Book Review Application - AWS Deployment Documentation

Project Overview

A 3-tier web application for browsing books and submitting reviews, deployed on AWS.

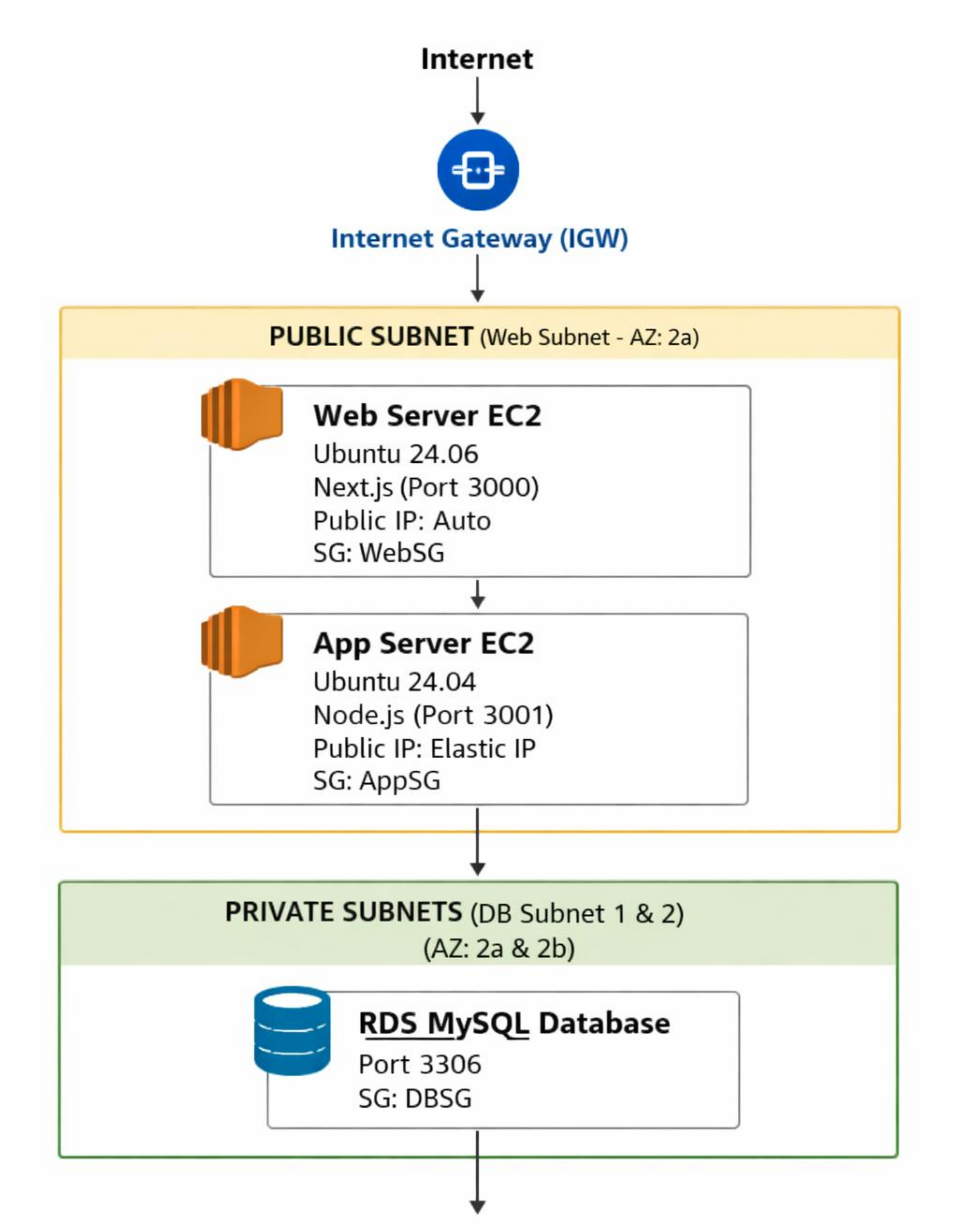
Application Features:

