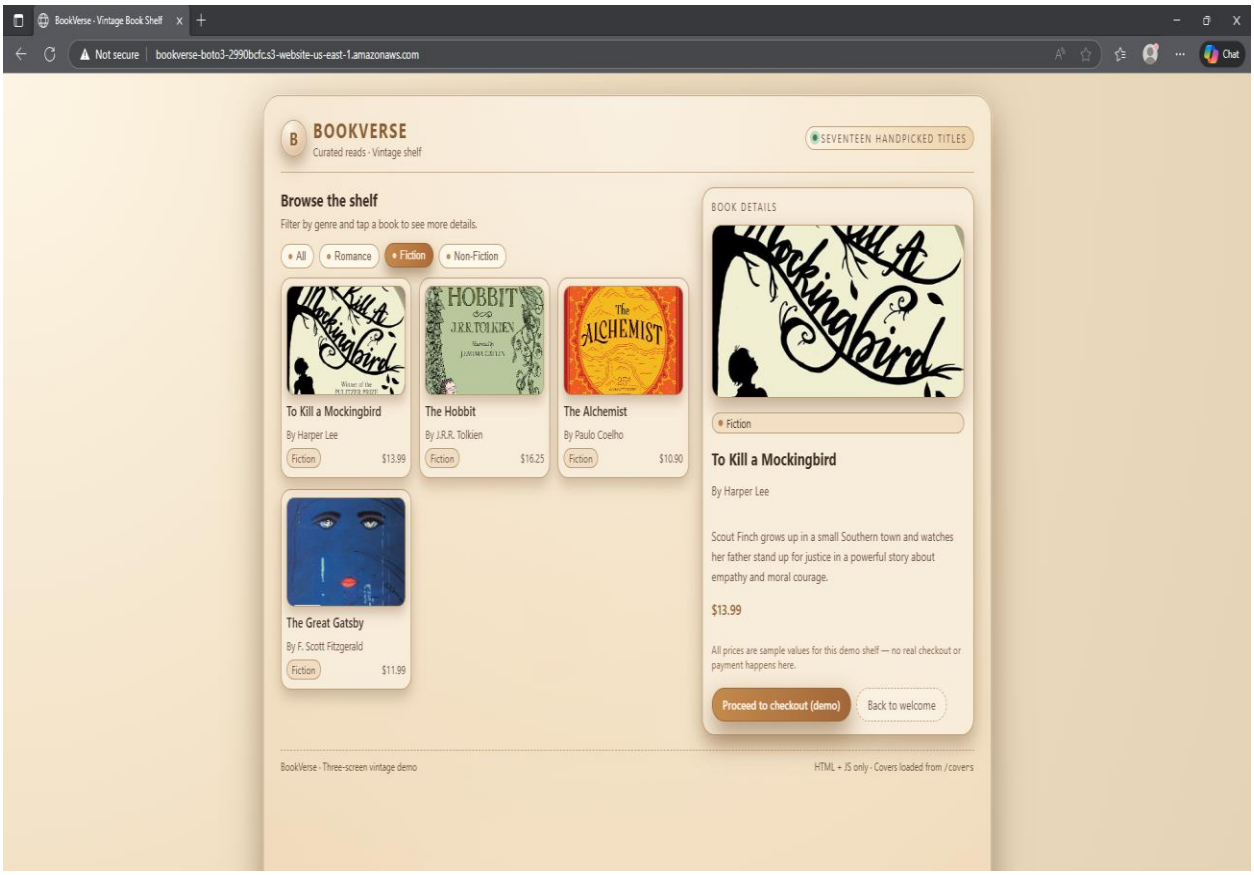
Home

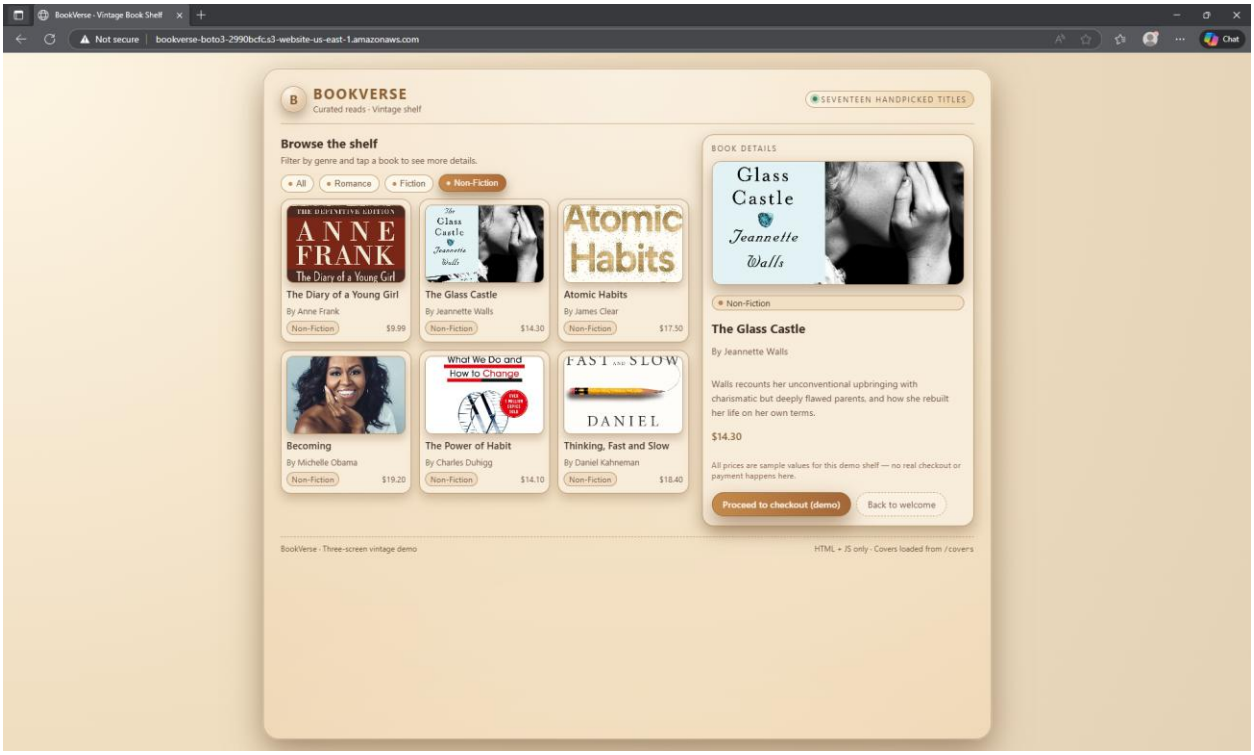
Link: http://bookverse-boto3-2990bcfc.s3-website-us-east-1.amazonaws.com



