EpicBooks Application - Installation and Configuration Guide

Table of Contents

---

1. Project Overview

What is EpicBooks?

EpicBooks is a full-stack web application for managing a digital bookstore. It allows users to:

- Browse and search for books
- View detailed book information
- Manage book inventory (admin functionality)
- User authentication and authorization

Application Type

This is a **monolithic web application** with a **3-tier architecture**:

- **Frontend**: EJS templates (server-side rendered)
- **Backend**: Node.js with Express.js
- **Database**: MySQL (RDS)

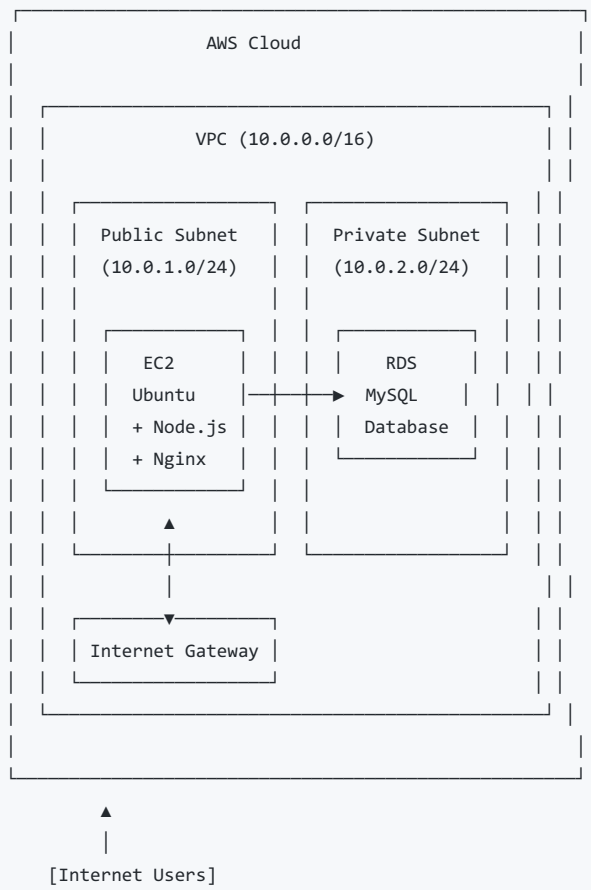
Why This Architecture?

- **Simplicity**: Single codebase, easier to develop and maintain
- **Cost-effective**: One server handles everything
- **Suitable for small to medium applications**
- **Industry-standard**: Commonly used for business applications

---

2. Architecture & Technology Stack

2.1 Application Architecture



2.2 Technology Stack

Component	Technology	Purpose
Runtime	Node.js v18+	JavaScript runtime for server-side execution
Framework	Express.js 4.x	Web application framework
View Engine	EJS	Server-side template rendering
Database	MySQL 8.0	Relational database management
ORM	Sequelize	Database abstraction and migrations
Reverse Proxy	Nginx	Production web server and load balancer
Process Manager	PM2	Production process manager
Version Control	Git	Source code management
Cloud Provider	AWS	Infrastructure hosting

2.3 Why These Technologies?

Node.js

- **Non-blocking I/O**: Handles multiple concurrent requests efficiently
- **JavaScript everywhere**: Same language for frontend and backend
- **Rich ecosystem**: NPM has packages for almost everything
- **Scalability**: Event-driven architecture supports high concurrency

Express.js

- **Minimalist**: Provides essential features without bloat

- **Middleware support**: Easy to add functionality
- **Routing**: Clean URL management
- **Industry standard**: Most popular Node.js framework

MySQL

- **ACID compliance**: Ensures data integrity
- **Relational model**: Perfect for structured data (books, users, orders)
- **Performance**: Fast query execution with proper indexing
- **Widely supported**: Mature ecosystem and community

Nginx

- **Reverse proxy**: Routes requests to Node.js application
- **Load balancing**: Distributes traffic across instances
- **SSL termination**: Handles HTTPS encryption
- **Static file serving**: Efficiently serves CSS, JS, images
- **Security**: Acts as a buffer between internet and application

PM2

- **Auto-restart**: Restarts app if it crashes
- **Clustering**: Runs multiple app instances
- **Monitoring**: Real-time performance metrics
- **Zero-downtime deployment**: Updates without service interruption

---

3. Prerequisites

3.1 Required Knowledge

- Basic Linux command line
- Understanding of web applications
- Familiarity with Git
- AWS account access

3.2 Required Tools

- SSH client (Terminal on Mac/Linux, PuTTY on Windows)
- Web browser
- Text editor (optional, for local reference)

3.3 AWS Account Requirements

- Active AWS account
- IAM user with EC2 and RDS permissions
- Credit card for billing (free tier eligible)

---

4. AWS Infrastructure Setup

Phase 1: VPC Architecture

What is a VPC?

A **Virtual Private Cloud (VPC)** is your isolated network in AWS. Think of it as your own data center in the cloud.

Why Do We Need a VPC?

- **Security**: Isolates your resources from other AWS users
- **Control**: You define IP ranges, subnets, and routing
- **Compliance**: Meets security requirements
- **Network segmentation**: Separates public and private resources

Step 1.1: Create VPC

```
 # VPC Configuration
Name: epicbooks-vpc
CIDR Block: 10.0.0.0/16
```

Understanding CIDR (10.0.0.0/16):

- **10.0.0.0**: Network address
- **/16**: Means first 16 bits are network, remaining 16 bits for hosts
- **Total IPs**: 65,536 IP addresses available
- **Why /16?**: Provides plenty of IPs for subnets and future growth

AWS Console Steps:

1. Go to VPC Dashboard
2. Click "Create VPC"
3. Select "VPC only"
4. Enter name: `epicbooks-vpc`
5. IPv4 CIDR: `10.0.0.0/16`
6. Click "Create VPC"

Step 1.2: Create Internet Gateway

What is an Internet Gateway (IGW)?

- Gateway between your VPC and the internet
- Allows public subnets to access the internet
- Required for any internet-facing resources

Why Do We Need It?

