

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**
- 2. PROJECT OVERVIEW**
- 3. ARCHITECTURE OVERVIEW**
- 4. TERRAFORM AND CLOUDFORMATION SCRIPTS**
- 5. STEP-BY-STEP IMPLEMENTATION PROCESS (CONSOLE, CLI, BOTO3)**
- 6. CHALLENGES AND SOLUTION**
- 7. FUTURE IMPROVEMENTS**
- 8. CONCLUSION**

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.

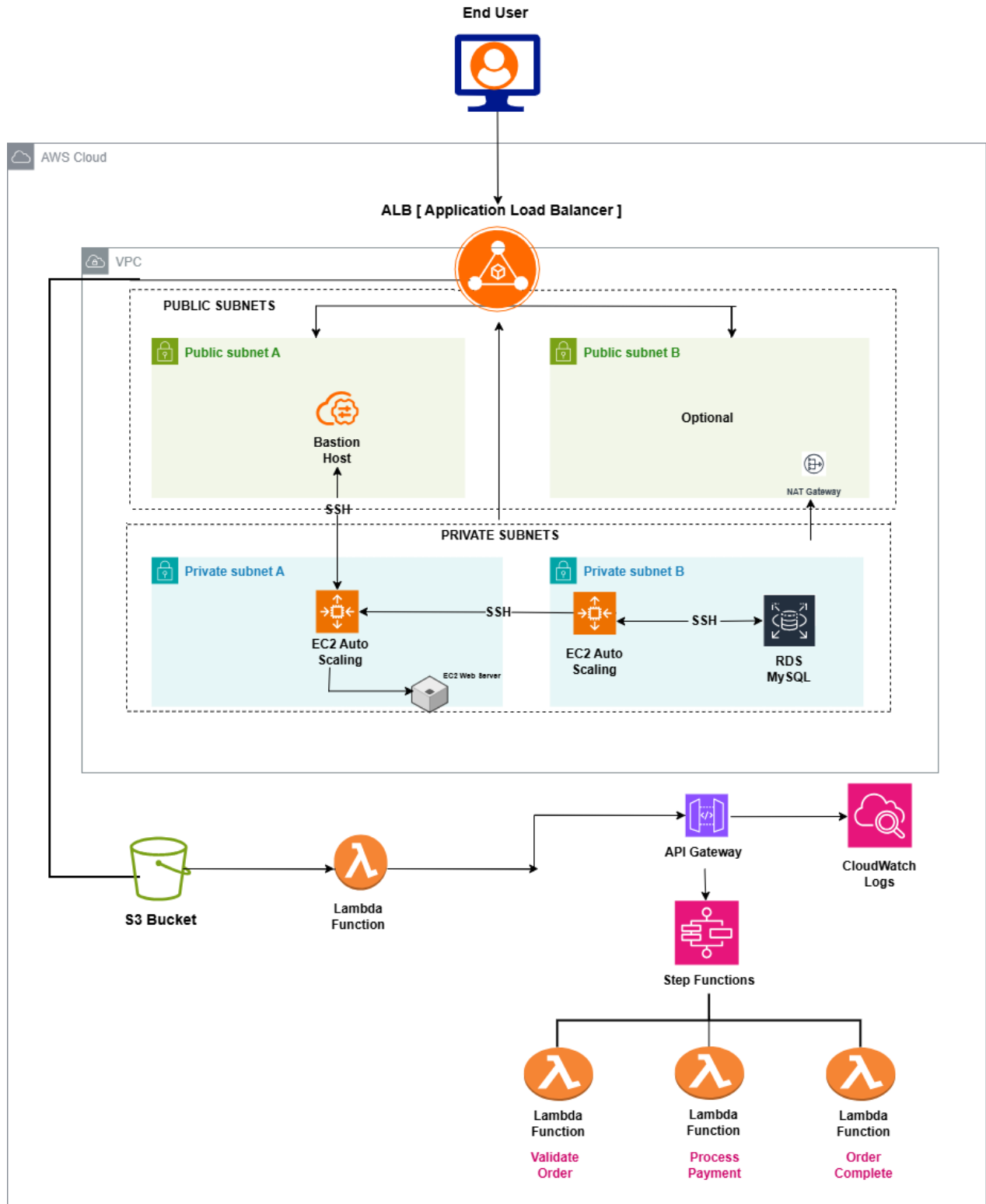


Fig : Architecture Diagram

4. TERRAFORM AND CLOUDFORMATION SCRIPTS

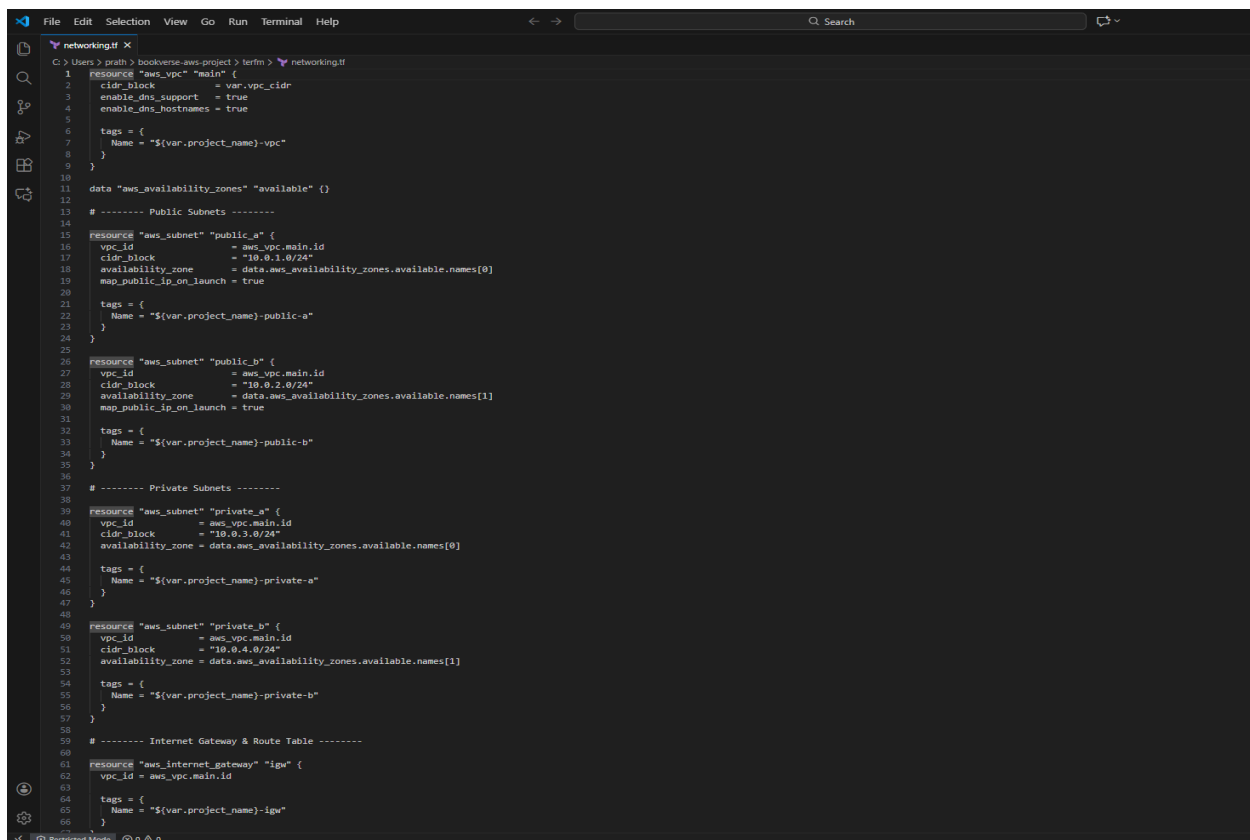
4.1 Terraform – Networking Infrastructure