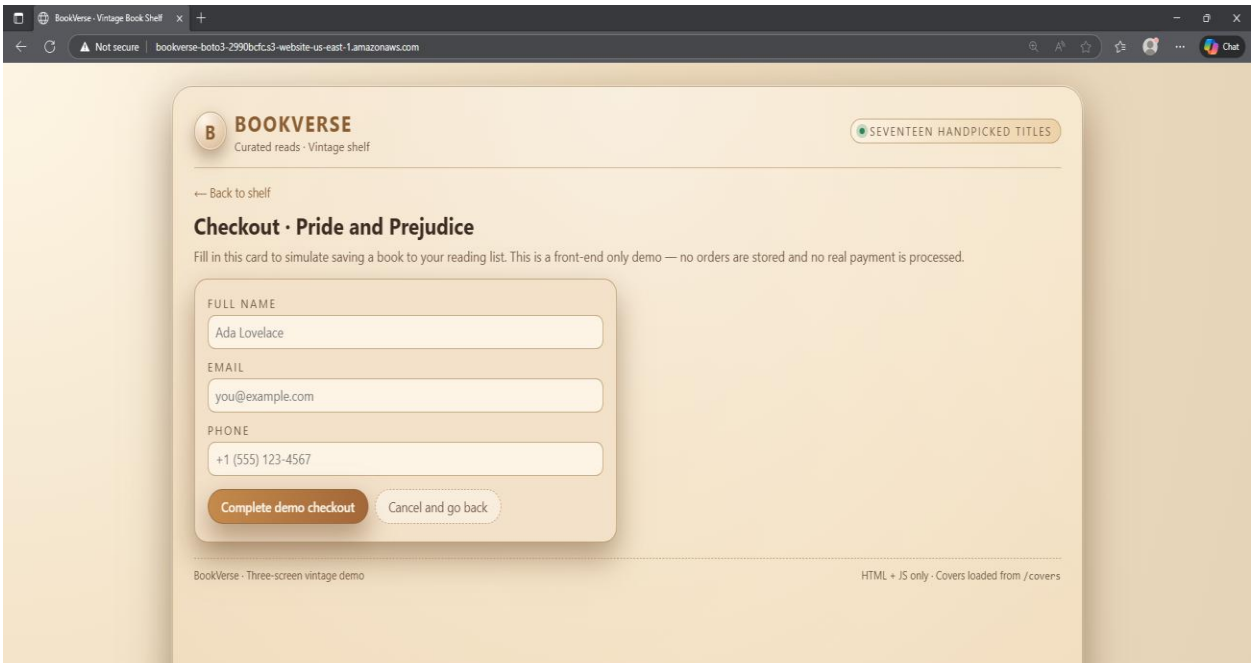


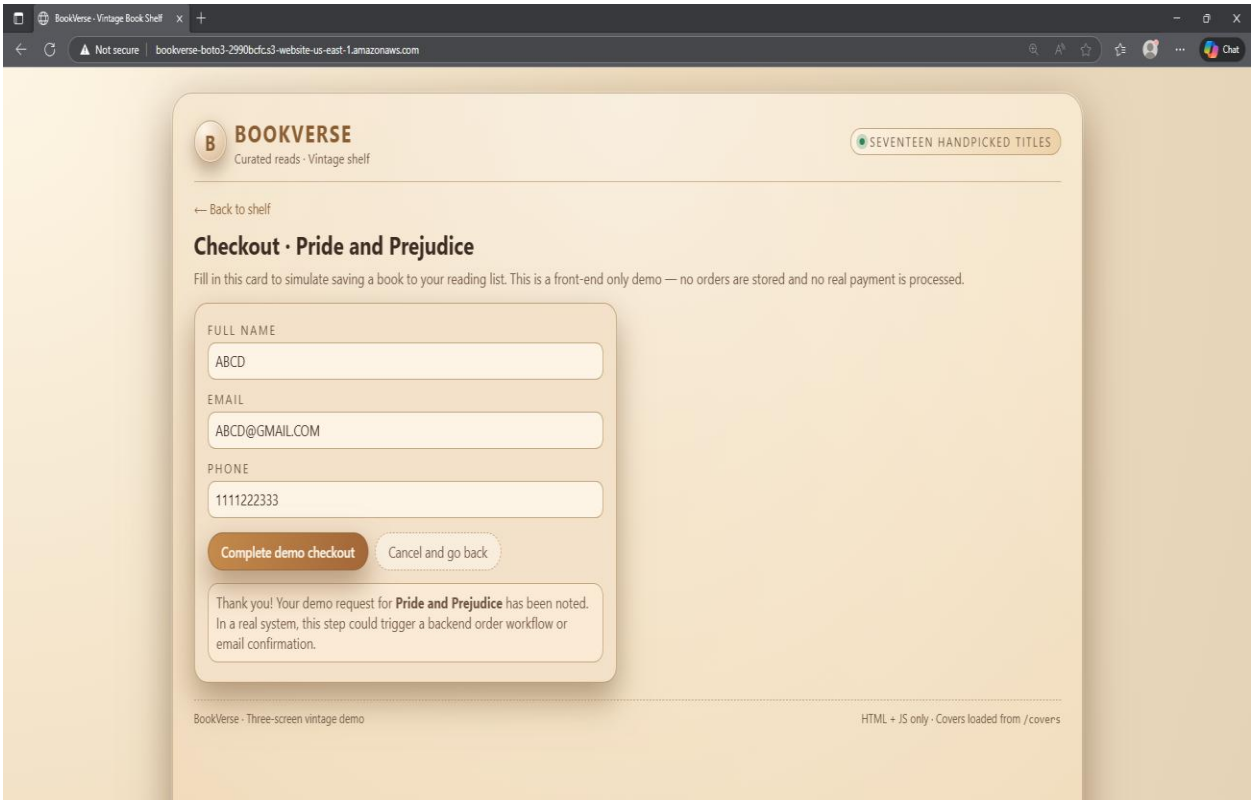
Genre Filtering


BookVerse - Vintage Book Shelf
Not secure
bookverse-boto3-2990bcfc.s3-website-us-east-1.amazonaws.com
Chat
BOOKVERSE
Curated reads · Vintage shelf
SEVENTEEN HANDPICKED TITLES
Browse the shelf
Filter by genre and tap a book to see more details.
All
Romance
Fiction
Non-Fiction
To Kill a Mockingbird
By Harper Lee
Fiction
$13.99
The Hobbit
By J.R.R. Tolkien
Fiction
$16.25
The Alchemist
By Paulo Coelho
Fiction
$10.90
The Great Gatsby
By F. Scott Fitzgerald
Fiction
$11.99
BOOK DETAILS
Fiction
To Kill a Mockingbird
By Harper Lee
Scout Finch grows up in a small Southern town and watches her father stand up for justice in a powerful story about empathy and moral courage.
$13.99
All prices are sample values for this demo shelf — no real checkout or payment happens here.
Proceed to checkout (demo)
Back to welcome
BookVerse · Three-screen vintage demo
HTML + JS only · Covers loaded from /covers


BookVerse - Vintage Book Shelf
Not secure
bookverse-boto3-2990bcfc.s3-website-us-east-1.amazonaws.com
Chat
BOOKVERSE
Curated reads · Vintage shelf
SEVENTEEN HANDPICKED TITLES
Browse the shelf
Filter by genre and tap a book to see more details.
All
Romance
Fiction
Non-Fiction
The Diary of a Young Girl
By Anne Frank
Non-Fiction
$9.99
The Glass Castle
By Jeannette Walls
Non-Fiction
$14.30
Atomic Habits
By James Clear
Non-Fiction
$17.50
Becoming
By Michelle Obama
Non-Fiction
$19.20
The Power of Habit
By Charles Duhigg
Non-Fiction
$14.10
Thinking, Fast and Slow
By Daniel Kahneman
Non-Fiction
$18.40
BOOK DETAILS
Non-Fiction
The Glass Castle
By Jeannette Walls
Walls recounts her unconventional upbringing with charismatic but deeply flawed parents, and how she rebuilt her life on her own terms.
$14.30
All prices are sample values for this demo shelf — no real checkout or payment happens here.
Proceed to checkout (demo)
Back to welcome
BookVerse · Three-screen vintage demo
HTML + JS only · Covers loaded from /covers

Purchasing book





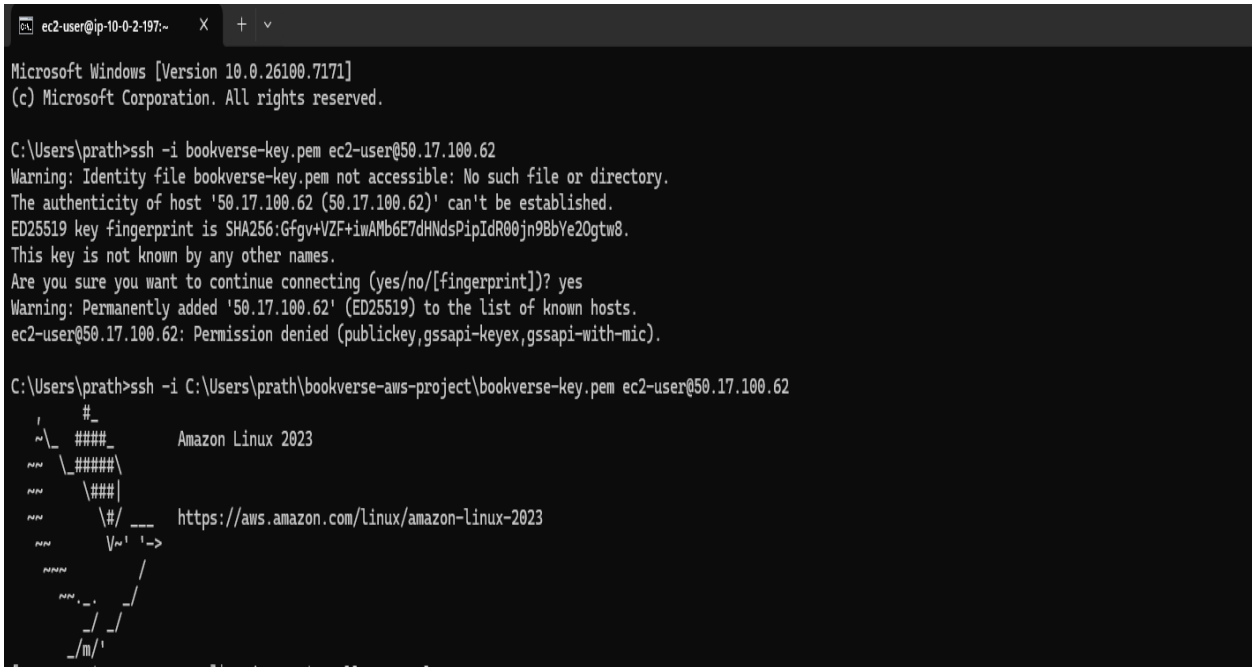
A Demo Video is uploaded in Github.