- EC2 instance needs to download updates
- Users need to access the application
- Enables outbound internet connections

AWS Console Steps:

1. Go to VPC → Internet Gateways
2. Click "Create internet gateway"
3. Name: `epicbooks-igw`
4. Click "Create"
5. Select the IGW → Actions → Attach to VPC
6. Select `epicbooks-vpc`
7. Click "Attach"

Step 1.3: Create Subnets

What is a Subnet? A subnet divides your VPC into smaller network segments.

Why Two Subnets?

- **Public Subnet**: For resources that need internet access (EC2)
- **Private Subnet**: For resources that should be hidden (RDS)
- **Security principle**: Database should not be directly accessible from internet

Public Subnet (EC2)

```
Name: epicbooks-public-subnet
CIDR: 10.0.1.0/24
Availability Zone: us-east-1a
```

Understanding 10.0.1.0/24:

- **10.0.1.0**: Subnet network address
- **/24**: First 24 bits are network, last 8 bits for hosts
- **Total IPs**: 256 IP addresses (251 usable)
- **Why /24?**: Enough IPs for multiple EC2 instances if needed

AWS Console Steps:

1. Go to VPC → Subnets
2. Click "Create subnet"
3. Select VPC: `epicbooks-vpc`
4. Subnet name: `epicbooks-public-subnet`
5. Availability Zone: `us-east-1a`
6. IPv4 CIDR: `10.0.1.0/24`
7. Click "Create subnet"
8. Select the subnet → Actions → Modify auto-assign IP settings
9. Enable "Auto-assign IPv4"
10. Click "Save"

Why auto-assign IPv4?

- EC2 instances automatically get public IP addresses
- Simplifies instance launch
- Required for internet connectivity

Private Subnet (RDS)

```
Name: epicbooks-private-subnet
CIDR: 10.0.2.0/24
Availability Zone: us-east-1a
```

AWS Console Steps:

1. Click "Create subnet"
2. Select VPC: `epicbooks-vpc`
3. Subnet name: `epicbooks-private-subnet`
4. Availability Zone: `us-east-1a`
5. IPv4 CIDR: `10.0.2.0/24`
6. Click "Create subnet"

Note: No public IP auto-assignment for private subnet.

Step 1.4: Create and Configure Route Tables

What is a Route Table?

- Defines how network traffic is directed
- Contains rules (routes) for traffic routing
- Each subnet must be associated with a route table

Public Route Table

AWS Console Steps:

1. Go to VPC → Route Tables
2. Find the main route table for `epicbooks-vpc`
3. Click "Edit routes"
4. Add route:
    - Destination: `0.0.0.0/0`
    - Target: Select `epicbooks-igw`
5. Click "Save routes"
6. Go to "Subnet associations" tab
7. Click "Edit subnet associations"
8. Select `epicbooks-public-subnet`
9. Click "Save associations"

Understanding 0.0.0.0/0:

- **Destination**: All IP addresses (the entire internet)
- **Meaning**: "Send all non-local traffic to the Internet Gateway"
- **Purpose**: Enables internet access

Private Route Table

AWS Console Steps:

1. Click "Create route table"
2. Name: `epicbooks-private-rt`
3. VPC: `epicbooks-vpc`
4. Click "Create"
5. Go to "Subnet associations"
6. Click "Edit subnet associations"
7. Select `epicbooks-private-subnet`
8. Click "Save associations"

Why no internet gateway route?

- Database should not be directly accessible from internet
- Only EC2 in public subnet can communicate with it
- Enhanced security posture

Step 1.5: Create Security Groups

What is a Security Group?

- Virtual firewall for your resources
- Controls inbound and outbound traffic
- Stateful: Return traffic automatically allowed

EC2 Security Group

```
Name: epicbooks-ec2-sg
Description: Security group for EC2 web server
```

Inbound Rules:

Type	Protocol	Port	Source	Purpose
SSH	TCP	22	Your IP	Remote server access
HTTP	TCP	80	0.0.0.0/0	Web traffic
HTTPS	TCP	443	0.0.0.0/0	Secure web traffic
Custom	TCP	8080	0.0.0.0/0	Node.js application (testing)

AWS Console Steps:

1. Go to EC2 → Security Groups
2. Click "Create security group"
3. Name: `epicbooks-ec2-sg`
4. Description: `Security group for EC2 web server`
5. VPC: `epicbooks-vpc`
6. Add inbound rules (as table above)
7. Outbound rules: Keep default (all traffic allowed)
8. Click "Create security group"

Why these ports?

- **Port 22 (SSH)**: You need to connect to server for configuration
- **Port 80 (HTTP)**: Standard web traffic (will redirect to HTTPS)
- **Port 443 (HTTPS)**: Secure web traffic (production)
- **Port 8080**: Node.js default port (for testing before Nginx setup)

Security Note:

- For production, restrict SSH to your IP only
- Remove port 8080 rule after Nginx is configured
- Consider using a VPN for SSH access

RDS Security Group

```
Name: epicbooks-rds-sg
Description: Security group for RDS MySQL database
```

Inbound Rules:

Type	Protocol	Port	Source	Purpose
MySQL	TCP	3306	epicbooks-ec2-sg	Database access from EC2 only

AWS Console Steps:

1. Click "Create security group"
2. Name: `epicbooks-rds-sg`
3. Description: `Security group for RDS MySQL database`
4. VPC: `epicbooks-vpc`
5. Add inbound rule:
   - Type: `MySQL/Aurora`
   - Source: `epicbooks-ec2-sg` (select the security group, not an IP)
6. Click "Create security group"

Why source is EC2 security group?

- Only EC2 instance can access database
- No direct internet access to database
- If EC2 IP changes, rule still works
- Best practice for database security

Phase 2: RDS MySQL Database

What is Amazon RDS?

RDS (Relational Database Service) is AWS's managed database service.

Why Use RDS Instead of Installing MySQL on EC2?