- Browse books and read reviews (unauthenticated users)
- Register and login (authentication)
- Submit book reviews (authenticated users only)

Tech Stack:

- Frontend: Next.js (Server-side rendering)
- Backend: Node.js + Express.js
- Database: MySQL (AWS RDS with Sequelize ORM)

Architecture - What Was Actually Built



Key Characteristics:

- Both Web and App servers in the same public subnet
- No load balancers
- No high availability (single instance per tier)
- No NAT Gateway
- Direct port access (:3000, :3001)

Step-by-Step Deployment

STEP 1: Create VPC

AWS Console: VPC Create VPC

Configuration:

- VPC Only (not "VPC and more")
- Name: Custom name (e.g., book-review-vpc)
- IPv4 CIDR: As per your planning (e.g., 10.0.0.0/20) Tenancy:
- Default

Action: Click "Create VPC"

STEP 2: Create Subnets

AWS Console: VPC Subnets Create subnet

Created 3 Subnets:

Subnet 1: Web Subnet (Public)

- Name: web-subnet-1
- VPC: Select your VPC
- Availability Zone: ap-southeast-2a (or your region's first AZ)
- IPv4 CIDR: As per your plan (e.g., 10.0.1.0/24)

Subnet 2: DB Subnet 1 (Private)

- Name: db-subnet-1
- VPC: Select your VPC
- Availability Zone: ap-southeast-2a
- IPv4 CIDR: As per your plan (e.g., 10.0.2.0/24)

Subnet 3: DB Subnet 2 (Private)

- Name: db-subnet-2
- VPC: Select your VPC
- Availability Zone: ap-southeast-2b
- IPv4 CIDR: As per your plan (e.g., 10.0.3.0/24)

Note: DB Subnet 2 is required because RDS DB Subnet Groups need at least 2 subnets in different AZs.

STEP 3: Create Internet Gateway

AWS Console: VPC Internet Gateways Create internet gateway

Configuration:

- Name: book-review-igw

Actions:

1. Create internet gateway
2. Select it Actions Attach to VPC
3. Select your VPC Attach

STEP 4: Create Route Tables

AWS Console: VPC Route Tables

Public Route Table

Create:

- Name: public-rt
- VPC: Select your VPC

Edit Routes:

- Click "Edit routes"
- Add route: 0.0.0.0/0 Target: Internet Gateway

Associate Subnets:

- Edit subnet associations
- Select: web-subnet-1
- Save

Private Route Table

The default route table automatically becomes private.

Rename it:

- Name: private-rt

Verify Routes:

- Should only have local route (10.0.0.0/20 local)
- No internet gateway route

Verify Associations:

- Should have db-subnet-1 and db-subnet-2

STEP 5: Create Security Groups

AWS Console: EC2 Security Groups Create security group

Web Server Security Group

Name: web-sg Description: Security group for web server VPC: Select your VPC

Inbound Rules:

Type	Protocol	Port	Source	Description
Custom TCP	TCP	3000	0.0.0.0/0	Next.js frontend
SSH	TCP	22	0.0.0.0/0	SSH access

Outbound Rules:

Type	Protocol	Port	Destination
All traffic	All	All	0.0.0.0/0

App Server Security Group

Name: app-sg Description: Security group for app server VPC: Select your VPC

Inbound Rules:

Type	Protocol	Port	Source	Description
Custom TCP	TCP	3001	sg-xxxxx (web-sg)	Node.js API from Web Server
SSH	TCP	22	0.0.0.0/0	SSH access

Outbound Rules:

Type	Protocol	Port	Destination
All traffic	All	All	0.0.0.0/0

Important: For port 3001, select "Custom" source type, then choose the web-sg security group ID (not IP address).