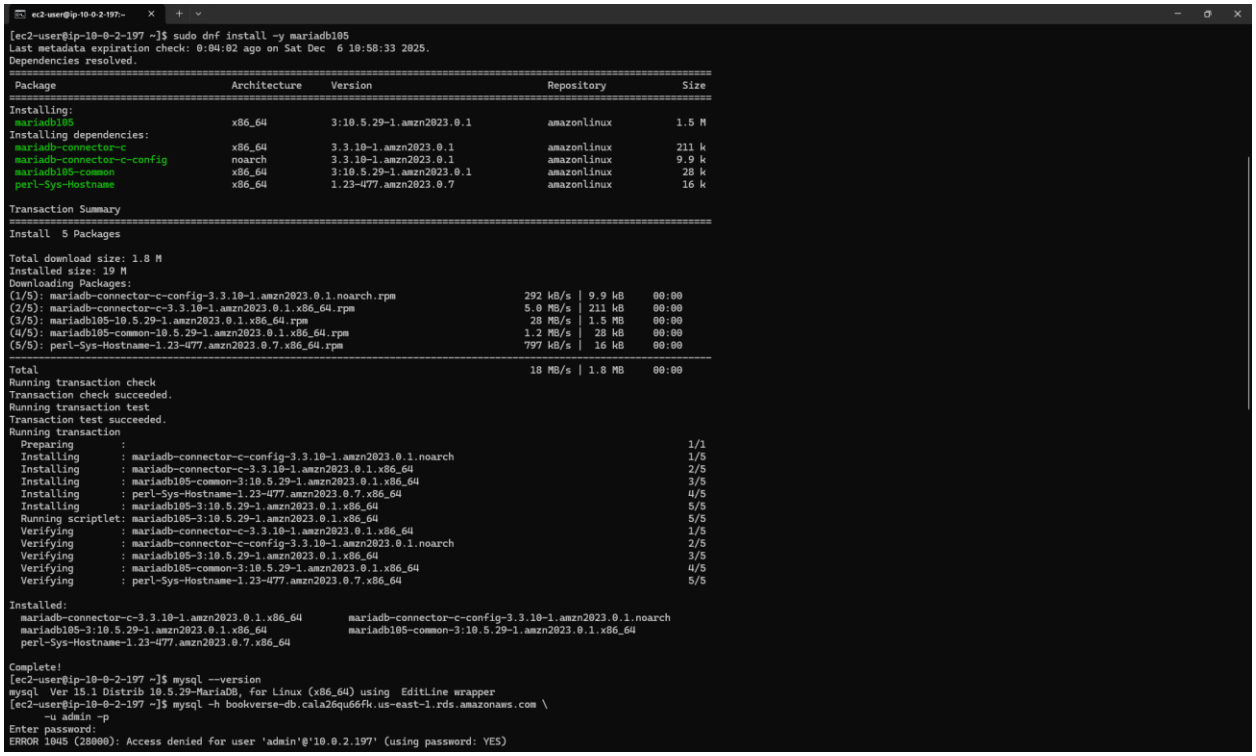
5.4 Bastion Host & RDS Connectivity (Console + SSH)

- **Launch bastion EC2 instance (Console)-**
    1. Launched a t3.micro in a **public subnet** using the same VPC.
    2. Attached a security group allowing SSH (22) from your IP.
    3. Used the Terraform VPC and subnet IDs.

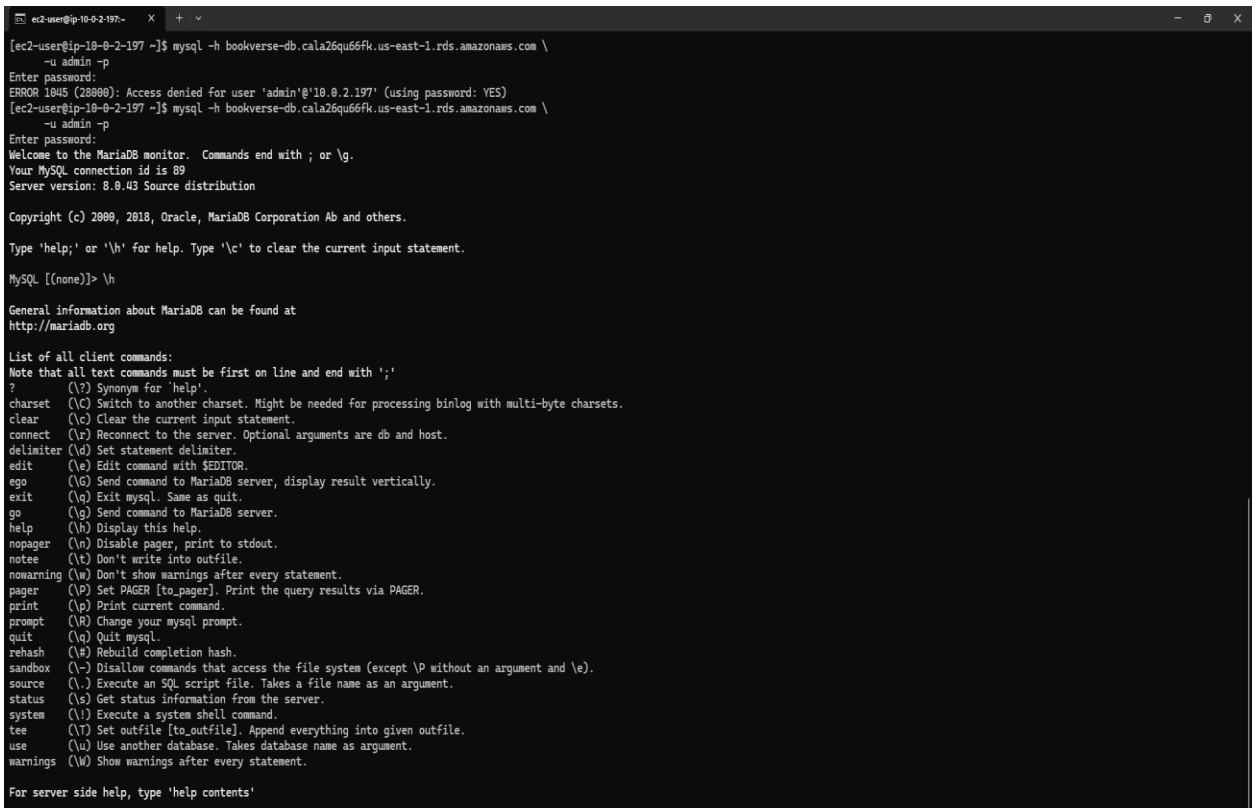
- **Allow bastion → RDS traffic (Console)**
- **SSH into bastion (CLI)**
- **Test DB connectivity** (Bastion)

    1. Installed mysql / mariadb client with sudo dnf install.
    2. Connected to RDS: *mysql -h bookverse-db.cala26qu66fk.us-east-1.rds.amazonaws.com -u admin -p*
    3. Confirmed successful connection to the BookVerse database.