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.



```
1 resource "aws_vpc" "main" {
2   cidr_block = var.vpc_cidr
3   enable_dns_support = true
4   enable_dns_hostnames = true
5
6   tags = {
7     Name = "${var.project_name}-vpc"
8   }
9 }
10
11 data "aws_availability_zones" "available" {}
12
13 # ----- Public Subnets -----
14
15 resource "aws_subnet" "public_a" {
16   vpc_id = aws_vpc.main.id
17   cidr_block = "10.0.1.0/24"
18   availability_zone = data.aws_availability_zones.available.names[0]
19   map_public_ip_on_launch = true
20
21   tags = {
22     Name = "${var.project_name}-public-a"
23   }
24 }
25
26 resource "aws_subnet" "public_b" {
27   vpc_id = aws_vpc.main.id
28   cidr_block = "10.0.2.0/24"
29   availability_zone = data.aws_availability_zones.available.names[1]
30   map_public_ip_on_launch = true
31
32   tags = {
33     Name = "${var.project_name}-public-b"
34   }
35 }
36
37 # ----- Private Subnets -----
38
39 resource "aws_subnet" "private_a" {
40   vpc_id = aws_vpc.main.id
41   cidr_block = "10.0.3.0/24"
42   availability_zone = data.aws_availability_zones.available.names[0]
43
44   tags = {
45     Name = "${var.project_name}-private-a"
46   }
47 }
48
49 resource "aws_subnet" "private_b" {
50   vpc_id = aws_vpc.main.id
51   cidr_block = "10.0.4.0/24"
52   availability_zone = data.aws_availability_zones.available.names[1]
53
54   tags = {
55     Name = "${var.project_name}-private-b"
56   }
57 }
58
59 # ----- Internet Gateway & Route Table -----
60
61 resource "aws_internet_gateway" "igw" {
62   vpc_id = aws_vpc.main.id
63
64   tags = {
65     Name = "${var.project_name}-igw"
66   }
67 }
```

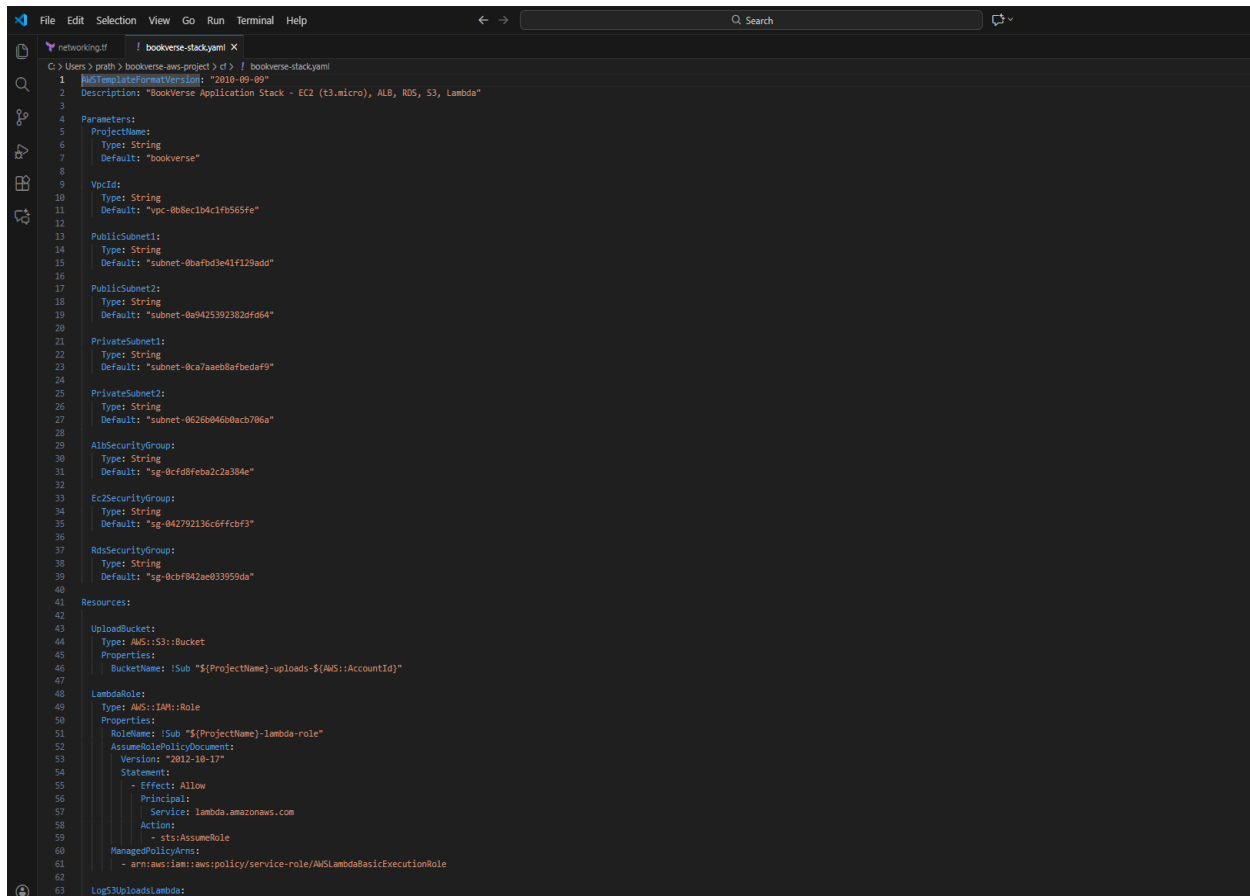
4.2 CloudFormation – Application and Service Deployment

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.



```
File Edit Selection View Go Run Terminal Help
networking / bookverse-stack.yml X
C:\Users\j\path> bookverse-aws-project> cd / bookverse-stack.yml
1 AWSTemplateFormatVersion: "2010-09-09"
2 Description: "BookVerse Application Stack - EC2 (t3.micro), ALB, RDS, S3, Lambda"
3
4 Parameters:
5   ProjectName:
6     Type: String
7     Default: "bookverse"
8
9   VpcId:
10    Type: String
11    Default: "vpc-0b8ec1b4c1fb565fe"
12
13   PublicSubnet1:
14     Type: String
15     Default: "subnet-0ba9bd3e41f129add"
16
17   PublicSubnet2:
18     Type: String
19     Default: "subnet-0a9425392382d6d4"
20
21   PrivateSubnet1:
22     Type: String
23     Default: "subnet-0ca7aeb8afbedaf9"
24
25   PrivateSubnet2:
26     Type: String
27     Default: "subnet-0626b0460bac706a"
28
29   AlbSecurityGroup:
30     Type: String
31     Default: "sg-0cfd8feba2ca384e"
32
33   Ec2SecurityGroup:
34     Type: String
35     Default: "sg-04279213c6ffcbf3"
36
37   RdsSecurityGroup:
38     Type: String
39     Default: "sg-0cbf842ae033959da"
40
41 Resources:
42
43   UploadBucket:
44     Type: AWS::S3::Bucket
45     Properties:
46       BucketName: !Sub "${ProjectName}-uploads-${AWS::AccountId}"
47
48   LambdaRole:
49     Type: AWS::IAM::Role
50     Properties:
51       RoleName: !Sub "${ProjectName}-lambda-role"
52       AssumeRolePolicyDocument:
53         Version: "2012-10-17"
54         Statement:
55           - Effect: Allow
56             Principal:
57               Service: lambda.amazonaws.com
58             Action:
59               - sts:AssumeRole
60       ManagedPolicyArns:
61         - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
62
63   LogS3UploadsLambda:
```