Database Security Group

Name: db-sg Description: Security group for RDS database VPC: Select your VPC

Inbound Rules:

Type	Protocol	Port	Source	Description
MYSQL/Aurora	TCP	3306	sg-xxxxx (app-sg)	MySQL from App Server

Outbound Rules:

Type	Protocol	Port	Destination
All traffic	All	All	0.0.0.0/0

Important: For port 3306, select "Custom" source type, then choose the app-sg security group ID.

STEP 6: Create Key Pair

AWS Console: EC2 Key Pairs Create key pair

Configuration:

- Name: Your choice (e.g., book-review-key)
- Key pair type: RSA
- Private key file format: .pem

Action:

- Create key pair
- Download and save the .pem file securely

STEP 7: Create RDS MySQL Database

AWS Console: RDS Databases Create database

Create DB Subnet Group First

RDS Subnet groups Create DB subnet group

Configuration:

- Name: mysql-subnet-group
- Description: Subnet group for MySQL
- VPC: Select your VPC
- Availability Zones: Select 2 AZs (e.g., 2a and 2b)
- Subnets: Select db-subnet-1 and db-subnet-2

Action: Create

Create Database Instance

Standard Create (not Easy Create)

Engine Options:

- Engine type: MySQL
- Version: MySQL 8.0 (default)

Templates:

- Free tier (for learning/demo)

Settings:

- DB instance identifier: database-1 (or your choice)
- Master username: admin
- Master password: Create and save securely
- Confirm password

Instance Configuration:

- DB instance class: db.t2.micro or db.t3.micro (free tier eligible)

Storage:

- Storage type: General Purpose SSD (gp2)
- Allocated storage: 20 GB (default)
- Uncheck "Enable storage autoscaling" (for demo)

Connectivity:

- VPC: Select your VPC
- DB subnet group: Select mysql-subnet-group
- Public access: No
- VPC security group: Choose existing Select db-sg
- Availability Zone: No preference (or select one)

Database Authentication:

- Password authentication

Additional Configuration:

- Initial database name: Leave blank (will create manually)
- Disable automated backups (for demo to save costs)
- Uncheck "Enable encryption" (for demo)

Action: Create database

Wait: Database creation takes 5-10 minutes. Status should become "Available".

Save the Endpoint:

- Once available, copy the Endpoint (e.g., database-1.c9u2ow2gwhdi.apsouth-1.rds.amazonaws.com)
- You'll need this for backend configuration

STEP 8: Launch App Server EC2 Instance

AWS Console: EC2 Instances Launch instances

Configuration:

Name: app-server

Application and OS Images:

- AMI: Ubuntu Server 24.04 LTS
- Architecture: 64-bit (x86)

Instance type: t2.micro

Key pair: Select your created key pair (e.g., book-review-key)

Network settings:

- Click "Edit"
- VPC: Select your VPC
- Subnet: Select web-subnet-1 (the public subnet) Auto-assign
- public IP:
  Enable (Important!
)
  - If you forget this, you'll need to allocate an Elastic IP later

- Firewall (security groups): Select existing security group Select: app-
- sg

Configure storage:

- 8 GB gp3 (default)

Action: Launch instance

Note About Public IP Issue

From the transcript, the instructor forgot to enable auto-assign public IP, so had to manually allocate an Elastic IP:

If you forgot to enable auto-assign public IP:

1. EC2 Elastic IPs Allocate Elastic IP address
2. Allocate from Amazon's pool of IPv4 addresses
3. Select the Elastic IP Actions Associate Elastic IP address
4. Instance: Select app-server
5. Associate

Setup App Server

SSH into the instance:

```
# On your local machine
# Set key permissions (Linux/Mac/WSL) chmod 400
book-review-key.pem

# SSH into app server
ssh -i "book-review-key.pem" ubuntu@<app-server-public-ip> On
Windows (using Command Prompt or PowerShell):
```

```
ssh -i "book-review-key.pem" ubuntu@<app-server-public-ip>
```