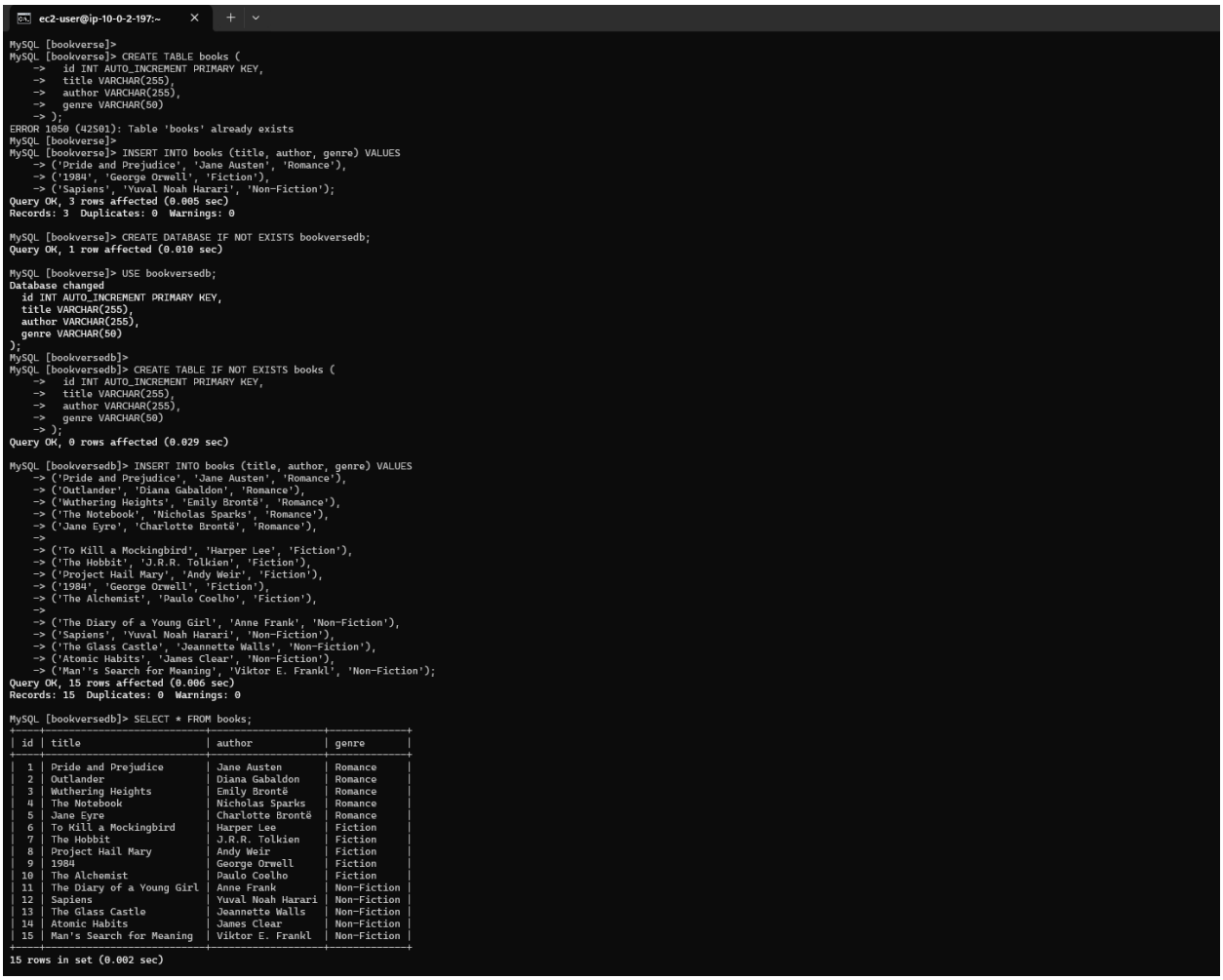
```
[ec2-user@ip-10-0-2-197 ~]$ sudo dnf install -y mariadb105
Last metadata expiration check: 0:04:02 ago on Sat Dec  6 10:58:33 2025.
Dependencies resolved.
================================================================================
 Package                 Architecture  Version                 Repository  Size
================================================================================
Installing:
 mariadb105              x86_64        3:10.5.29-1.amzn2023.0.1  amazonlinux  1.5 M
Installing dependencies:
 mariadb-connector-c     x86_64        3.3.10-1.amzn2023.0.1     amazonlinux  211 k
 mariadb-connector-c-config noarch     3.3.10-1.amzn2023.0.1     amazonlinux  9.9 k
 mariadb105-common       x86_64        3:10.5.29-1.amzn2023.0.1  amazonlinux   28 k
 perl-Sys-Hostname       x86_64        1.23-477.amzn2023.0.7     amazonlinux   16 k

Transaction Summary
================================================================================
Install  5 Packages

Total download size: 1.8 M
Installed size: 19 M
Downloading Packages:
(1/5): mariadb-connector-c-config-3.3.10-1.amzn2023.0.1.noarch.rpm  292 kB/s | 9.9 kB  00:00
(2/5): mariadb-connector-c-3.3.10-1.amzn2023.0.1.x86_64.rpm         5.0 MB/s | 211 kB  00:00
(3/5): mariadb105-10.5.29-1.amzn2023.0.1.x86_64.rpm                  28 MB/s | 1.5 MB  00:00
(4/5): mariadb105-common-10.5.29-1.amzn2023.0.1.x86_64.rpm          1.2 MB/s |  28 kB  00:00
(5/5): perl-Sys-Hostname-1.23-477.amzn2023.0.7.x86_64.rpm           797 kB/s |  16 kB  00:00
--------------------------------------------------------------------------------
Total                                                                18 MB/s | 1.8 MB  00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : mariadb-connector-c-config-3.3.10-1.amzn2023.0.1.noarch 1/5
  Installing       : mariadb-connector-c-3.3.10-1.amzn2023.0.1.x86_64       2/5
  Installing       : mariadb105-common-3:10.5.29-1.amzn2023.0.1.x86_64      3/5
  Installing       : perl-Sys-Hostname-1.23-477.amzn2023.0.7.x86_64         4/5
  Installing       : mariadb105-3:10.5.29-1.amzn2023.0.1.x86_64             5/5
  Running scriptlet: mariadb105-3:10.5.29-1.amzn2023.0.1.x86_64             5/5
  Verifying        : mariadb-connector-c-3.3.10-1.amzn2023.0.1.x86_64       1/5
  Verifying        : mariadb-connector-c-config-3.3.10-1.amzn2023.0.1.noarch 2/5
  Verifying        : mariadb105-3:10.5.29-1.amzn2023.0.1.x86_64             3/5
  Verifying        : mariadb105-common-3:10.5.29-1.amzn2023.0.1.x86_64      4/5
  Verifying        : perl-Sys-Hostname-1.23-477.amzn2023.0.7.x86_64         5/5

Installed:
  mariadb-connector-c-3.3.10-1.amzn2023.0.1.x86_64
  mariadb-connector-c-config-3.3.10-1.amzn2023.0.1.noarch
  mariadb105-3:10.5.29-1.amzn2023.0.1.x86_64
  mariadb105-common-3:10.5.29-1.amzn2023.0.1.x86_64
  perl-Sys-Hostname-1.23-477.amzn2023.0.7.x86_64

Complete!
[ec2-user@ip-10-0-2-197 ~]$ mysql --version
mysql  Ver 15.1 Distrib 10.5.29-MariaDB, for Linux (x86_64) using  EditLine wrapper
[ec2-user@ip-10-0-2-197 ~]$ mysql -h bookverse-db.cala26qu66fk.us-east-1.rds.amazonaws.com \
      -u admin -p
Enter password:
ERROR 1045 (28000): Access denied for user 'admin'@'10.0.2.197' (using password: YES)
```



```
[ec2-user@ip-10-0-2-197 ~]$ mysql -h bookverse-db.cala26qu66fk.us-east-1.rds.amazonaws.com \
      -u admin -p
Enter password:
ERROR 1045 (28000): Access denied for user 'admin'@'10.0.2.197' (using password: YES)
[ec2-user@ip-10-0-2-197 ~]$ mysql -h bookverse-db.cala26qu66fk.us-east-1.rds.amazonaws.com \
      -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 89
Server version: 8.0.43 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> \h

General information about MariaDB can be found at
http://mariadb.org

List of all client commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
charset   (\C) Switch to another charset. Might be needed for processing binlog with multi-byte charsets.
clear     (\c) Clear the current input statement.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter.
edit      (\e) Edit command with $EDITOR.
ego       (\G) Send command to MariaDB server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to MariaDB server.
help      (\h) Display this help.
nopager   (\n) Disable pager, print to stdout.
notee     (\t) Don't write into outfile.
nowarning (\w) Don't show warnings after every statement.
pager     (\P) Set PAGER [to_pager]. Print the query results via PAGER.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
sandbox   (\-) Disallow commands that access the file system (except \P without an argument and \e).
source    (\.) Execute an SQL script file. Takes a file name as an argument.
status    (\s) Get status information from the server.
system    (\!) Execute a system shell command.
tee       (\T) Set outfile [to_outfile]. Append everything into given outfile.
use       (\u) Use another database. Takes database name as argument.
warnings  (\W) Show warnings after every statement.

For server side help, type 'help contents'
```

5.5 Database & App Data Validation (Bastion + Python)

- **Install Python + dependencies (Bastion)**
    1. *sudo dnf install -y python3-pip*
    2. *pip3 install --user mysql-connector-python boto3 requests.*
- **Run DB check script (db_check.py)**
    a. Script connects to RDS, queries the books table, and prints sample rows:
        i. Romance: Pride and Prejudice, Outlander, etc.
        ii. Fiction: To Kill a Mockingbird, 1984, etc.
    b. Output showed
    c. This proves the RDS backend is configured and populated with BookVerse data.