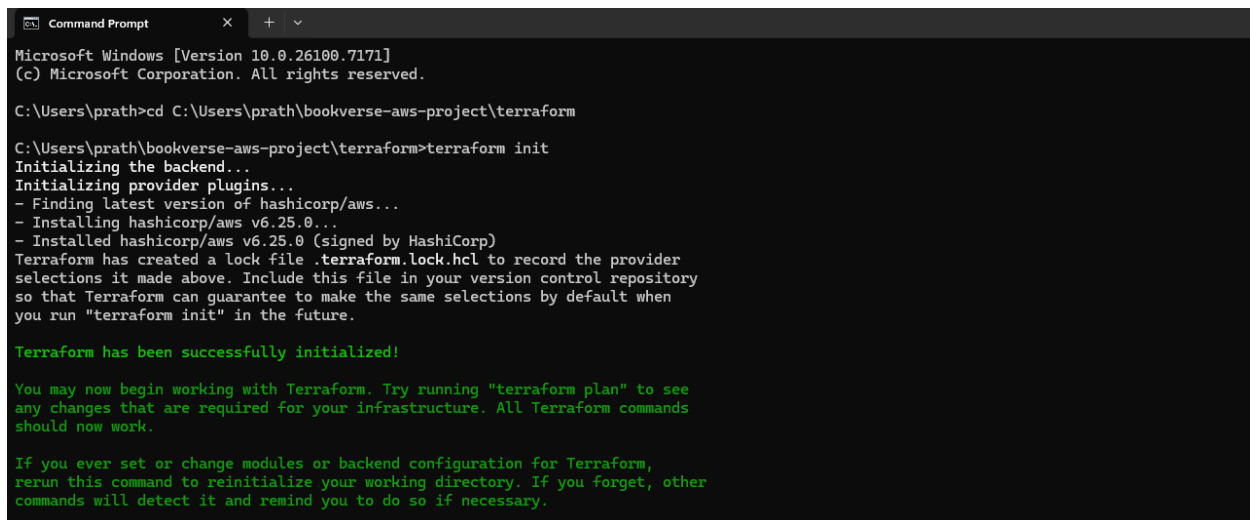
Script Files and Logs

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.



```
Microsoft Windows [Version 10.0.26100.7171]
(c) Microsoft Corporation. All rights reserved.

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

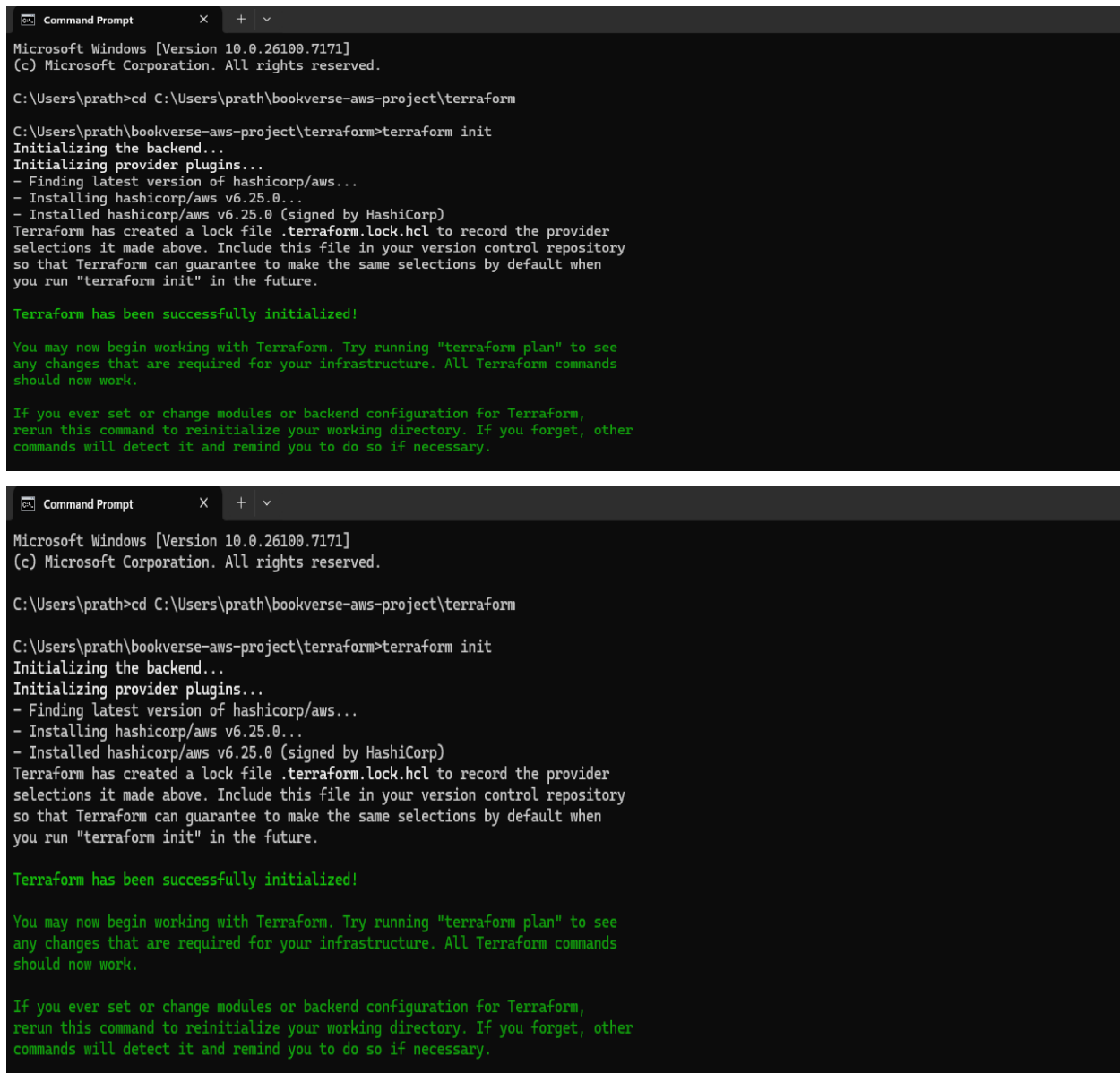
C:\Users\prath\bookverse-aws-project\terraform>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v6.25.0...
- Installed hashicorp/aws v6.25.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

- **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.



The image contains two identical screenshots of a Windows Command Prompt window. The window title is "Command Prompt". The text inside shows the following sequence of commands and output:

```
Microsoft Windows [Version 10.0.26100.7171]
(c) Microsoft Corporation. All rights reserved.

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

C:\Users\prath\bookverse-aws-project\terraform>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v6.25.0...
- Installed hashicorp/aws v6.25.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

- **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:

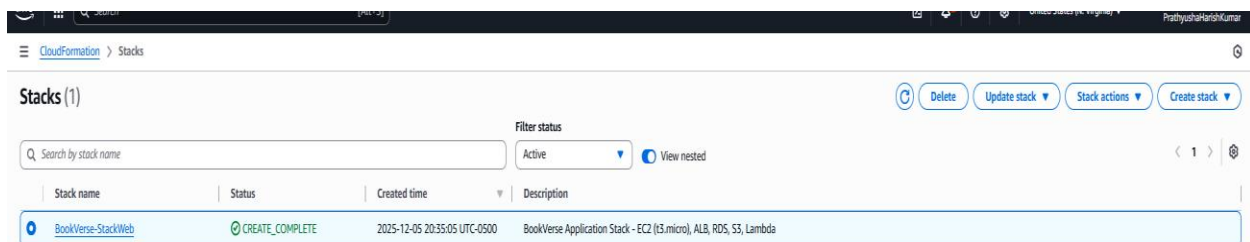
1. Template: bookverse-stack.yaml
2. Parameters: VpcId, PublicSubnet1/2, PrivateSubnet1/2, SG IDs from Terraform
3. Capabilities: CAPABILITY_NAMED_IAM
4. Stack name: CloudFormationStack-BookVerse.

```
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*.



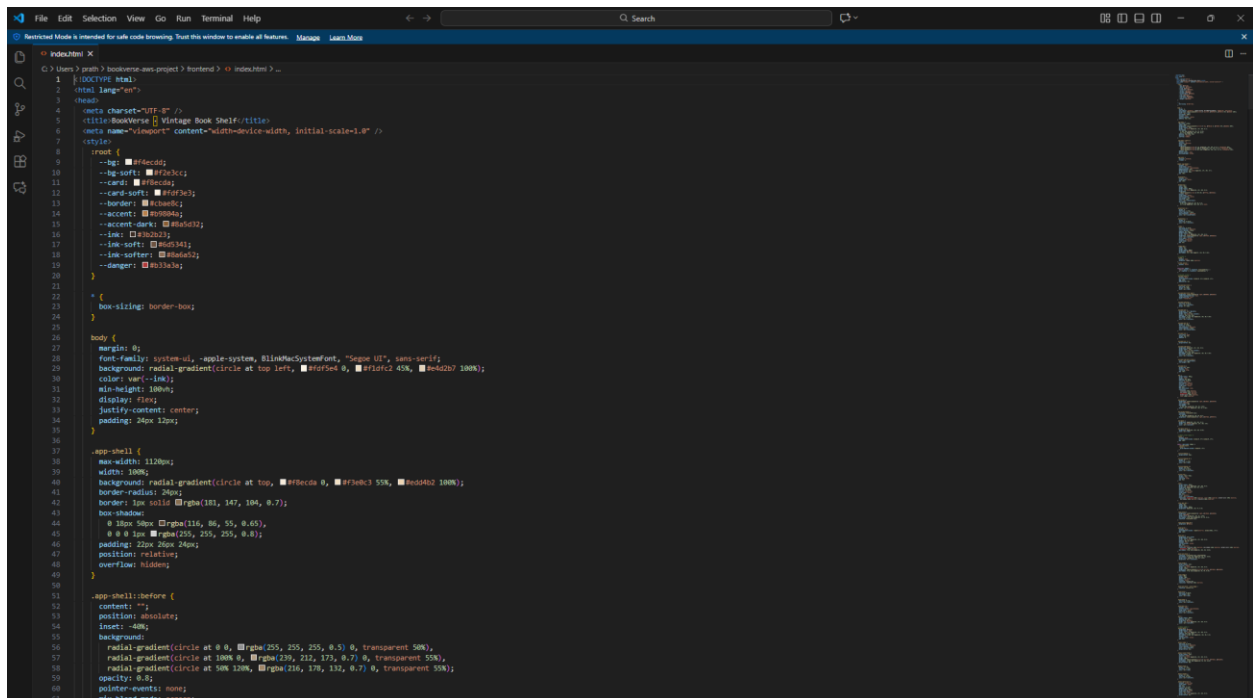
Stack name	Status	Created time	Description
BookVerse-StackWeb	CREATE_COMPLETE	2025-12-05 20:35:05 UTC-0500	BookVerse Application Stack - EC2 (t3.micro), ALB, RDS, S3, Lambda

Retrieved outputs:

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

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).

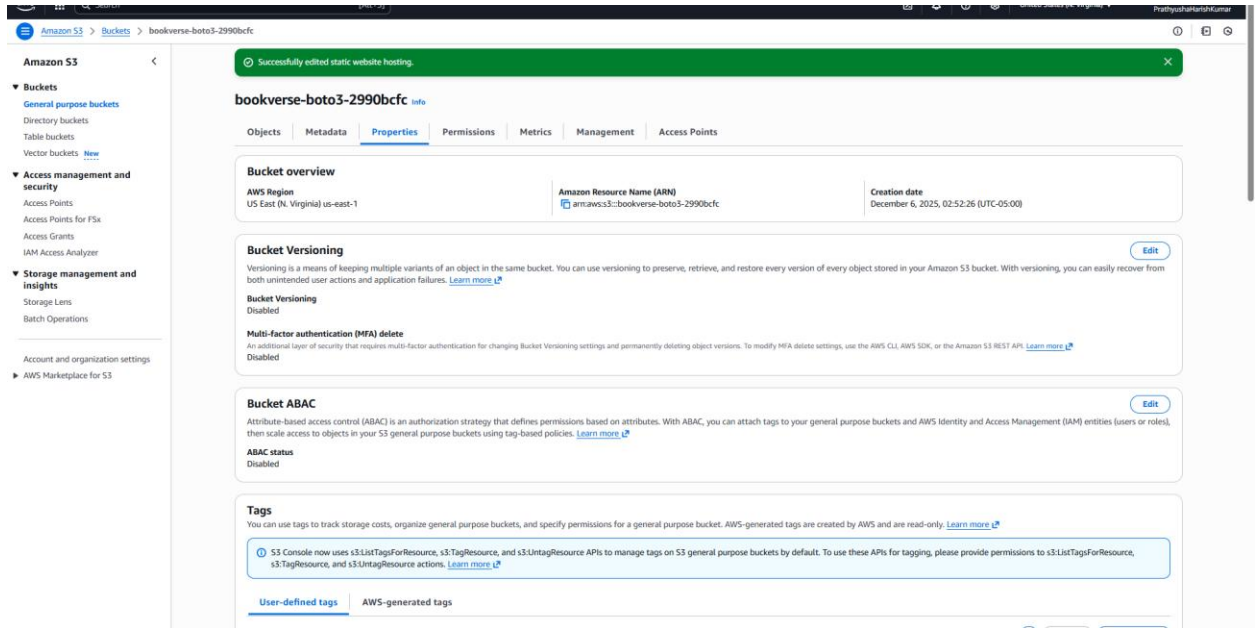