Update system:

```
sudo apt update
```

Install Node.js:

Go to: https://github.com/nodesource/distributions

Follow Ubuntu installation instructions:

```
# Download and run Node.js setup script
curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -

# Install Node.js sudo apt-get
install -y nodejs

# Verify installation
node -v npm -v
```

Install MySQL Client:

```
sudo apt install mysql-client -y

# Verify mysql --
version
```

Test Database Connection:

```
mysql -h <rds-endpoint> -u admin -p #
Enter password when prompted
```

Example:

```
mysql -h database-1.c9u2ow2gwhdi.ap-south-1.rds.amazonaws.com -u admin -p
```

If connection successful, you'll see:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

Create Database:

CREATE DATABASE book_review_db;
SHOW DATABASES;
exit;

Clone Application Repository:

cd ~
git clone https://github.com/pravinmishraaws/book-review-app.git cd
book-review-app/backend

Install Dependencies:

npm install
This installs all packages listed in package.json.

Create Environment Configuration:

nano .env
Add the following (replace with your actual values):

# Database Configuration
DB_HOST=database-1.c9u2ow2gwhdi.ap-south-1.rds.amazonaws.com
DB_USER=admin
DB_PASS=your_password_here
DB_NAME=book_review_db
DB_DIALECT=mysql

# JWT Secret
JWT_SECRET=mysecretkey

# CORS Configuration
ALLOWED_ORIGINS=http://:3000 Important
Notes:

- DB_HOST: Your RDS endpoint
- DB_PASS: Your RDS password (not DB_PASSWORD)
-

ALLOWED_ORIGINS: Will be filled after creating web server

Save: Ctrl+O, Enter, Ctrl+X

Start Backend Application:

node src/server.js Expected
Output:

Database 'book_review_db' connected successfully with SSL!
Database schema updated successfully! Server
running on port 3001

Note: Keep this terminal open. The application will stop if you close it or disconnect from SSH.

STEP 9: Launch Web Server EC2 Instance

AWS Console: EC2 Instances Launch instances

Configuration:

Name: web-server

Application and OS Images:

- AMI: Ubuntu Server 24.04 LTS

Instance type: t2.micro

Key pair: Select your key pair

Network settings:

- VPC: Select your VPC
- Subnet: Select web-subnet-1 (same public subnet as app server)
- Auto-assign public IP: Enable
- Firewall: Select existing web-sg

Configure storage: 8 GB gp3

Action: Launch instance

Setup Web Server

SSH into the instance:

```
ssh -i "book-review-key.pem" ubuntu@<web-server-public-ip>
```

Update system:

```
sudo apt update
```

Install Node.js:

```
curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash sudo apt-
get install -y nodejs

# Verify
node -v npm
-v
```

Clone Application Repository:

```
cd ~
git clone https://github.com/pravinmishraaws/book-review-app.git cd
book-review-app/frontend
```

Install Dependencies:

```
npm install
```

Create Environment Configuration:

```
nano .env.local Add:
```

NEXT_PUBLIC_API_URL=http://:3001

Critical:

- Use the PUBLIC IP of your App Server (NOT private IP like 10.0.x.x)
- Do NOT include /api at the end
- Example: NEXT_PUBLIC_API_URL=http://13.232.165.93:3001

Why Public IP? Your browser (on your laptop) cannot access private AWS IPs. It needs the public IP.

Save: Ctrl+O, Enter, Ctrl+X

Start Frontend Application:

```
npm run dev
```

Expected Output:

```
 Next.js 14.x.x
- Local: http://localhost:3000
- Network: http://0.0.0.0:3000

 Ready in 2.5s
```

STEP 10: Update Backend CORS Configuration

Now that you have the Web Server's public IP, update the backend configuration.