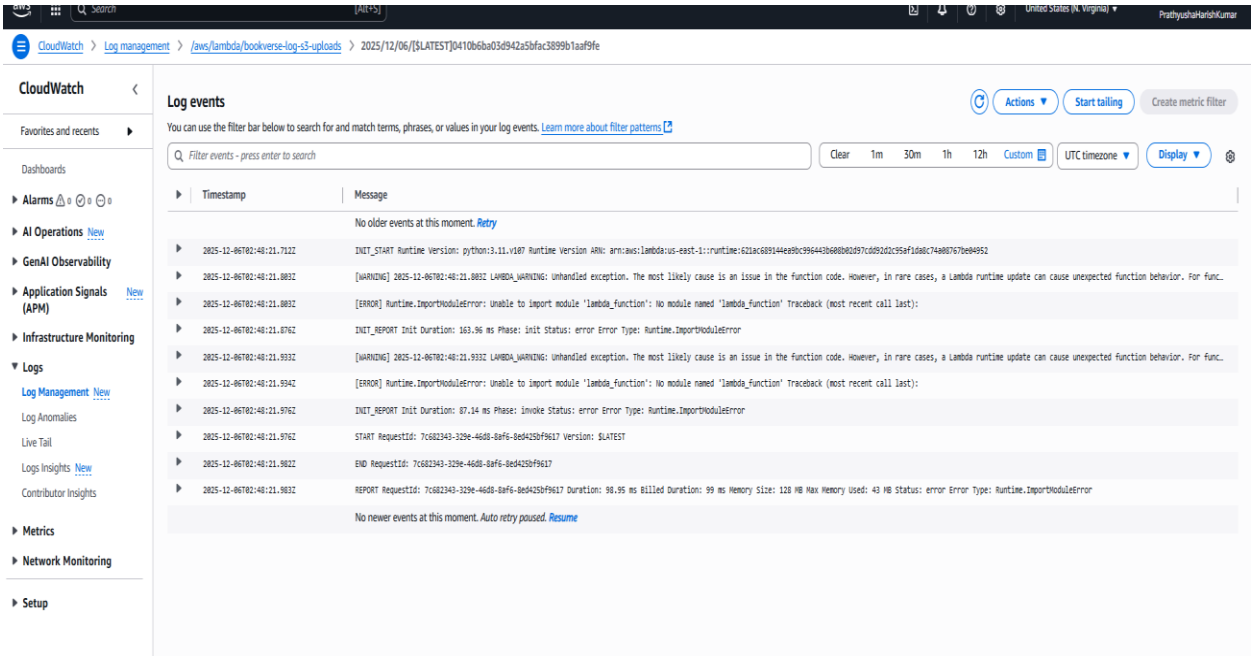


```
ec2-user@ip-10-0-2-197:~

MySQL [bookverse]>
MySQL [bookverse]> CREATE TABLE books (
    ->   id INT AUTO_INCREMENT PRIMARY KEY,
    ->   title VARCHAR(255),
    ->   author VARCHAR(255),
    ->   genre VARCHAR(50)
    -> );
ERROR 1050 (42S01): Table 'books' already exists
MySQL [bookverse]>
MySQL [bookverse]> INSERT INTO books (title, author, genre) VALUES
    -> ('Pride and Prejudice', 'Jane Austen', 'Romance'),
    -> ('1984', 'George Orwell', 'Fiction'),
    -> ('Sapiens', 'Yuval Noah Harari', 'Non-Fiction');
Query OK, 3 rows affected (0.005 sec)
Records: 3  Duplicates: 0  Warnings: 0

MySQL [bookverse]> CREATE DATABASE IF NOT EXISTS bookversedb;
Query OK, 1 row affected (0.010 sec)

MySQL [bookverse]> USE bookversedb;
Database changed
  id INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(255),
  author VARCHAR(255),
  genre VARCHAR(50)
);
MySQL [bookversedb]>
MySQL [bookversedb]> CREATE TABLE IF NOT EXISTS books (
    ->   id INT AUTO_INCREMENT PRIMARY KEY,
    ->   title VARCHAR(255),
    ->   author VARCHAR(255),
    ->   genre VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.029 sec)

MySQL [bookversedb]> INSERT INTO books (title, author, genre) VALUES
    -> ('Pride and Prejudice', 'Jane Austen', 'Romance'),
    -> ('Outlander', 'Diana Gabaldon', 'Romance'),
    -> ('Wuthering Heights', 'Emily Brontë', 'Romance'),
    -> ('The Notebook', 'Nicholas Sparks', 'Romance'),
    -> ('Jane Eyre', 'Charlotte Brontë', 'Romance'),
    ->
    -> ('To Kill a Mockingbird', 'Harper Lee', 'Fiction'),
    -> ('The Hobbit', 'J.R.R. Tolkien', 'Fiction'),
    -> ('Project Hail Mary', 'Andy Weir', 'Fiction'),
    -> ('1984', 'George Orwell', 'Fiction'),
    -> ('The Alchemist', 'Paulo Coelho', 'Fiction'),
    ->
    -> ('The Diary of a Young Girl', 'Anne Frank', 'Non-Fiction'),
    -> ('Sapiens', 'Yuval Noah Harari', 'Non-Fiction'),
    -> ('The Glass Castle', 'Jeannette Walls', 'Non-Fiction'),
    -> ('Atomic Habits', 'James Clear', 'Non-Fiction'),
    -> ('Man''s Search for Meaning', 'Viktor E. Frankl', 'Non-Fiction');
Query OK, 15 rows affected (0.006 sec)
Records: 15  Duplicates: 0  Warnings: 0

MySQL [bookversedb]> SELECT * FROM books;
+----+---------------------------+------------------+-------------+
| id | title                     | author           | genre       |
+----+---------------------------+------------------+-------------+
|  1 | Pride and Prejudice       | Jane Austen      | Romance     |
|  2 | Outlander                 | Diana Gabaldon   | Romance     |
|  3 | Wuthering Heights         | Emily Brontë     | Romance     |
|  4 | The Notebook              | Nicholas Sparks  | Romance     |
|  5 | Jane Eyre                 | Charlotte Brontë | Romance     |
|  6 | To Kill a Mockingbird     | Harper Lee       | Fiction     |
|  7 | The Hobbit                | J.R.R. Tolkien   | Fiction     |
|  8 | Project Hail Mary         | Andy Weir        | Fiction     |
|  9 | 1984                      | George Orwell    | Fiction     |
| 10 | The Alchemist             | Paulo Coelho     | Fiction     |
| 11 | The Diary of a Young Girl | Anne Frank       | Non-Fiction |
| 12 | Sapiens                   | Yuval Noah Harari| Non-Fiction |
| 13 | The Glass Castle          | Jeannette Walls  | Non-Fiction |
| 14 | Atomic Habits             | James Clear      | Non-Fiction |
| 15 | Man's Search for Meaning  | Viktor E. Frankl | Non-Fiction |
+----+---------------------------+------------------+-------------+
15 rows in set (0.002 sec)
```

5.6 AWS Lambda and S3 Logging (CloudFormation + Console + Boto3)

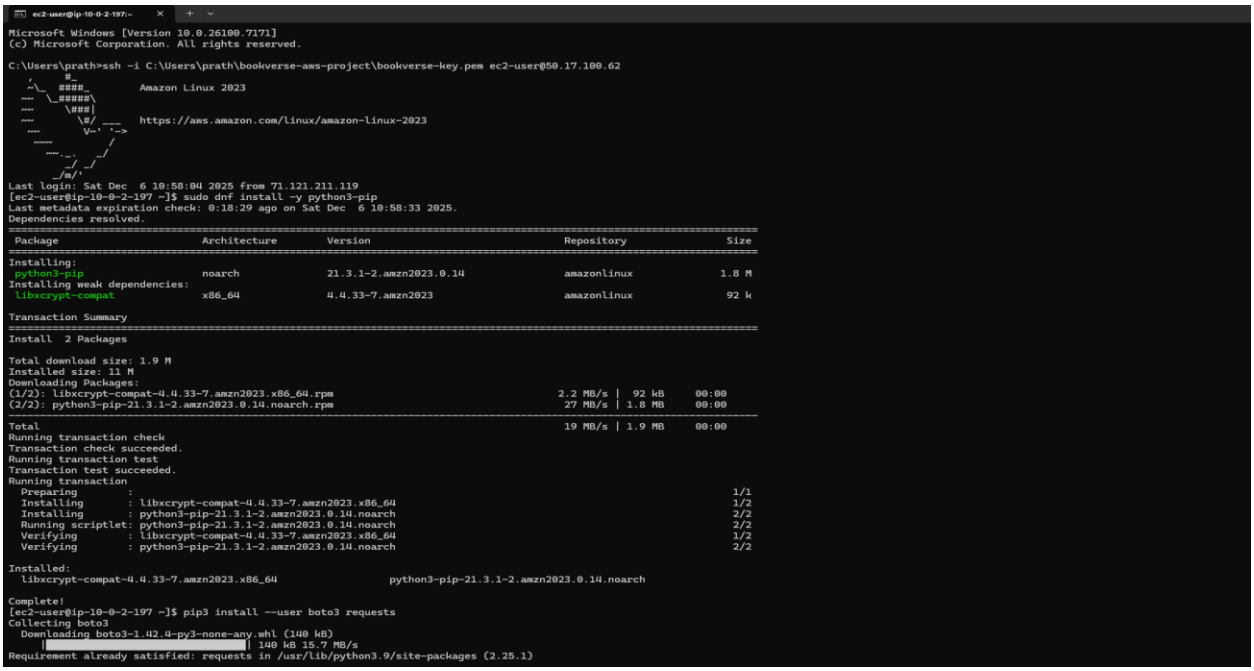
- **Lambda function for S3 uploads** (CloudFormation)
    1. *bookverse-log-s3-uploads* created via CloudFormation with inline Python.
    2. S3 bucket trigger configured via template.
- **Test Lambda manually** (Boto3)
    1. Used *invoke_lambda.py* from the boto3 folder:
        - Boto3 *lambda_client.invoke()* on *bookverse-log-s3-uploads*.
        - Saw status code 200 and payload in terminal.
- **Verify S3 upload event** (Console)
    1. Uploaded test file *boto3-test-file.txt* to S3 with Boto3 script *create_bucket_and_upload.py*.
    2. Checked **CloudWatch Logs** for the Lambda group to confirm the S3 event details were logged.

```
C:\Users\prath\bookverse-aws-project\boto3>python create_bucket_and_upload.py
Creating bucket: bookverse-boto3-0fc94320
Uploading boto3-test-file.txt to s3://bookverse-boto3-0fc94320/
✅ Done.
✅ Bucket: bookverse-boto3-0fc94320
✅ Object: boto3-test-file.txt

C:\Users\prath\bookverse-aws-project\boto3>cd C:\Users\prath\bookverse-aws-project\boto3

C:\Users\prath\bookverse-aws-project\boto3>python list_running_ec2.py
Running EC2 instances in us-east-1:
- i-00c0e3ae4aa049b21 | t3.micro | running | 10.0.3.88

C:\Users\prath\bookverse-aws-project\boto3>python invoke_lambda.py
Invoking Lambda: bookverse-log-s3-uploads
StatusCode: 200
Response payload: {"statusCode": 200, "body": "logged"}
```