```
1 |<DOCTYPE html>
2 |<html lang="en">
3 |<head>
4 |<meta charset="utf-8" />
5 |<title>BookVerse [ Vintage Book Shelf / title ]>
6 |<meta name="viewport" content="width=device-width, initial-scale=1.0" />
7 |<style>
8 |<!--root-->
9 |<!--bg-->#f8d7da;
10 |<!--bg-soft-->#fff3f3;
11 |<!--card-->#fff3f3;
12 |<!--card-soft-->#fff3f3;
13 |<!--border-->#f8d7da;
14 |<!--accent-->#f8d7da;
15 |<!--accent-dark-->#f8d7da;
16 |<!--line-->#f8d7da;
17 |<!--line-soft-->#f8d7da;
18 |<!--line-soft-->#f8d7da;
19 |<!--decor-->#f8d7da;
20 |</style>
21 |
22 |* {
23 |  box-sizing: border-box;
24 |}
25 |
26 |body {
27 |  margin: 0;
28 |  font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", sans-serif;
29 |  background: radial-gradient(circle at top left, #f8d7da 0%, #f8d7da 45%, #f8d7da 100%);
30 |  color: var(--color);
31 |  min-height: 100vh;
32 |  display: flex;
33 |  justify-content: center;
34 |  padding: 20px 10px;
35 |}
36 |
37 |.app-shell {
38 |  width: 100%;
39 |  height: 100%;
40 |  background: radial-gradient(circle at top, #f8d7da 0%, #f8d7da 55%, #f8d7da 100%);
41 |  border-radius: 20px;
42 |  border: 1px solid #f8d7da;
43 |  box-shadow:
44 |    0 0 10px #f8d7da,
45 |    0 0 10px #f8d7da,
46 |    0 0 10px #f8d7da;
47 |  padding: 20px 20px 20px;
48 |  position: relative;
49 |  overflow: hidden;
50 |}
51 |
52 |.app-shell::before {
53 |  content: "";
54 |  position: absolute;
55 |  inset: -40px;
56 |  background:
57 |    radial-gradient(circle at 0 0, #f8d7da 25%, 25%, 25%, 0.5) 0, transparent 50%,
58 |    radial-gradient(circle at 100% 0, #f8d7da 25%, 25%, 175, 0.7) 0, transparent 55%,
59 |    radial-gradient(circle at 100% 100%, #f8d7da 25%, 175, 175, 0.7) 0, transparent 55%;
60 |  opacity: 0.5;
61 |  pointer-events: none;
62 |  mix-blend-mode: screen;
```

- **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)**

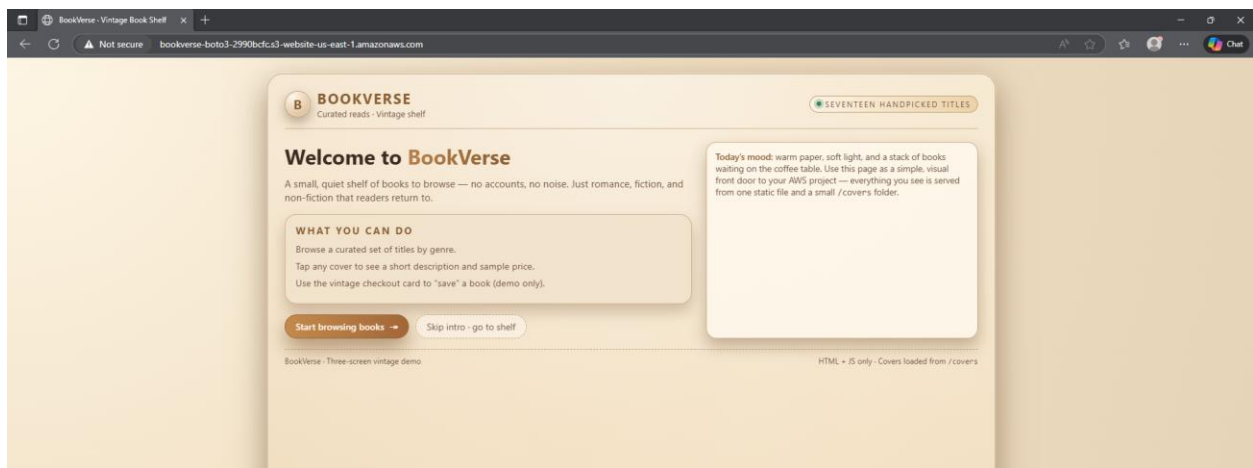
1. Opened S3 bucket *bookverse-boto3-2990bcfc*.
2. Enabled Static website hosting with index.html as the index document.
3. Noted the website endpoint:
http://bookverse-boto3-2990bcfc.s3-website-us-east-1.amazonaws.com/.

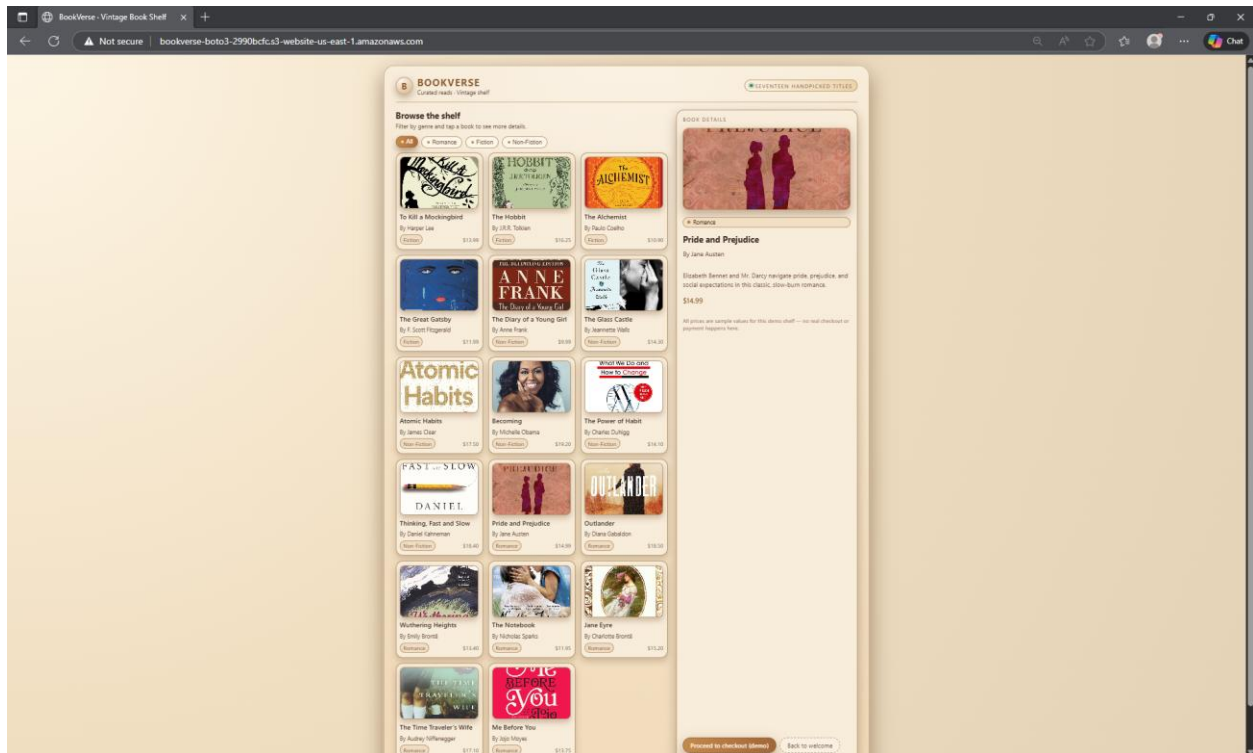
- **Test static site (Browser)**

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

Home

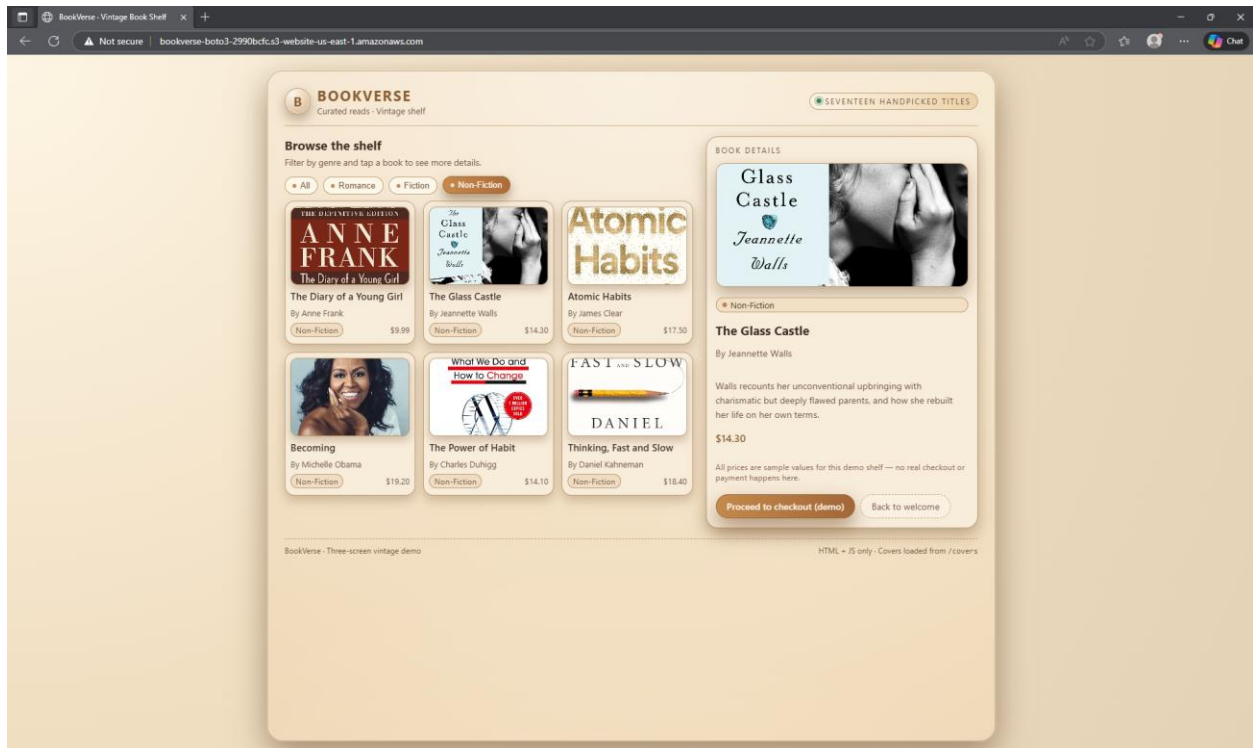
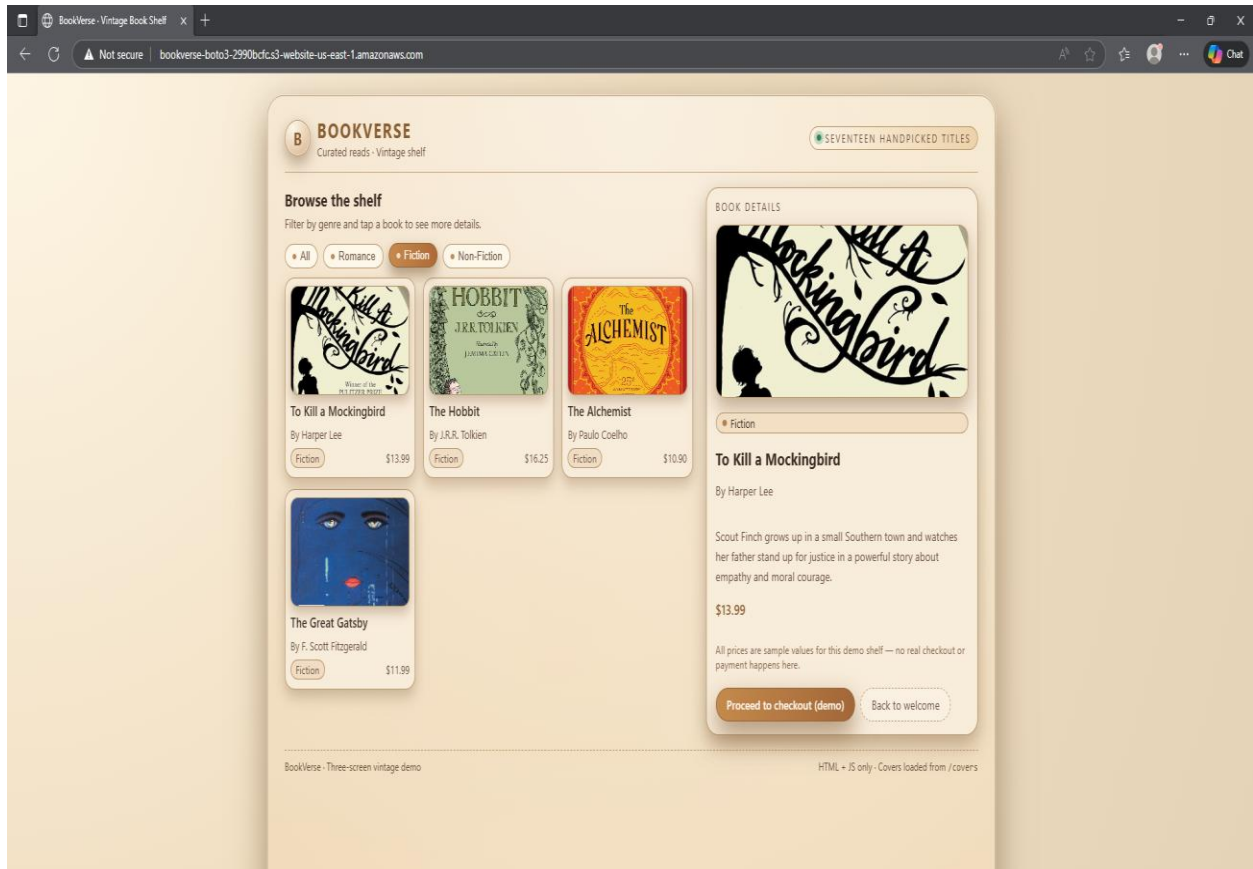
Link: <http://bookverse-boto3-2990bcfc.s3-website-us-east-1.amazonaws.com>





Genre Filtering





Purchasing book

The screenshot shows a web browser window with the address bar displaying "bookverse-boto3-2990bfc3-website-us-east-1.amazonaws.com". The page has a light beige background. At the top left is the "BOOKVERSE" logo with the tagline "Curated reads · Vintage shelf". To the right is a button labeled "SEVENTEEN HANDPICKED TITLES". Below the logo is a link "← Back to shelf". The main heading is "Checkout · Pride and Prejudice". A note states: "Fill in this card to simulate saving a book to your reading list. This is a front-end only demo — no orders are stored and no real payment is processed." The form contains three input fields: "FULL NAME" with the value "Ada Lovelace", "EMAIL" with "you@example.com", and "PHONE" with "+1 (555) 123-4567". At the bottom of the form are two buttons: "Complete demo checkout" (highlighted in orange) and "Cancel and go back". At the very bottom of the page, it says "BookVerse · Three-screen vintage demo" on the left and "HTML + JS only · Covers loaded from /covers" on the right.

This screenshot shows the same checkout page after the form has been submitted. The input fields now contain: "FULL NAME" as "ABCD", "EMAIL" as "ABCD@GMAIL.COM", and "PHONE" as "1111222333". The "Complete demo checkout" button remains highlighted. Below the form, a new message box appears: "Thank you! Your demo request for **Pride and Prejudice** has been noted. In a real system, this step could trigger a backend order workflow or email confirmation." The rest of the page layout, including the logo, navigation links, and footer, remains identical to the previous screenshot.

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.