Go back to App Server terminal (or open a new SSH session):

ssh -i "book-review-key.pem" ubuntu@<app-server-public-ip> cd
~/book-review-app/backend nano .env

Update the ALLOWED_ORIGINS line:

ALLOWED_ORIGINS=http://:3000 Example:

ALLOWED_ORIGINS=http://13.233.0.185:3000 Save:
Ctrl+O, Enter, Ctrl+X

Restart Backend:

Stop the running server (Ctrl+C in the terminal where it's running)
Then start again: node src/server.js

Now you should see successful requests:

Server running on port 3001
Incoming request from: http://13.233.0.185:3000 No
more CORS errors!

STEP 11: Test the Application

Open your browser:

http://:3000

Example: http://13.233.0.185:3000

You should see:

- Homepage with list of books
- Book details and reviews when clicking on a book
- Register and Login functionality
- Ability to submit reviews (when logged in)

Check Browser Console (F12 Console):

- Should show no errors
- Network tab should show successful API calls to http://<app-server-ip>:3001/api/...

STEP 12: Configure DNS (Optional)

If you have a domain registered in Route 53:
AWS Console: Route 53 Hosted zones Your domain

Create Record:

- Record name: bookreview (or blank for root domain)
- Record type: A
- Value: <web-server-public-ip>
- TTL: 300
- Routing policy: Simple routing

Action: Create records

Access: http://bookreview.yourdomain.com:3000

Note: You still need to include port :3000 because the application isn't running on port 80.

Common Issues and Solutions (From Transcript)

Issue 1: Security Group Mistakes

Problem: Wrong security groups assigned to instances during creation.

Example: Web Server had App SG instead of Web SG.

Solution:

1. EC2 Instances Select instance
2. Actions Security Change security groups
3. Remove wrong SG, add correct SG
4. Save

Issue 2: Missing Public IP on App Server

Problem: Forgot to enable auto-assign public IP during instance launch.
Solution:

1. Allocate Elastic IP
2. Associate it with the App Server instance

Issue 3: CORS Errors

Problem: Backend showing:

Error: CORS policy: Not allowed by server
Root Cause: ALLOWED_ORIGINS in backend .env doesn't match Web Server's public IP.

Solution:

1. Get Web Server's exact public IP
2. Update backend .env: env ALLOWED_ORIGINS=http://13.233.0.185:3000
3. Restart backend: node src/server.js

Issue 4: "Books Not Available" Message

Problem: Frontend shows "books not available" or empty page.

Possible Causes:

1. Frontend .env.local has wrong backend URL
2. Backend not running
3. Database connection failed
4. Security groups blocking traffic

Solution:

Check backend is running:

```
# On App Server ps aux | grep node
curl http://localhost:3001/api/books
Check frontend can reach backend: #
On Web Server curl http://<app-server-
public-ip>:3001/api/books Verify
.env.local uses public IP:
```

```
# On Web Server
cat ~/book-review-app/frontend/.env.local
# Should show: NEXT_PUBLIC_API_URL=http://<public-ip>:3001
```

Issue 5: MySQL Connection Hangs

Problem: When connecting to database, command hangs indefinitely.

Root Cause: Database security group not allowing App Server.

Solution:

1. RDS Databases database-1 Connectivity & Security
2. Click on VPC security group
3. Inbound rules Edit
4. Ensure Port 3306 allows source from app-sg (security group ID)

Issue 6: High CPU Usage / Instance Unresponsive

Problem: t2.micro instance reaching 100% CPU, becoming slow or unresponsive.

From Transcript:

"I have noticed here... this was reaching 100%. So server was not responding well."

Temporary Solution:

- Reboot the instance
- EC2 Instances Select instance Instance state Reboot

Long-term Solution:

- Use larger instance type (t2.small or t3.small)
- Implement proper process management with PM2 Use
- Auto Scaling

Keeping Applications Running (Production Setup)

Current Issue: Applications stop when you close SSH or terminal.