5.7 Boto3 Interaction Scripts (Local + Bastion)

- **Upload file to S3 Bucket**
    1. From boto3 directory on local machine:
        - Created a uniquely named S3 bucket (e.g., *bookverse-boto3-2990bcfc*).
        - Uploaded *boto3-test-file.txt*.
- **List running EC2 instances** *(list_running_ec2.py)*
    1. Listed all running instances in us-east-1 with instance ID, type, and private IP.
- **Invoke Lambda manually** *(invoke_lambda.py)*
    1. Called bookverse-log-s3-uploads with a small JSON event.
    2. Printed response StatusCode and Payload.
- **Get instance metadata from bastion** *(get_instance_metadata.py)*
    1. On bastion, called **Instance Metadata Service (IMDSv2)** via requests:
        - Retrieved instance ID, type, local IP, and AZ.
    2. Also used Boto3 *ec2.describe_instances()* to match metadata through AWS APIs.



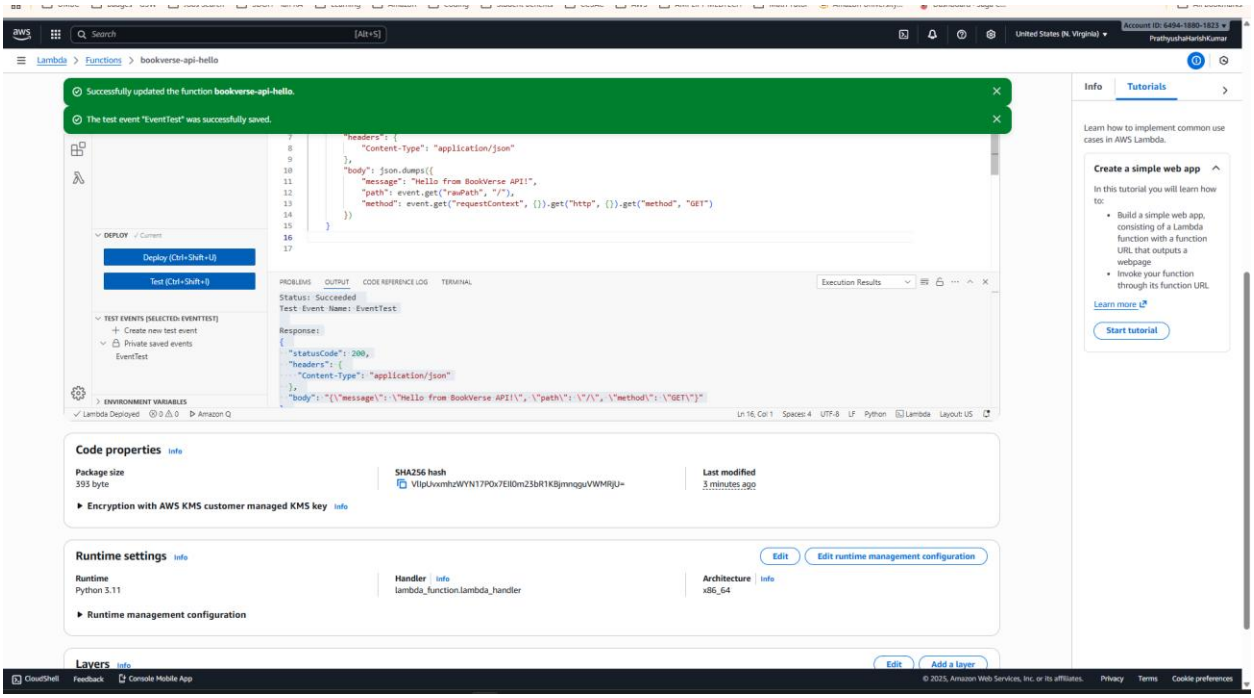
BONUS STEPS PERFORMED

5.8. API Gateway + Lambda (Console + CLI)

- **Create Lambda for API** (*api_hello_lambda.py*)
    1. Simple function returning JSON:
        - *"message": "Hello from BookVerse API!"*
- **Create HTTP API in API Gateway** (Console)
    1. Created *BookverseAPI* with:
        - Route: *GET /hello*
        - Integration: the above Lambda.
- **Deploy API stage** (Console)
    1. Created stage *prod.*
    2. Noted invoke URL:

       https://<api-id>.execute-api.us-east-1.amazonaws.com/prod/hello.
- **Test API from CLI**
    1. From Windows CMD:

curl "https://<api-id>.execute-api.us-east-1.amazonaws.com/prod/hello".



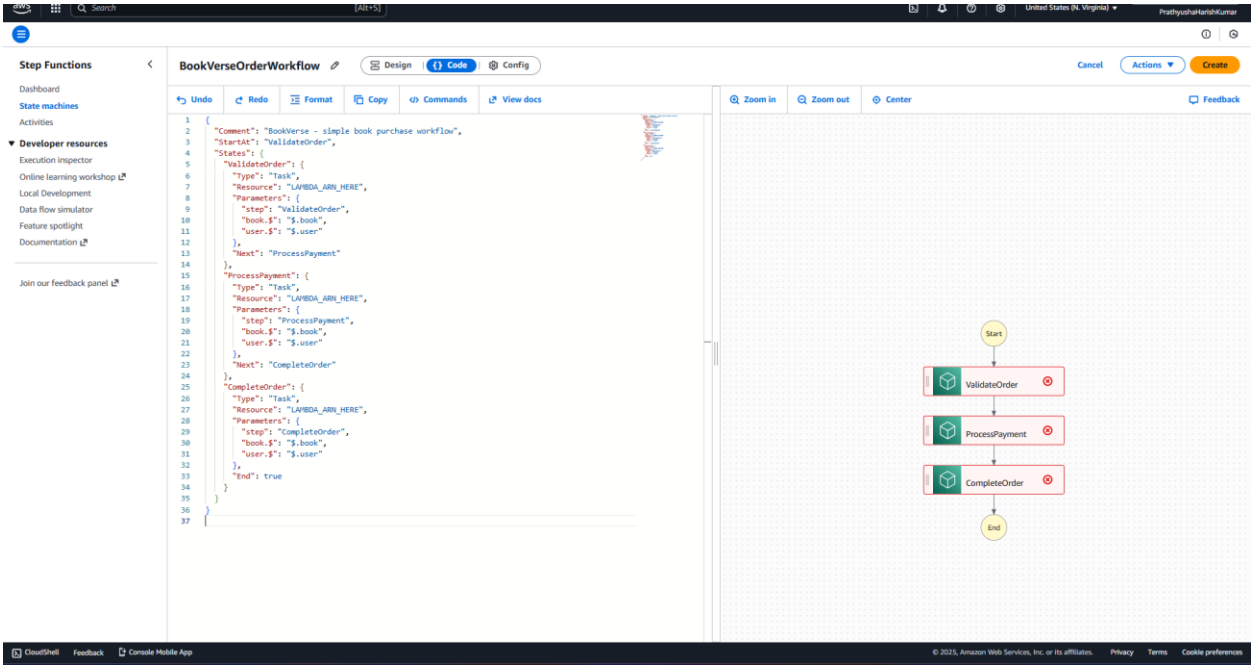
5.9 Step Functions Workflow (Console)