Feature	RDS	Self-Managed MySQL on EC2
Backups	Automated, daily	Manual setup required
Updates	Automatic patches	Manual updates
Scalability	Easy resize	Complex migration
High Availability	Multi-AZ with one click	Complex setup
Monitoring	Built-in CloudWatch	Manual setup
Maintenance	AWS handles it	You manage everything
Cost	Pay for what you use	Pay for EC2 + your time

Step 2.1: Create DB Subnet Group

What is a DB Subnet Group?

- Collection of subnets for RDS to use
- Required for placing database in VPC
- Enables Multi-AZ deployments

AWS Console Steps:

1. Go to RDS Dashboard
2. Click "Subnet groups" in left menu
3. Click "Create DB subnet group"
4. Name: `epicbooks-db-subnet-group`
5. Description: `Subnet group for EpicBooks database`
6. VPC: `epicbooks-vpc`
7. Add subnets:
    - Availability Zone: `us-east-1a`
    - Subnet: `epicbooks-private-subnet` (10.0.2.0/24)
8. Click "Create"

Why do we need this?

- RDS requires at least 2 subnets in different AZs for high availability
- For this project, we use one subnet (sufficient for development)
- Production would use multiple subnets for redundancy

Step 2.2: Create RDS MySQL Instance

Database Configuration:

```
 Engine: MySQL 8.0
Template: Free tier
DB Instance Identifier: epicbooks-db
Master Username: admin
Master Password: [Create secure password]
DB Instance Class: db.t3.micro
Storage: 20 GB GP2 SSD
VPC: epicbooks-vpc
Subnet Group: epicbooks-db-subnet-group
Public Access: No
Security Group: epicbooks-rds-sg
Initial Database Name: epicbooks
```

AWS Console Steps:

1. **Go to RDS Dashboard**
2. **Click "Create database"**
3. **Choose database creation method:**
    - Select "Standard create"
    - **Why?** More control over configuration
4. **Engine options:**
    - Engine type: `MySQL`

- Version: `MySQL 8.0.35` (or latest 8.0.x)
- **Why MySQL 8.0?**
  - Better performance than 5.7
  - JSON support
  - Window functions
  - CTE (Common Table Expressions)

5. **Templates:**
   - Select "Free tier"
   - **Why?** No cost for 750 hours/month
   - **Limitations**: Single-AZ only, limited storage
6. **Settings:**
   - DB instance identifier: `epicbooks-db`
   - Master username: `admin`
   - Master password: `[YourSecurePassword123!]`
   - **Why strong password?** Security best practice
   - **Note**: Save password securely; you'll need it later
7. **DB instance class:**
   - Select "Burstable classes"
   - Choose `db.t3.micro` (1 vCPU, 1 GB RAM)
   - **Why t3.micro?** Free tier eligible, sufficient for development
8. **Storage:**
   - Storage type: `General Purpose SSD (gp2)`
   - Allocated storage: `20 GB`
   - **Disable** "Enable storage autoscaling"
   - **Why 20 GB?** Free tier limit
   - **Why disable autoscaling?** Avoid unexpected costs
9. **Connectivity:**
   - VPC: `epicbooks-vpc`
   - DB subnet group: `epicbooks-db-subnet-group`
   - Public access: `No`
   - **Why no public access?** Security - only EC2 can access
   - VPC security group: `epicbooks-rds-sg`
   - Availability Zone: `us-east-1a`
10. **Database authentication:**
    - Keep "Password authentication"
    - **Why?** Simple, sufficient for our needs
    - IAM authentication adds complexity
11. **Additional configuration:**
    - Initial database name: `epicbooks`
    - **Why?** Creates database on launch; no manual creation needed
    - DB parameter group: `default.mysql8.0`
    - Backup retention: `7 days` (free tier limit)
    - **Disable** Enhanced monitoring (costs extra)
12. **Click "Create database"**

Wait time: 10-15 minutes for database to become available.

Step 2.3: Note Database Endpoint

After database is available:

1. Go to RDS → Databases
2. Click on `epicbooks-db`
3. In "Connectivity & security" tab, copy the **Endpoint**
   - Example: `epicbooks-db.abcd1234.us-east-1.rds.amazonaws.com`
4. Save this endpoint - you'll need it for application configuration

What is an endpoint?

- DNS name to connect to your database
- Remains consistent even if underlying IP changes
- Format: `<instance-name>.<random-id>.<region>.rds.amazonaws.com`

Phase 3: EC2 Instance Setup

Step 3.1: Launch EC2 Instance

Instance Configuration:

```
Name: epicbooks-server
AMI: Ubuntu Server 22.04 LTS
Instance Type: t2.micro
Key Pair: epicbooks-key (create new)
VPC: epicbooks-vpc
Subnet: epicbooks-public-subnet
Auto-assign Public IP: Enable
Security Group: epicbooks-ec2-sg
Storage: 8 GB GP2
```

AWS Console Steps:

1. **Go to EC2 Dashboard**
2. **Click "Launch Instance"**
3. **Name and tags:**
   - Name: `epicbooks-server`
   - **Why name?** Easy identification in console
4. **Application and OS Images (AMI):**
   - Select "Ubuntu"
   - Choose "Ubuntu Server 22.04 LTS (HVM), SSD Volume Type"
   - Architecture: `64-bit (x86)`
   - **Why Ubuntu 22.04 LTS?**
     - LTS: Long-term support (5 years updates)
     - Popular: Large community, lots of tutorials
     - Stable: Well-tested for production
     - Package availability: Easy to install software
5. **Instance type:**
   - Select `t2.micro` (1 vCPU, 1 GB RAM)
   - **Why t2.micro?** Free tier eligible
   - **Is 1 GB RAM enough?** Yes, for development. Production needs more.
6. **Key pair:**
   - Click "Create new key pair"
   - Key pair name: `epicbooks-key`
   - Key pair type: `RSA`
   - Private key file format:
     - `.pem` for Mac/Linux
     - `.ppk` for Windows/PuTTY
   - Click "Create key pair"
   - **IMPORTANT**: Save the .pem file securely
   - **Why?** You can only download once; it's your SSH key
7. **Network settings:**
   - VPC: `epicbooks-vpc`
   - Subnet: `epicbooks-public-subnet`
   - Auto-assign public IP: `Enable`
   - Firewall (security groups): Select existing
   - Security group: `epicbooks-ec2-sg`
8. **Configure storage:**
   - Size: `8 GB`
   - Volume type: `gp2` (General Purpose SSD)
   - Delete on termination: `Yes`
   - **Why 8 GB?** Sufficient for OS + Node.js + application
9. **Advanced details:**
   - Leave all default settings
10. **Review and Launch:**
    - Review configuration
    - Click "Launch instance"

Wait time: 2-3 minutes for instance to be running.