```

Microsoft Windows [Version 10.0.26100.7171]
(c) Microsoft Corporation. All rights reserved.

C:\Users\prath>ssh -i bookverse-key.pem ec2-user@50.17.100.62
Warning: Identity file bookverse-key.pem not accessible: No such file or directory.
The authenticity of host '50.17.100.62 (50.17.100.62)' can't be established.
ED25519 key fingerprint is SHA256:Gfgv+VZF+iwAMb6E7dHNdsPipIdR00jn9BbVe20gtw8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '50.17.100.62' (ED25519) to the list of known hosts.
ec2-user@50.17.100.62: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).

C:\Users\prath>ssh -i C:\Users\prath\bookverse-aws-project\bookverse-key.pem ec2-user@50.17.100.62

#_
  _\ #####_ Amazon Linux 2023
#####_ \#####\
#####_ \###|
#####_ \#/ --- https://aws.amazon.com/linux/amazon-linux-2023
#####_ Vno' ' ->
#####_ /
#####_ - - - /
#####_ / - /
#####_ /m/ '

```

```

[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 : mariadb105-common-3:10.5.29-1.amzn2023.0.1.x86_64 2/5
Installing : perl-Sys-Hostname-1.23-477.amzn2023.0.7.x86_64 3/5
Installing : mariadb105-3:10.5.29-1.amzn2023.0.1.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     (\P) 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 -u ec2-user -h ec2-10-0-2-197.us-east-1.rds.amazonaws.com -P 3306 -e "CREATE DATABASE IF NOT EXISTS bookversedb;"
mysql: [Warning] Using a password on the command line interface can be insecure.
mysql> use bookversedb;
Database changed
mysql> create table books (
  id int(11) unsigned autoincrement primary key,
  title varchar(255),
  author varchar(255),
  genre varchar(50)
);
ERROR 1059 (42501): Table 'books' already exists
mysql> 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> create database if not exists bookversedb;
Query OK, 1 row affected (0.010 sec)

mysql> use bookversedb;
Database changed
mysql> create table books (
  id int(11) unsigned autoincrement primary key,
  title varchar(255),
  author varchar(255),
  genre varchar(50)
);
Query OK, 0 rows affected (0.029 sec)

mysql> 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> 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)