Solution: Use PM2 (Process Manager)

On App Server:

```
# Install PM2 globally sudo
npm install -g pm2

# Start backend with PM2 cd ~/book-
review-app/backend pm2 start
src/server.js --name backend
```

Check status pm2
status

View logs pm2
logs backend

Make PM2 start on system boot pm2
startup
Follow the command it outputs and run it

pm2 save

On Web Server:

Install PM2 sudo npm
install -g pm2

Start frontend with PM2 cd ~/book-review-
app/frontend pm2 start npm --name
frontend -- run dev

For production build:
npm run build
pm2 start npm --name frontend -- start
Check status pm2
status

Make PM2 start on boot pm2
startup
Run the command it outputs

pm2 save

Benefits of PM2:

- Applications run in background
- Auto-restart on crash
- Auto-start on server reboot
- Easy log management

Testing Checklist

Backend Tests (from App Server):

Test database connection
mysql -h <rds-endpoint> -u admin -p -e "SHOW DATABASES;"

Test backend API locally curl
http://localhost:3001/api/books

Should return JSON with books

Frontend to Backend Test (from Web Server):

Test if Web Server can reach App Server curl
http://<app-server-public-ip>:3001/api/books

Should return JSON with books

Browser Tests:

1. Homepage:

   http://<web-server-ip>:3000

   - Should display list of books

2. Book Details:
   Click on any book
   - Should show book details and reviews

3. Register:
   Click Register
   - Create new account
   - Should redirect to login after success

4. Login:
   Login with credentials
   - Should show username in navbar
   - Logout button should appear

5. Submit Review:
   Go to a book, write review

- Should appear immediately after submission

6. Browser Console:
   F12 Console
   - Should show no errors
   - Network tab should show successful API calls (status 200)

Architecture Summary

Component	Details
VPC	Custom VPC in Sydney region
Subnets	1 Public (Web & App), 2 Private (Database)
Internet Gateway	1 (attached to VPC)
Route Tables	2 (Public with IGW route, Private without)
Security Groups	3 (Web-SG, App-SG, DB-SG)
EC2 Instances	2 (Web Server, App Server) - both in public subnet
RDS	1 MySQL instance in private subnets
Load Balancers	0 (not used in demo)
NAT Gateway	0 (not needed, both servers in public subnet)
Elastic IP	1 (for App Server)
High Availability	No (single instance per tier)

What This Demo Does NOT Include

(But students should implement):

- ❌ Load Balancers (Public ALB, Internal ALB)
- ❌ High Availability (Multi-AZ for compute)
- ❌ Separate subnets for each tier
- ❌ NAT Gateway
- ❌ Nginx reverse proxy
- ❌ Running on port 80 (uses 3000 and 3001)
- ❌ Auto Scaling Groups
- ❌ Multi-AZ RDS
- ❌ Read Replicas
- ❌ SSL/TLS certificates

❌ Bastion Host for SSH
❌ CloudWatch monitoring
❌ CI/CD pipeline

Resource Summary

Resource Type	Count	Purpose
VPC	1	Network isolation
Subnets	3	1 public, 2 private
Internet Gateway	1	Internet access for public subnet
Route Tables	2	Public & private routing
Security Groups	3	Network-level security
EC2 Instances	2	Web and App servers
Elastic IP	1	Static IP for App Server
RDS MySQL	1	Database (single AZ)
Key Pair	1	SSH access

Access URLs

Application Access:

http://:3000
With DNS (if configured):

http://yourdomain.com:3000 Backend
API (for testing):

http://:3001/api/books

Project Completion

VPC and Network: Created and configured Security Groups: Configured with proper rules Database: RDS MySQL running and accessible Backend: Node.js API running and connected to database Frontend: Next.js application running and connected to backend CORS: Configured correctly Application: Fully functional - browse, register, login, review

Congratulations! Your 3-tier application is now deployed on AWS!

End of Documentation