Step 3.2: Connect to EC2 Instance

For Mac/Linux:

```
# Set correct permissions for key file
chmod 400 epicbooks-key.pem

# Connect to instance
ssh -i "epicbooks-key.pem" ubuntu@<your-ec2-public-ip>
```

Example:

```
ssh -i "epicbooks-key.pem" ubuntu@54.123.45.67
```

Why chmod 400?

- SSH requires private key to be readable only by you
- 400 permissions = read-only for owner
- Security measure to prevent unauthorized use

For Windows (using PuTTY):

1. **Download PuTTY** from https://www.putty.org/
2. **Open PuTTYgen:**
   - Click "Load"
   - Select your .pem file
   - Click "Save private key" (save as .ppk)
3. **Open PuTTY:**
   - Host Name: `ubuntu@<your-ec2-public-ip>`
   - Port: `22`
   - Connection type: `SSH`
   - In left menu: Connection → SSH → Auth
   - Browse and select your .ppk file
   - Click "Open"

First Connection Warning:

```
The authenticity of host can't be established.
Are you sure you want to continue connecting (yes/no)?
```

- Type: `yes`
- **Why?** This is normal for first connection
- Adds server to known hosts

---

5. Server Configuration

Phase 4: Install Dependencies

Step 4.1: Update System

```
sudo apt update && sudo apt upgrade -y
```

What this does:

- `apt update` : Updates package list from repositories
- `apt upgrade` : Upgrades all installed packages
- `-y` : Automatically says "yes" to all prompts

Why update first?

- Gets latest security patches
- Ensures package compatibility
- Fixes known bugs

Expected output:

```
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
...
Reading package lists... Done
Building dependency tree... Done
```

Time: 5-10 minutes

Step 4.2: Install Node.js

Why NodeSource Repository?

- Ubuntu's default Node.js is outdated
- NodeSource provides latest LTS versions
- Officially supported by Node.js

Install Node.js 18.x LTS:

```
# Add NodeSource repository
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -

# Install Node.js
sudo apt install -y nodejs

# Verify installation
node --version  # Should show v18.x.x
npm --version   # Should show v9.x.x
```

Understanding the commands:

- `curl -fsSL` : Downloads NodeSource setup script
  - `-f` : Fail silently on HTTP errors
  - `-s` : Silent mode (no progress)
  - `-S` : Show errors
  - `-L` : Follow redirects
- `| sudo -E bash -` : Pipes to bash for execution
  - `-E` : Preserves environment variables
- `sudo apt install -y nodejs` : Installs Node.js package

Why Node.js 18?

- LTS (Long-Term Support) until April 2025
- Stable and production-ready
- Security updates guaranteed
- Compatible with latest npm packages

Expected output:

```
Node.js v18.17.0
npm v9.6.7
```

Step 4.3: Install MySQL Client

```
# Install MySQL client tools
sudo apt install -y mysql-client

# Verify installation
mysql --version
```

Why install MySQL client?

- Connect to RDS database from EC2
- Run SQL queries and migrations
- Debug database issues
- Import/export data

What's installed:

- `mysql` : Command-line client
- `mysqldump` : Backup tool
- `mysql_config` : Configuration utility

Why not full MySQL server?

- We're using RDS for the database
- Client is enough for connection
- Saves resources on EC2

Expected output:

```
mysql  Ver 8.0.34-0ubuntu0.22.04.1 for Linux on x86_64
```

Step 4.4: Install Git

```
# Install Git
sudo apt install -y git

# Verify installation
git --version
```

Why Git?

- Clone application code from repository
- Version control
- Pull updates
- Industry standard

Expected output:

```
git version 2.34.1
```

Step 4.5: Install PM2 (Production Process Manager)

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

# Verify installation
pm2 --version
```

What is PM2? PM2 is a production process manager for Node.js applications.

Key Features:

1. **Auto-restart**: Restarts app if it crashes
2. **Clustering**: Runs multiple instances (uses all CPU cores)
3. **Load balancing**: Distributes requests across instances
4. **Monitoring**: Real-time CPU, memory metrics
5. **Log management**: Centralized log collection
6. **Zero-downtime reload**: Updates without stopping service
7. **Startup scripts**: Auto-start on server boot

Why use PM2?

- Node.js process can crash (memory leak, exception, etc.)
- Manual restart not feasible in production
- Need high availability
- Industry standard for Node.js production

PM2 vs alternatives:

Feature	PM2	Forever	SystemD	Nodemon
Clustering	✅	✅	✅	✅
Monitoring	✅	✅	Basic	✅
Zero-downtime	✅	✅	✅	✅
Cross-platform	✅	✅	✅	✅
Production-ready	✅	⚠	✅	✅

Expected output:

```
[PM2] Successfully installed PM2
5.3.0
```

Step 4.6: Install Nginx

```
# Install Nginx
sudo apt install -y nginx

# Verify installation
nginx -v

# Check Nginx status
sudo systemctl status nginx
```

What is Nginx? Nginx (pronounced "engine-x") is a high-performance web server and reverse proxy.

Why use Nginx with Node.js?

1. **Reverse Proxy:**

   ```
   Internet → Nginx (Port 80/443) → Node.js (Port 8080)
   ```

   - Nginx receives requests on standard ports (80, 443)
   - Forwards to Node.js on port 8080
   - Node.js doesn't need root permissions

2. **SSL/TLS Termination:**

   - Nginx handles HTTPS encryption/decryption
   - Node.js receives plain HTTP
   - Offloads CPU-intensive cryptography

3. **Static File Serving:**

   - Nginx serves CSS, JS, images directly
   - Node.js only handles dynamic requests
   - Much faster than Node.js for static files

4. **Load Balancing:**

   ```
   Nginx → Node.js Instance 1
        → Node.js Instance 2
        → Node.js Instance 3
   ```

   - Distributes load across multiple instances
   - Horizontal scaling

5. **Security:**

   - Acts as a buffer between internet and app
   - Rate limiting
   - Request filtering
   - DDoS protection

6. **Performance:**

   - Nginx: C language, highly optimized
   - Handles 10,000+ concurrent connections
   - Low memory footprint

Nginx vs Alternatives:

Feature	Nginx	Apache	Node.js alone
Speed	Very Fast	Fast	Moderate
Static files	Excellent	Good	Poor
Concurrent connections	10,000+	1,000	100
Memory usage	Very Low	High	Moderate
Configuration	Simple	Complex	N/A
Reverse proxy	Excellent	Good	N/A

Expected output:

```
nginx version: nginx/1.18.0 (Ubuntu)
● nginx.service - A high performance web server
     Loaded: loaded
     Active: active (running)
```

Nginx is now running! You can verify by visiting your EC2 public IP in a browser. You should see the default Nginx page.

---

6. Application Deployment

Phase 5: Clone and Configure Application

Step 5.1: Create Application Directory

```
# Create directory for application
sudo mkdir -p /var/www/epicbooks

# Change ownership to ubuntu user
sudo chown -R ubuntu:ubuntu /var/www/epicbooks

# Navigate to directory
cd /var/www/epicbooks
```

Understanding the commands:

`sudo mkdir -p /var/www/epicbooks`:

- `mkdir`: Make directory
- `-p`: Create parent directories if they don't exist
- `/var/www`: Standard location for web applications in Linux
- **Why /var/www?** Convention; easy to remember and manage

`sudo chown -R ubuntu:ubuntu /var/www/epicbooks`:

- `chown`: Change ownership
- `-R`: Recursive (all files and subdirectories)
- `ubuntu:ubuntu`: user:group
- **Why?** So you can edit files without sudo

Step 5.2: Clone Repository

```
# Clone your repository
git clone https://github.com/230919bca-ctis-bit/EpicBooks.git .
```

Note the dot (.) at the end:

- `.`: Means "current directory"
- Without it: Creates a subdirectory named `EpicBooks`
- With it: Clones contents directly into current directory

If your repository is private:

```
# You'll be prompted for GitHub credentials
Username: your-github-username
Password: [use Personal Access Token, not password]
```

Expected output:

```
Cloning into '.'...
remote: Enumerating objects: 150, done.
remote: Counting objects: 100% (150/150), done.
remote: Compressing objects: 100% (95/95), done.
remote: Total 150 (delta 45), reused 150 (delta 45)
Receiving objects: 100% (150/150), 2.5 MiB | 8.0 MiB/s, done.
Resolving deltas: 100% (45/45), done.
```

Verify clone:

```
ls -la
```

You should see:

```
drwxrwxr-x  5 ubuntu ubuntu 4096 Dec 15 10:00 .
drwxr-xr-x  3 root   root   4096 Dec 15 09:55 ..
drwxrwxr-x  8 ubuntu ubuntu 4096 Dec 15 10:00 .git
-rw-rw-r--  1 ubuntu ubuntu  450 Dec 15 10:00 .gitignore
-rw-rw-r--  1 ubuntu ubuntu 1234 Dec 15 10:00 package.json
-rw-rw-r--  1 ubuntu ubuntu 2345 Dec 15 10:00 README.md
drwxrwxr-x  2 ubuntu ubuntu 4096 Dec 15 10:00 src
...
```

Step 5.3: Install Dependencies

```
# Install all npm packages
npm install
```

What happens:

1. Reads `package.json` file
2. Downloads all dependencies
3. Installs in `node_modules` directory
4. Creates `package-lock.json` (lock versions)

Expected output:

```
added 145 packages, and audited 146 packages in 15s

found 0 vulnerabilities
```

If you see vulnerabilities:

```
npm audit fix
```

Common dependencies in package.json:

```
{
  "dependencies": {
    "express": "^4.18.2",        // Web framework
    "ejs": "^3.1.9",             // Template engine
    "sequelize": "^6.32.1",      // ORM for MySQL
    "mysql2": "^3.5.0",          // MySQL driver
    "dotenv": "^16.3.1",         // Environment variables
    "bcrypt": "^5.1.0",          // Password hashing
    "express-session": "^1.17.3", // Session management
    "body-parser": "^1.20.2"     // Parse request bodies
  },
  "devDependencies": {
    "nodemon": "^3.0.1"          // Auto-restart during development
  }
}
```

Step 5.4: Configure Environment Variables

What are Environment Variables?

- Configuration settings stored outside code
- Different values for development/production
- Keeps sensitive data (passwords) out of Git

Why use .env file?

- Security: Passwords not in code repository
- Flexibility: Change config without editing code
- Portability: Same code runs in different environments

Create .env file:

```
nano .env
```

Add configuration:

```
# Application Configuration
NODE_ENV=production
PORT=8080

# Database Configuration
DB_HOST=epicbooks-db.abcd1234.us-east-1.rds.amazonaws.com
DB_PORT=3306
DB_NAME=epicbooks
DB_USER=admin
DB_PASSWORD=YourSecurePassword123!