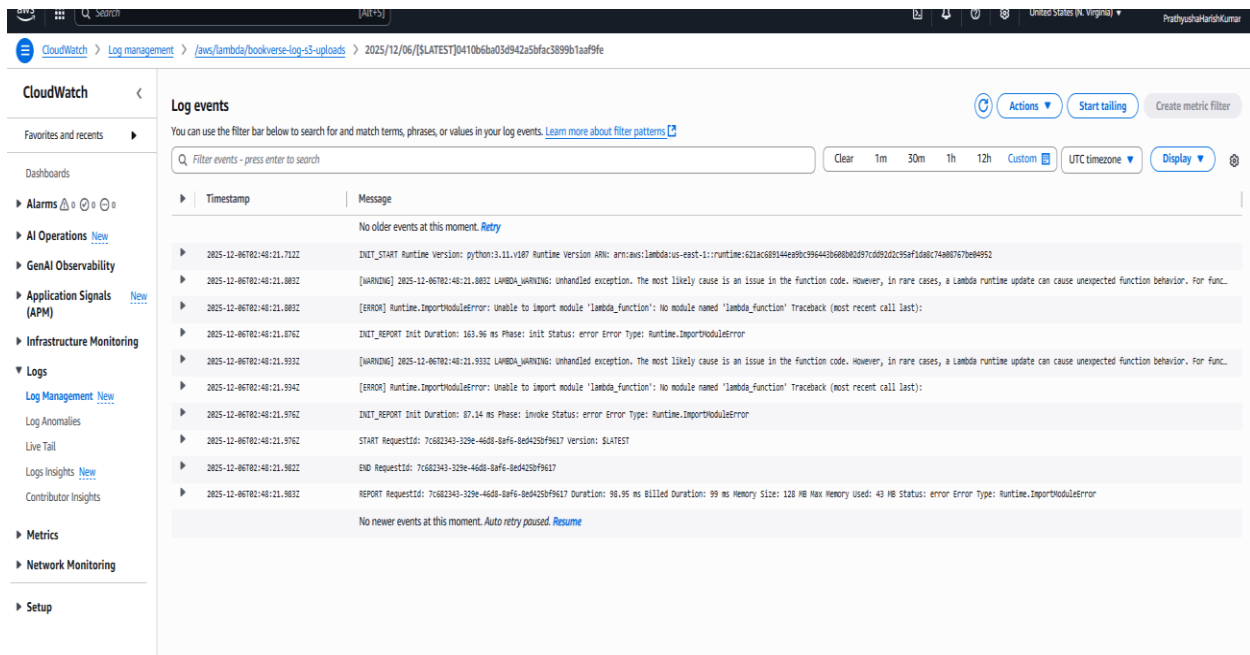
- **Lambda function for S3 uploads** (CloudFormation)
 1. *bookverse-log-s3-uploads* created via CloudFormation with inline Python.
 2. S3 bucket trigger configured via template.
- **Test Lambda manually** (Boto3)
 1. Used *invoke_lambda.py* from the boto3 folder:
 - Boto3 *lambda_client.invoke()* on *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.

```
Microsoft Windows [Version 10.0.26100.7171]
(c) Microsoft Corporation. All rights reserved.

C:\Users\prath>ssh -i C:\Users\prath\bookverse-aws-project\bookverse-key.pem ec2-user@50.17.100.62

Amazon Linux 2023

Last login: Sat Dec 6 10:58:04 2025 from 71.121.211.119
[ec2-user@ip-10-0-2-197 ~]$ sudo dnf install -y python3-pip
Last metadata expiration check: 0:18:29 ago on Sat Dec 6 10:58:33 2025.
Dependencies resolved.
=====
Package                Architecture      Version           Repository        Size
=====
Installing:
python3-pip            noarch            21.3.1-2.amzn2023.0.14  amazonlinux      1.8 M
Installing weak dependencies:
libcrypt-compat        x86_64            4.4.33-7.amzn2023  amazonlinux       92 k
=====
Transaction Summary
-----
Install 2 Packages

Total download size: 1.9 M
Installed size: 11 M
Downloading Packages:
(1/2): libcrypt-compat-4.4.33-7.amzn2023.x86_64.rpm 2.2 MB/s | 92 kB 00:00
(2/2): python3-pip-21.3.1-2.amzn2023.0.14.noarch.rpm 27 MB/s | 1.8 MB 00:00
-----
Total                                           19 MB/s | 1.9 MB 00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : libcrypt-compat-4.4.33-7.amzn2023.x86_64 1/1
Installing : python3-pip-21.3.1-2.amzn2023.0.14.noarch 2/2
Running scriptlet: python3-pip-21.3.1-2.amzn2023.0.14.noarch 2/2
Verifying : libcrypt-compat-4.4.33-7.amzn2023.x86_64 1/2
Verifying : python3-pip-21.3.1-2.amzn2023.0.14.noarch 2/2
Installed:
libcrypt-compat-4.4.33-7.amzn2023.x86_64 python3-pip-21.3.1-2.amzn2023.0.14.noarch

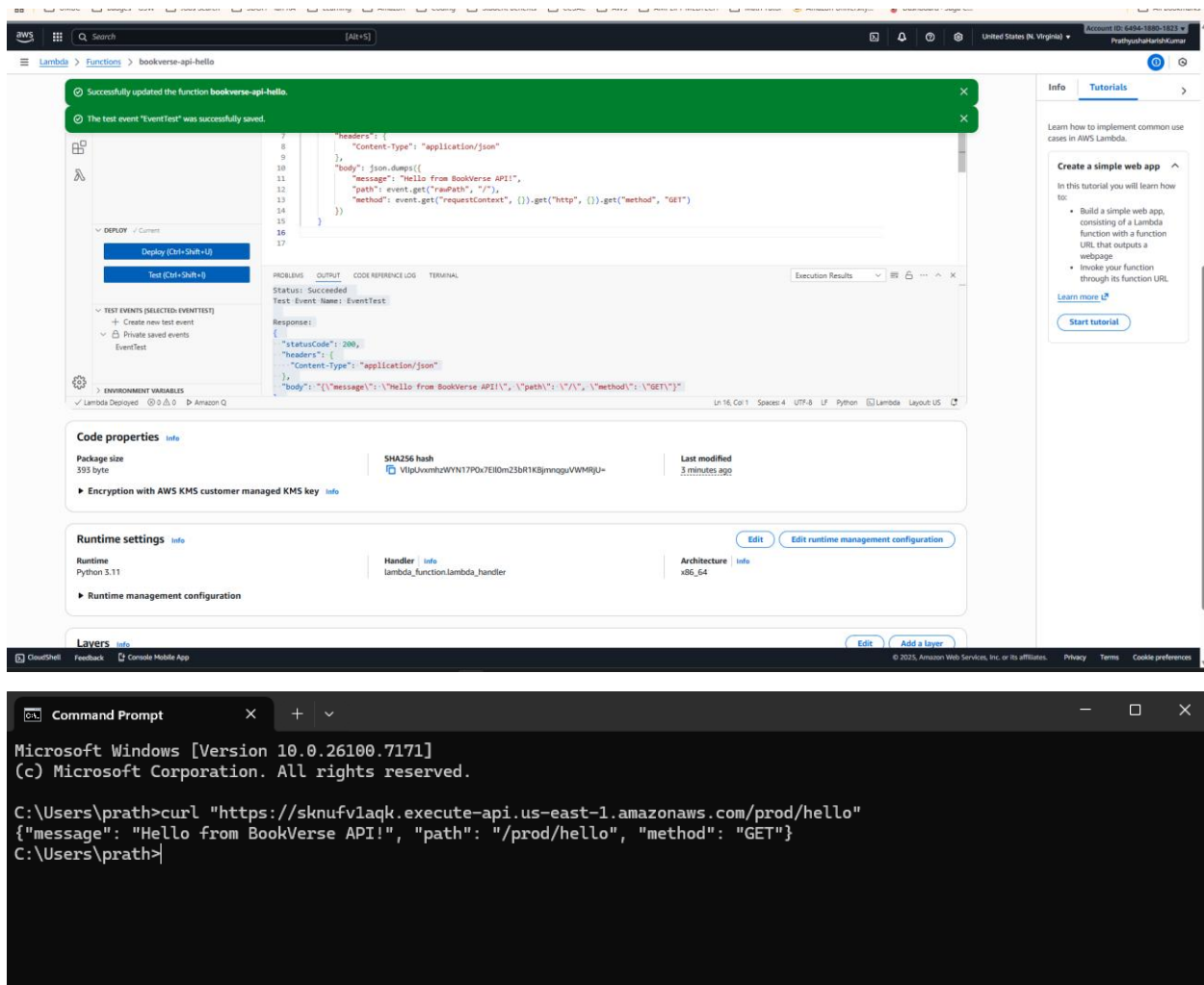
Complete!
[ec2-user@ip-10-0-2-197 ~]$ pip3 install --user boto3 requests
Collecting boto3
  Downloading boto3-1.42.0-py3-none-any.whl (140 kB)
    | 140 kB 35.7 MB/s
Requirement already satisfied: requests in /usr/lib/python3.9/site-packages (2.25.1)
```