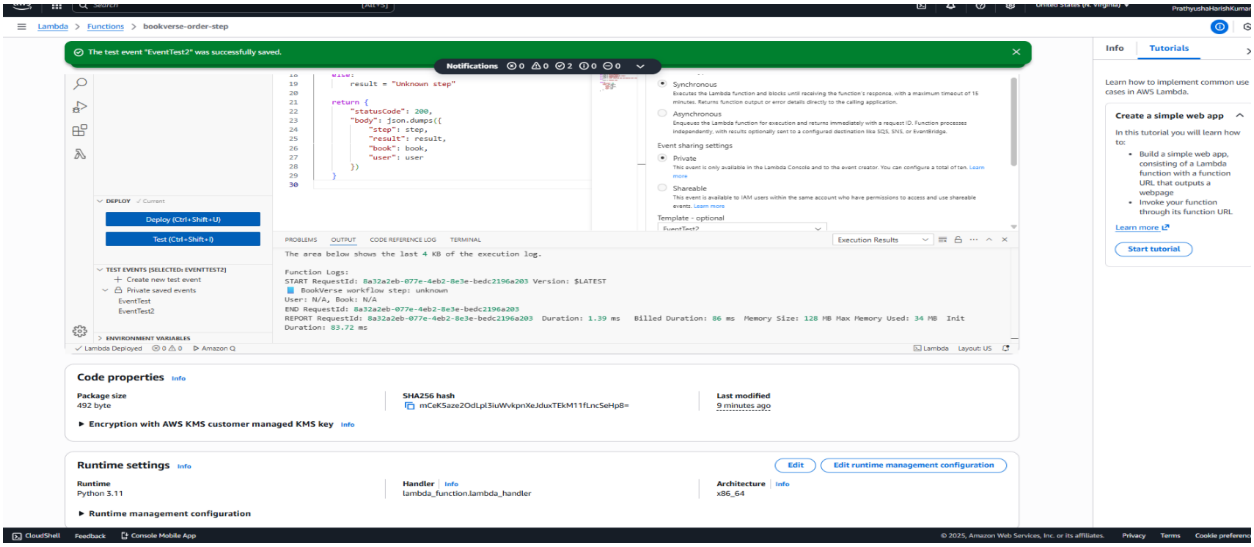
- **Define state machine** (bookverse-order-workflow)
    1. Three states in JSON:
        - *ValidateOrder* → Lambda
        - *ProcessPayment* → Lambda
        - *CompleteOrder* → Lambda
    2. Used the **Amazon States Language** in the Step Functions console.
- **Fix JSONPath / input mapping**

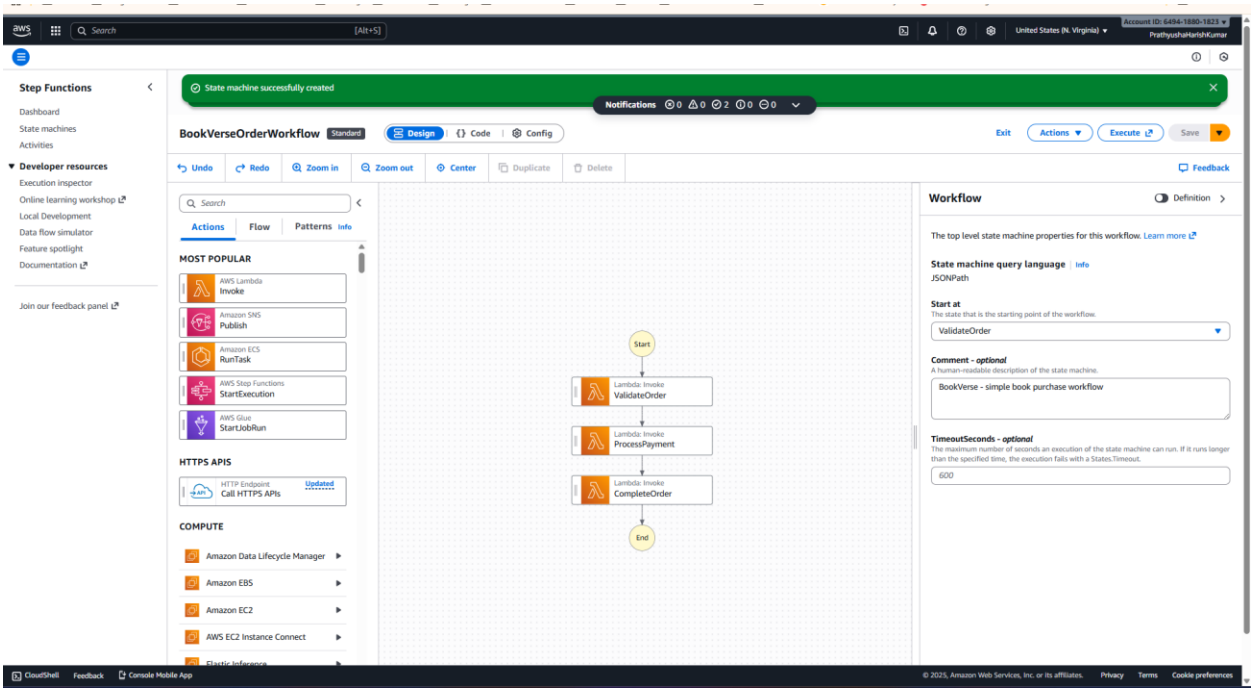
1. Adjusted *ResultPath* and *Parameters* so that:
    - The output of one Lambda became the input for the next.

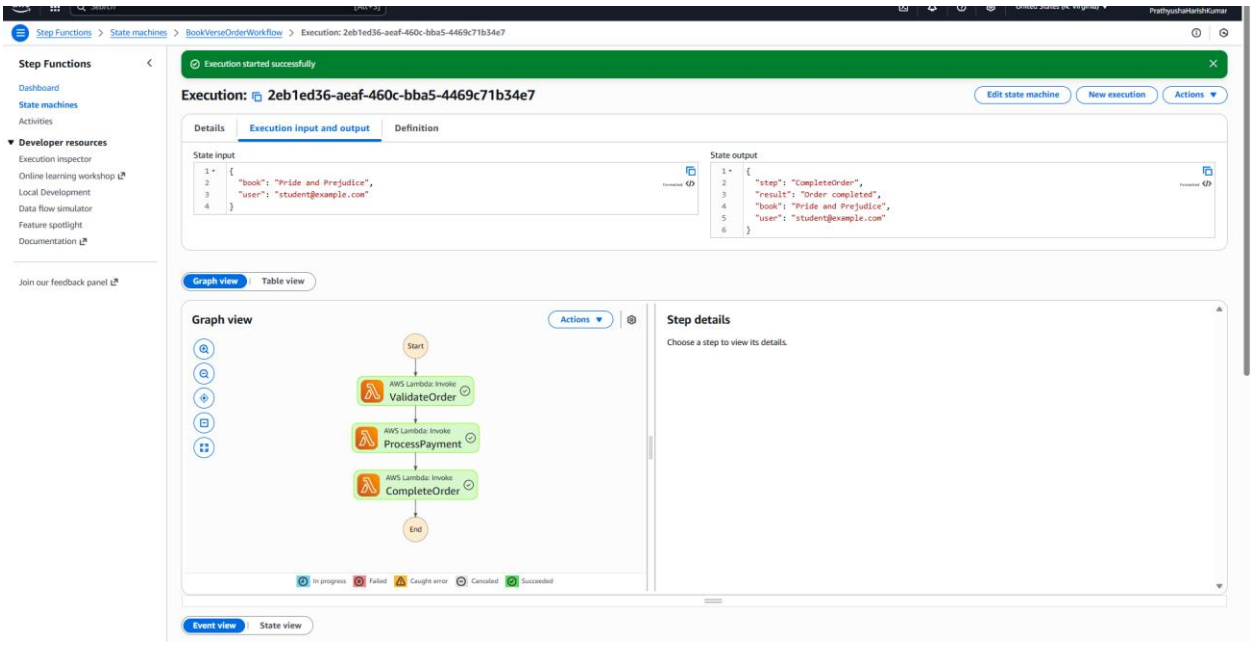
- **Run successful execution** (Console)
    1. Started execution with input:

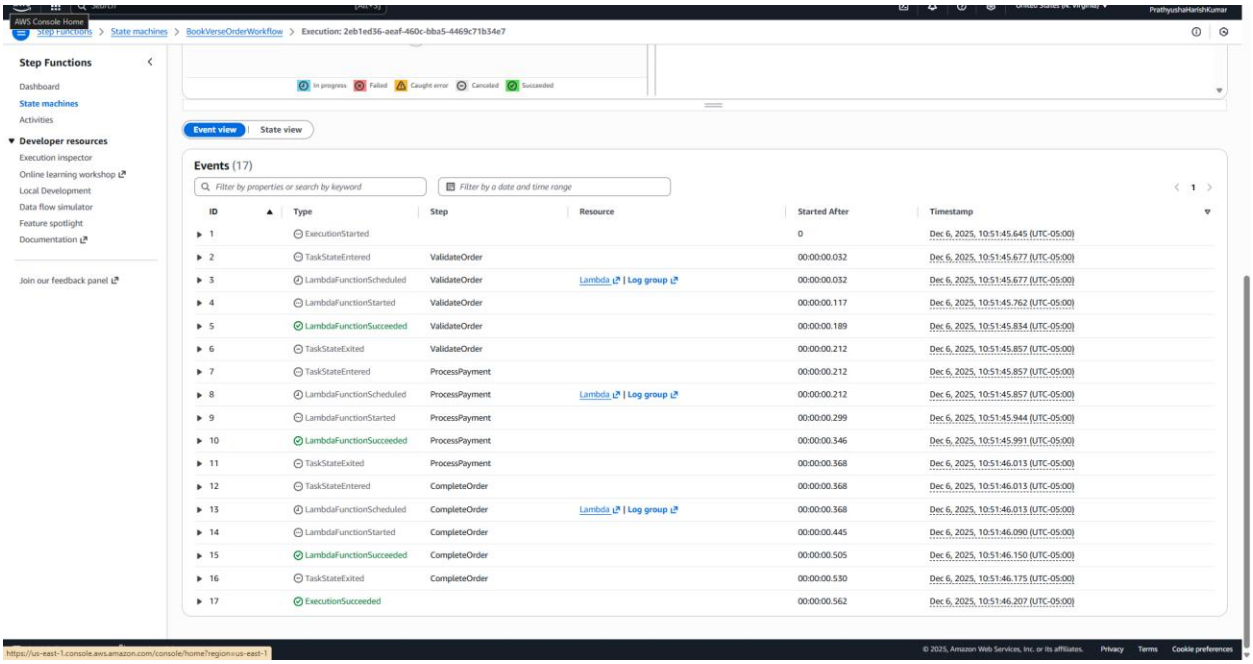
       { "book": "Pride and Prejudice", "user": "student@example.com" }.
    2. Execution status: **SUCCEEDED**.
    3. Final output:
        - *"step": "CompleteOrder", "result": "Order completed".*