# Session Secret (generate random string)
SESSION_SECRET=your-very-long-random-secret-key-here

# Application URL
APP_URL=http://your-ec2-public-ip
```

How to fill this out:

1. **NODE_ENV=production**
    - Tells app to run in production mode
    - Disables verbose logging
    - Enables performance optimizations
2. **PORT=8080**
    - Port where Node.js listens
    - Nginx will forward requests to this port
    - Any port > 1024 works (doesn't need root)
3. **DB_HOST**
    - Use RDS endpoint from Phase 2, Step 2.3
    - Example: `epicbooks-db.abcd1234.us-east-1.rds.amazonaws.com`
4. **DB_PORT=3306**
    - Standard MySQL port
    - Don't change unless you customized RDS
5. **DB_NAME=epicbooks**
    - Database name you specified in RDS creation
6. **DB_USER=admin**
    - Master username from RDS creation
7. **DB_PASSWORD**
    - Master password you set for RDS
    - Must match exactly
8. **SESSION_SECRET**
    - Used to encrypt session cookies
    - Should be long and random
    - Generate with: `openssl rand -base64 32`
9. **APP_URL**
    - Your EC2 public IP address
    - Example: `http://54.123.45.67`
    - Used for redirects and link generation

Generate secure session secret:

```
openssl rand -base64 32
```

Output example:

```
k7JhF3mPqR8wT5vN2xL9cS4bA6dE1fG0
```

Save the file:

- Press `Ctrl + X`
- Press `Y` (yes)
- Press `Enter` (confirm filename)

Set proper permissions:

```
chmod 600 .env
```

Why chmod 600?

- `6` : Owner can read and write
- `0` : Group has no access
- `0` : Others have no access
- Security: Only you can read the password

Verify configuration:

```
cat .env
```

Test database connection:

```
mysql -h epicbooks-db.abcd1234.us-east-1.rds.amazonaws.com -u admin -p
```

Enter password when prompted.

If connection successful:

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| epicbooks          |
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0.01 sec)

mysql> exit
```

If connection fails:

- Check RDS security group allows EC2 security group
- Verify endpoint spelling
- Verify username and password
- Check RDS is in "Available" state

Phase 6: Database Setup

Step 6.1: Understanding Database Schema

What is a schema?

- Blueprint of database structure
- Defines tables, columns, relationships
- Like a plan before building a house

Common tables in EpicBooks:

Users Table:

```
CREATE TABLE users (
  id INT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(50) UNIQUE NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  password VARCHAR(255) NOT NULL,  -- Hashed
  role ENUM('user', 'admin') DEFAULT 'user',
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Books Table:

```
CREATE TABLE books (
  id INT PRIMARY KEY AUTO_INCREMENT,
  title VARCHAR(200) NOT NULL,
  author VARCHAR(100) NOT NULL,
  isbn VARCHAR(20) UNIQUE,
  description TEXT,
  price DECIMAL(10, 2) NOT NULL,
  stock_quantity INT DEFAULT 0,
  image_url VARCHAR(255),
  category VARCHAR(50),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Orders Table:

```
CREATE TABLE orders (
  id INT PRIMARY KEY AUTO_INCREMENT,
  user_id INT NOT NULL,
  total_amount DECIMAL(10, 2) NOT NULL,
  status ENUM('pending', 'processing', 'shipped', 'delivered') DEFAULT 'pending',
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Order Items Table:

```
CREATE TABLE order_items (
  id INT PRIMARY KEY AUTO_INCREMENT,
  order_id INT NOT NULL,
  book_id INT NOT NULL,
  quantity INT NOT NULL,
  price DECIMAL(10, 2) NOT NULL,
  FOREIGN KEY (order_id) REFERENCES orders(id),
  FOREIGN KEY (book_id) REFERENCES books(id)
);
```

Step 6.2: Run Migrations with Sequelize

What are Migrations?

- Version control for database schema
- Scripts that modify database structure
- Can be applied or rolled back
- Track changes over time

Why use Migrations?

- Team collaboration: Everyone has same schema
- Deployment: Automatically update production database
- Rollback: Undo changes if needed
- Documentation: History of schema changes

Sequelize Migration Files:

Typically in `database/migrations/` directory:

```
database/
└── migrations/
    ├── 20231215120000-create-users.js
    ├── 20231215120001-create-books.js
    ├── 20231215120002-create-orders.js
    └── 20231215120003-create-order-items.js
```

Migration file structure:

```
// 20231215120000-create-users.js
module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.createTable('users', {
      id: {
        type: Sequelize.INTEGER,
        primaryKey: true,
        autoIncrement: true
      },
      username: {
        type: Sequelize.STRING(50),
        allowNull: false,
        unique: true
      },
      // ... more columns
    });
  },

  down: async (queryInterface, Sequelize) => {
    await queryInterface.dropTable('users');
  }
};
```

Run migrations:

```
# If project uses Sequelize CLI
npx sequelize-cli db:migrate
```

Or if custom migration script:

```
npm run migrate
```

Or if manual SQL file:

```
mysql -h $DB_HOST -u $DB_USER -p$DB_PASSWORD $DB_NAME < database/schema.sql
```

Expected output:

```
Sequelize CLI [Node: 18.17.0, CLI: 6.6.1, ORM: 6.32.1]

== 20231215120000-create-users: migrating =======
== 20231215120000-create-users: migrated (0.234s)

== 20231215120001-create-books: migrating =======
== 20231215120001-create-books: migrated (0.187s)

== 20231215120002-create-orders: migrating ======
== 20231215120002-create-orders: migrated (0.156s)

== 20231215120003-create-order-items: migrating =
== 20231215120003-create-order-items: migrated (0.145s)
```

Verify tables created:

```
mysql -h $DB_HOST -u $DB_USER -p$DB_PASSWORD -e "USE epicbooks; SHOW TABLES;"
```

Expected output:

```
+---------------------+
| Tables_in_epicbooks |
+---------------------+
| books               |
| order_items         |
| orders              |
| users               |
+---------------------+
```

Step 6.3: Seed Initial Data

What is Seeding?

- Populating database with initial data
- Sample data for testing
- Default admin account
- Essential configuration

Why seed data?

- Can't test app with empty database
- Need admin user to access admin panel
- Sample books to display on homepage
- Pre-fill categories, settings, etc.

Seeder file example:

```
// database/seeders/20231215-demo-books.js
module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.bulkInsert('books', [
      {
        title: 'The Great Gatsby',
        author: 'F. Scott Fitzgerald',
        isbn: '978-0743273565',
        description: 'A classic novel of the Jazz Age',
        price: 14.99,
        stock_quantity: 50,
        category: 'Fiction',
        created_at: new Date()
      },
      {
        title: 'To Kill a Mockingbird',
        author: 'Harper Lee',
        isbn: '978-0061120084',
        description: 'A gripping tale of racial injustice',
        price: 12.99,
        stock_quantity: 40,
        category: 'Fiction',
        created_at: new Date()
      }
      // ... more books
    ]);