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".*

```
1 {
2   "Comment": "BookVerse - simple book purchase workflow",
3   "startAt": "ValidateOrder",
4   "states": {
5     "ValidateOrder": {
6       "Type": "Task",
7       "Resource": "LAMBDA_ABN_HERE",
8       "Parameters": {
9         "step": "ValidateOrder",
10        "book.$": "$.book",
11        "user.$": "$.user"
12      },
13       "Next": "ProcessPayment"
14     },
15     "ProcessPayment": {
16       "Type": "Task",
17       "Resource": "LAMBDA_ABN_HERE",
18       "Parameters": {
19         "step": "ProcessPayment",
20         "book.$": "$.book",
21         "user.$": "$.user"
22      },
23       "Next": "CompleteOrder"
24     },
25     "CompleteOrder": {
26       "Type": "Task",
27       "Resource": "LAMBDA_ABN_HERE",
28       "Parameters": {
29         "step": "CompleteOrder",
30         "book.$": "$.book",
31         "user.$": "$.user"
32      },
33       "End": true
34     }
35   }
36 }
37
```

The test event "EventTest2" was successfully saved.

Notifications: 0 0 2 0 0 0 0

Function Log:

```
START RequestId: 8a32a2e6-077e-4eb2-b63e-bdc2396a203 Version: SLATEST
BookVerse workflow step: unknown
User: N/A, Book: N/A
END RequestId: 8a32a2e6-077e-4eb2-b63e-bdc2396a203
REPORT RequestId: 8a32a2e6-077e-4eb2-b63e-bdc2396a203 Duration: 1.39 ms Billed Duration: 86 ms Memory Size: 128 MB Max Memory Used: 34 MB Init
Duration: 83.72 ms
```

Code properties

Package size: 492 byte

Encryption with AWS KMS customer managed KMS key

Runtime settings

Runtime: Python 3.11

Handler: lambda_function.lambda_handler

Architecture: x86_64

State machine successfully created

BookVerseOrderWorkflow

Design

Workflow

The top level state machine properties for this workflow. [Learn more](#)

State machine query language: [info](#)
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

```
graph TD; Start((Start)) --> ValidateOrder[Lambda Invoke ValidateOrder]; ValidateOrder --> ProcessPayment[Lambda Invoke ProcessPayment]; ProcessPayment --> CompleteOrder[Lambda Invoke CompleteOrder]; CompleteOrder --> End((End));
```

Execution started successfully

Execution: 2eb1ed36-aeaf-460c-bba5-4469c71b34e7

Details Execution input and output Definition

State input

```
1 * {
2   "book": "Pride and Prejudice",
3   "user": "student@example.com",
4 }
```

State output

```
1 * {
2   "step": "CompleteOrder",
3   "result": "Order completed",
4   "book": "Pride and Prejudice",
5   "user": "student@example.com",
6 }
```

Graph view

Step details

Choose a step to view its details.

```
graph TD; Start((Start)) --> ValidateOrder[AWS Lambda Invoke ValidateOrder]; ValidateOrder --> ProcessPayment[AWS Lambda Invoke ProcessPayment]; ProcessPayment --> CompleteOrder[AWS Lambda Invoke CompleteOrder]; CompleteOrder --> End((End));
```

The screenshot displays the AWS Step Functions console for an execution named 'BookVerseOrderWorkflow'. The left sidebar shows navigation options like 'Step Functions', 'Dashboard', 'State machines', and 'Developer resources'. The main area shows the 'Event view' for the execution, which is currently in a 'Succeeded' state. A table lists 17 events, including 'ExecutionStarted', 'TaskStateEntered', 'LambdaFunctionScheduled', 'LambdaFunctionStarted', 'LambdaFunctionSucceeded', 'TaskStateExited', 'ProcessPayment', and 'CompleteOrder'. Each event includes details like ID, Type, Step, Resource, Started After, and Timestamp.

ID	Type	Step	Resource	Started After	Timestamp
1	ExecutionStarted			0	Dec 6, 2025, 10:51:45.645 (UTC-05:00)
2	TaskStateEntered	ValidateOrder		00:00:00.032	Dec 6, 2025, 10:51:45.677 (UTC-05:00)
3	LambdaFunctionScheduled	ValidateOrder	Lambda Log group	00:00:00.032	Dec 6, 2025, 10:51:45.677 (UTC-05:00)
4	LambdaFunctionStarted	ValidateOrder		00:00:00.117	Dec 6, 2025, 10:51:45.762 (UTC-05:00)
5	LambdaFunctionSucceeded	ValidateOrder		00:00:00.189	Dec 6, 2025, 10:51:45.834 (UTC-05:00)
6	TaskStateExited	ValidateOrder		00:00:00.212	Dec 6, 2025, 10:51:45.857 (UTC-05:00)
7	TaskStateEntered	ProcessPayment		00:00:00.212	Dec 6, 2025, 10:51:45.857 (UTC-05:00)
8	LambdaFunctionScheduled	ProcessPayment	Lambda Log group	00:00:00.212	Dec 6, 2025, 10:51:45.857 (UTC-05:00)
9	LambdaFunctionStarted	ProcessPayment		00:00:00.299	Dec 6, 2025, 10:51:45.944 (UTC-05:00)
10	LambdaFunctionSucceeded	ProcessPayment		00:00:00.346	Dec 6, 2025, 10:51:45.991 (UTC-05:00)
11	TaskStateExited	ProcessPayment		00:00:00.368	Dec 6, 2025, 10:51:46.013 (UTC-05:00)
12	TaskStateEntered	CompleteOrder		00:00:00.368	Dec 6, 2025, 10:51:46.013 (UTC-05:00)
13	LambdaFunctionScheduled	CompleteOrder	Lambda Log group	00:00:00.368	Dec 6, 2025, 10:51:46.013 (UTC-05:00)
14	LambdaFunctionStarted	CompleteOrder		00:00:00.445	Dec 6, 2025, 10:51:46.090 (UTC-05:00)
15	LambdaFunctionSucceeded	CompleteOrder		00:00:00.505	Dec 6, 2025, 10:51:46.150 (UTC-05:00)
16	TaskStateExited	CompleteOrder		00:00:00.530	Dec 6, 2025, 10:51:46.175 (UTC-05:00)
17	ExecutionSucceeded			00:00:00.562	Dec 6, 2025, 10:51:46.207 (UTC-05:00)

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.