Step Functions
State machine successfully created
BookVerseOrderWorkflow
Standard
Design
Code
Config
Exit
Actions
Execute
Save
Undo
Redo
Zoom in
Zoom out
Center
Duplicate
Delete
Feedback
Search
Actions
Flow
Patterns
MOST POPULAR
AWS Lambda Invoke
Amazon SNS Publish
Amazon ECS RunTask
AWS Step Functions StartExecution
AWS Glue StartJobRun
HTTPS APIS
HTTP Endpoint Call HTTPS APIs
COMPUTE
Amazon Data Lifecycle Manager
Amazon EBS
Amazon EC2
AWS EC2 Instance Connect
Start
Lambda: Invoke ValidateOrder
Lambda: Invoke ProcessPayment
Lambda: Invoke CompleteOrder
End
Workflow
Definition
The top level state machine properties for this workflow. Learn more
State machine query language
JSONPath
Start at
The state that is the starting point of the workflow.
ValidateOrder
Comment - optional
A human-readable description of the state machine.
BookVerse - simple book purchase workflow
TimeoutSeconds - optional
The maximum number of seconds an execution of the state machine can run. If it runs longer than the specified time, the execution fails with a States.Timeout.
600
CloudShell
Feedback
Console Mobile App
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Step Functions > State machines > BookVerseOrderWorkflow > Execution: 2eb1ed36-aeaf-460c-bba5-4469c71b34e7
Execution started successfully
Execution: 2eb1ed36-aeaf-460c-bba5-4469c71b34e7
Edit state machine
New execution
Actions
Details
Execution input and output
Definition
State input
{
"book": "Pride and Prejudice",
"user": "student@example.com"
}
State output
{
"step": "CompleteOrder",
"result": "Order completed",
"book": "Pride and Prejudice",
"user": "student@example.com"
}
Graph view
Table view
Graph view
Actions
Start
AWS Lambda: Invoke ValidateOrder
AWS Lambda: Invoke ProcessPayment
AWS Lambda: Invoke CompleteOrder
End
In progress
Failed
Caught error
Canceled
Succeeded
Step details
Choose a step to view its details.
Event view
State view



6. CHALLENGES ENCOUNTERED AND SOLUTIONS APPLIED

6.1 Coordinating Terraform and CloudFormation

Challenge:

One of the biggest challenges I faced was coordinating Terraform and CloudFormation together. Terraform was responsible for creating networking components such as the VPC, subnets, and security groups, while CloudFormation depended on those resources to deploy EC2, RDS, Lambda, and the ALB. Initially, the CloudFormation stack failed multiple times because the VPC or subnet IDs I referenced no longer existed or had been recreated.

Solution:

To resolve this, I relied entirely on Terraform outputs. After Terraform finished provisioning, I captured the VPC ID, subnet IDs, and security group IDs from the outputs and passed them explicitly as parameters into the CloudFormation template. I ensured Terraform always ran first and verified its outputs before deploying CloudFormation, which eliminated consistency issues.

6.2 Auto Scaling Replacing EC2 Instances

Challenge:

While configuring the EC2 instances, I noticed that any manual changes I made were lost after some time. This happened because the Auto Scaling Group automatically replaced instances that failed health checks, wiping out my configurations.

Solution:

Instead of relying on manual changes, I reduced configuration inside running instances and focused on validation rather than persistence. This experience helped me understand why production systems rely on automation (such as user data scripts or custom AMIs) rather than manual setup. I documented this as a future improvement for the project.

6.3 Connecting to RDS in Private Subnets

Challenge:

Even though the RDS instance was deployed successfully, I couldn't connect to it at first because it lived inside private subnets with no public access. This made troubleshooting difficult since the database had no external endpoint.

Solution:

I deployed a bastion host in a public subnet and adjusted security group rules so that MySQL traffic (port 3306) was allowed only from the bastion's security group. After connecting through the bastion and successfully accessing the database, I was confident that the networking and security design was correct.

6.4 Lambda Handler and Event Mapping Issues

Challenge:

I initially encountered Lambda execution failures due to incorrect handler definitions and mismatched event payloads, particularly when invoking the function manually using Boto3. These resulted in ImportModuleError and runtime failures.

Solution:

I carefully aligned the handler names in the CloudFormation template with the actual Python files and standardized the event payload format. By monitoring CloudWatch Logs, I verified that the Lambda function was executing correctly, and logging S3 upload events as expected.

6.5 Step Functions JSON Path Errors

Challenge:

While implementing Step Functions, executions failed because data was not being passed correctly between states. Errors like $.book not found indicated incorrect JSONPath usage between Lambda outputs.

Solution:

I fixed this by restructuring the state machine definition, using proper ResultPath mappings and ensuring each Lambda returned consistent JSON output. After these changes, the full workflow executed successfully from order validation to payment processing and order completion.

6.6 S3 Static Website Access Problems

Challenge:

When hosting the static website on S3, I initially received 404 errors even though the bucket existed. This was caused by missing index files and blocked public access settings.

Solution:

I uploaded the correct index.html file, enabled static website hosting, and carefully reviewed the bucket configuration. Once these were corrected, the website was accessible using the S3 website endpoint.

7. FUTURE IMPROVEMENTS

Despite the fact that this project has accomplished all the functional and architectural requirements, there are still several places where it could be enhanced to be a production-grade system.

1. Full Automation of EC2 Configuration

Manual execution of some EC2 configuration and validation steps took place for learning and debugging purposes. The complete automation of this process is expected in the production environment to be done with user data scripts, configuration management tools, or custom AMIs. As a result, any instance created by Auto Scaling would be instantly production-ready and would not require human intervention.

2. Deeper Application-Level Database Integration

Database connectivity was successfully validated and real data was queried from Amazon RDS, however, the web application could have been improved to communicate with the database layer directly. Future versions could retrieve book data, user selections, or order details from RDS on the fly rather than depending on static content, thereby making the whole application data-driven.

3. HTTPS and Security Enhancements

The application is currently only reachable through HTTP. The production version would be able to utilize AWS Certificate Manager (ACM) for HTTPS enabling and SSL certificate attachment to the Application Load Balancer as a part of the deployment. Moreover, AWS WAF could be included to mitigate risks related to common web attacks, therefore, securing the application further.

4. CI/CD Pipeline Implementation

Even with the appropriate use of Infrastructure-as-Code, modifying the application remained a manual task. The establishment of a CI/CD pipeline through GitHub Actions and AWS CodeDeploy could be a way of automating the whole process of keeping test and deployment with every commit.

5. Monitoring and Alerting Improvements

The basics of monitoring are provided by CloudWatch logs; however, an upgrade in the CloudWatch dashboards would be the inclusion of alarming for EC2 and RDS performance metrics, and the use of Amazon SNS for alerting the administrators about issues proactively would be the alerting method.

6. High Availability and Disaster Recovery

The database tier might be upgraded to a Multi-AZ RDS deployment and the application of AWS Backup could make backups automatic thus giving the system more durable and robust. This would result in a significant increase in fault tolerance and data durability.

8. CONCLUSION

The scalable AWS based cloud architecture using Infrastructure as Code (IaC) has been successfully demonstrated in this project through its design, deployment, and validation.

The BookVerse application has been used as a perfect example to illustrate the concepts of load balancing, auto scaling, secure database access, and event-driven serverless processing. By interacting with the AWS Management Console, AWS CLI, and Python Boto3 scripts the project has not only confirmed resource creation but also has been successful in checking operational connectivity and behavior across services.

Among the key learning outcomes there are those linked to multi-tier architecture design, network isolation, and security groups being used for security enforcement, distributed system troubleshooting, and AWS Step Functions being used for workflow orchestration.