    // Create admin user
    const bcrypt = require('bcrypt');
    const hashedPassword = await bcrypt.hash('admin123', 10);

    await queryInterface.bulkInsert('users', [{
      username: 'admin',
      email: 'admin@epicbooks.com',
      password: hashedPassword,
      role: 'admin',
      created_at: new Date()
    }]);
  },

  down: async (queryInterface, Sequelize) => {
    await queryInterface.bulkDelete('books', null, {});
    await queryInterface.bulkDelete('users', null, {});
  }
};
```

Run seeders:

```
npx sequelize-cli db:seed:all
```

Or custom seed script:

```
npm run seed
```

Or manual SQL:

```
mysql -h $DB_HOST -u $DB_USER -p$DB_PASSWORD $DB_NAME < database/seeds/initial_data.sql
```

Expected output:

```
Sequelize CLI [Node: 18.17.0, CLI: 6.6.1, ORM: 6.32.1]

== 20231215-demo-books: migrating =======
== 20231215-demo-books: migrated (0.089s)
```

Verify seed data:

```
mysql -h $DB_HOST -u $DB_USER -p$DB_PASSWORD -e "USE epicbooks; SELECT COUNT(*) FROM books;"
```

Expected output:

```
+----------+
| COUNT(*) |
+----------+
|       25 |
+----------+
```

Test admin login:

```
Username: admin
Password: admin123
```

IMPORTANT: Change admin password after first login!

Phase 7: Testing Application

Step 7.1: Start Application in Development Mode

```
# Start with npm
npm start

# Or if nodemon configured for development
npm run dev
```

Expected output:

```
> epicbooks@1.0.0 start
> node src/index.js

[INFO] EpicBooks application starting...
[INFO] Environment: production
[INFO] Connecting to database...
[INFO] Database connected successfully
[INFO] Server listening on port 8080
[INFO] Application ready at http://localhost:8080
```

If errors occur, common issues:

Error: Cannot find module

```
Error: Cannot find module 'express'
```

Solution:

```
npm install
```

Error: connect ECONNREFUSED

```
Error: connect ECONNREFUSED <db-endpoint>:3306
```

Solution:

- Check RDS is running

- Verify security group rules
- Check .env database credentials

Error: Access denied for user

```
Error: Access denied for user 'admin'@'<ip>' (using password: YES)
```

Solution:

- Verify DB_PASSWORD in .env
- Check RDS master password
- Ensure no extra spaces in .env

Error: Port already in use

```
Error: listen EADDRINUSE: address already in use :::8080
```

Solution:

```
# Find process using port 8080
lsof -i :8080

# Kill the process
kill -9 <PID>
```

Step 7.2: Test Application Locally

Open a new SSH session (keep app running in first session):

```
# Test if application responds
curl http://localhost:8080
```

Expected output:

```
<!DOCTYPE html>
<html>
<head>
  <title>EpicBooks - Home</title>
  ...
</head>
<body>
  <h1>Welcome to EpicBooks</h1>
  ...
</body>
</html>
```

Test specific routes:

```
# Health check
curl http://localhost:8080/health

# Books API
curl http://localhost:8080/api/books

# Login page
curl http://localhost:8080/login
```

Step 7.3: Test from Browser

Open your web browser:

```
http://<your-ec2-public-ip>:8080
```

Example:

```
http://54.123.45.67:8080
```

What you should see:

- Homepage with book listings
- Navigation menu
- Search functionality
- Login/Register links

Test these pages:

1. **Homepage**: `http://<ip>:8080/`
2. **Books**: `http://<ip>:8080/books`
3. **Login**: `http://<ip>:8080/login`
4. **Register**: `http://<ip>:8080/register`
5. **Admin**: `http://<ip>:8080/admin` (login as admin first)

If page doesn't load:

1. Check EC2 security group allows port 8080 from 0.0.0.0/0
2. Verify application is running (check SSH session)
3. Try accessing from EC2 terminal: `curl http://localhost:8080`
4. Check application logs for errors

Step 7.4: Check Application Logs

View logs in terminal:

```
# If using console.log statements
# Logs appear in SSH terminal where app is running

# If using file logging (check package.json for log location)
tail -f logs/application.log
```

Common log entries:

```
[2023-12-15 10:30:00] INFO: Server started on port 8080
[2023-12-15 10:30:15] INFO: GET / 200 45ms
[2023-12-15 10:30:17] INFO: GET /books 200 23ms
[2023-12-15 10:30:20] INFO: POST /login 200 156ms
[2023-12-15 10:30:25] INFO: GET /admin 200 12ms
```

Stop application:

```
# Press Ctrl + C in SSH terminal where app is running
```

---

7. Production Configuration

Phase 8: Configure Nginx Reverse Proxy

What is a Reverse Proxy?

Normal (Forward) Proxy:

```
[Your Computer] → [Proxy] → [Internet] → [Website]
```

- Hides your IP address
- Used for privacy

Reverse Proxy:

```
[Internet] → [Nginx] → [Your Application]
```

- Hides your application server
- Load balancing, SSL termination, caching

Why Nginx as Reverse Proxy?

Without Nginx:

```
[Browser] → Port 8080 → [Node.js]
```

Problems:

- Users must type `:8080` in URL (ugly)
- Can't use standard ports 80/443 without root
- No SSL/HTTPS
- Node.js handles everything (inefficient)
- Single point of failure

With Nginx:

```
[Browser] → Port 80 → [Nginx] → Port 8080 → [Node.js]
```

Benefits:

- Clean URLs (no port number)
- SSL termination
- Static file caching
- Load balancing
- DDoS protection
- Multiple apps on same server

Step 8.1: Remove Default Nginx Configuration

```
# Remove default configuration
sudo rm /etc/nginx/sites-enabled/default
```

Why remove default?

- Conflicts with our configuration
- Shows "Welcome to Nginx" page
- Uses port 80 (we need it)

Step 8.2: Create Nginx Configuration

Create new config file:

```
sudo nano /etc/nginx/sites-available/epicbooks
```

Add this configuration:

```
# Define upstream (Node.js application)
upstream nodejs_backend {
    # Node.js runs on localhost:8080
    server 127.0.0.1:8080;

    # Optional: Multiple instances for load balancing
    # server 127.0.0.1:8081;
    # server 127.0.0.1:8082;

    # Load balancing method: least connections
    least_conn;
}

# HTTP server (port 80)
server {
    # Listen on port 80 (standard HTTP)
    listen 80;
    listen [::]:80;  # IPv6 support

    # Server name (use your domain or EC2 public IP)
    server_name _;  # Matches any hostname

    # Maximum upload size (for book cover images)
    client_max_body_size 10M;

    # Root directory (not used, just for reference)
    root /var/www/epicbooks/public;
```

```
    # Logging
    access_log /var/log/nginx/epicbooks_access.log;
    error_log /var/log/nginx/epicbooks_error.log;

    # Serve static files directly (faster than Node.js)
    location /static/ {
        alias /var/www/epicbooks/public/;
        expires 30d;  # Cache for 30 days
        add_header Cache-Control "public, immutable";
    }

    # Serve uploaded images
    location /uploads/ {
        alias /var/www/epicbooks/uploads/;
        expires 7d;
        add_header Cache-Control "public";
    }

    # Proxy all other requests to Node.js
    location / {
        # Forward to upstream (Node.js)
        proxy_pass http://nodejs_backend;

        # Preserve original request information
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;

        # Pass real IP address to Node.js
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # Timeouts (prevent hanging requests)
        proxy_connect_timeout 60s;
        proxy_send_timeout 60s;
        proxy_read_timeout 60s;
    }

    # Health check endpoint (doesn't log)
    location /health {
        proxy_pass http://nodejs_backend;
        access_log off;
    }
}
```

Understanding the configuration:

`upstream nodejs_backend`:

- Defines backend servers
- Can have multiple servers for load balancing
- `least_conn`: Sends request to server with fewest connections

`listen 80`:

- Standard HTTP port
- No need to specify port in URL

`server_name _`:

- Underscore matches any hostname
- Use your domain in production: `server_name epicbooks.com www.epicbooks.com;`

`client_max_body_size 10M`:

- Maximum upload size
- Needed for book cover images
- Default is 1M (too small)

`location /static/` :

- Nginx serves CSS, JS, images directly
- Much faster than Node.js
- `expires 30d` : Browser caches for 30 days
- Reduces server load

`location /` :

- All other requests go to Node.js
- `proxy_pass` : Forwards to localhost:8080
- `proxy_set_header` : Passes metadata to Node.js

`proxy_set_header X-Real-IP` :

- Node.js sees user's real IP, not 127.0.0.1
- Important for logging, rate limiting, geolocation

`proxy_connect_timeout 60s` :

- How long to wait for Node.js to accept connection
- Prevents indefinite waiting

Save the file:

- Press `Ctrl + X`
- Press `Y`
- Press `Enter`

Step 8.3: Enable Configuration

```
# Create symbolic link to sites-enabled
sudo ln -s /etc/nginx/sites-available/epicbooks /etc/nginx/sites-enabled/

# Test configuration for syntax errors
sudo nginx -t
```

Expected output:

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

If errors:

```
nginx: [emerg] unexpected "}" in /etc/nginx/sites-available/epicbooks:45
```

- Check for typos
- Ensure all brackets match
- Look at line number in error

Reload Nginx:

```
sudo systemctl reload nginx
```

Verify Nginx status:

```
sudo systemctl status nginx
```

Expected output:

```
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2023-12-15 10:45:00 UTC; 2min 30s ago
```

Step 8.4: Test Nginx Configuration

Start Node.js application (if not running):

```
cd /var/www/epicbooks
npm start
```

Open browser (without port 8080):

```
http://<your-ec2-public-ip>
```

Example:

```
http://54.123.45.67
```

You should see:

- Same application as before
- But without `:8080` in URL
- Cleaner, more professional

Test static files:

```
http://<your-ec2-public-ip>/static/css/style.css
```

Should load instantly (served by Nginx, not Node.js).

Check Nginx logs:

```
# Access log (successful requests)
sudo tail -f /var/log/nginx/epicbooks_access.log

# Error log (problems)
sudo tail -f /var/log/nginx/epicbooks_error.log
```

Access log format:

```
54.123.45.67 - - [15/Dec/2023:10:50:00 +0000] "GET / HTTP/1.1" 200 1234 "-" "Mozilla/5.0..."
```

Explanation:

- `54.123.45.67` : Client IP address
- `[15/Dec/2023:10:50:00 +0000]` : Timestamp
- `"GET / HTTP/1.1"` : HTTP method and path
- `200` : Status code (success)
- `1234` : Response size in bytes
- `"Mozilla/5.0..."` : User agent (browser)

Phase 9: Run Application with PM2

Why PM2 in Production?

Current situation:

- Application runs in SSH terminal
- If you close terminal → app stops
- If app crashes → app stops forever
- No automatic restart
- Can't use multiple CPU cores

With PM2:

- App runs as background service
- Survives SSH disconnection
- Auto-restart on crash
- Cluster mode (multiple instances)
- Zero-downtime updates
- Built-in monitoring
- Log management

Step 9.1: Stop Current Application

If app is running in terminal:

```
# Press Ctrl + C to stop
```

Step 9.2: Start with PM2

```
# Navigate to project directory
cd /var/www/epicbooks

# Start application with PM2
pm2 start src/index.js --name epicbooks
```

Understanding the command:

- `pm2 start` : Start an application
- `src/index.js` : Entry point file
- `--name epicbooks` : Give it a recognizable name

Expected output:

```
[PM2] Starting /var/www/epicbooks/src/index.js in fork_mode (1 instance)
[PM2] Done.
┌────┬─────────────┬─────────────┬─────────┬─────────┬──────────┬────────┐
│ id │ name        │ namespace   │ version │ mode    │ pid      │ status │
├────┼─────────────┼─────────────┼─────────┼─────────┼──────────┼────────┤
│ 0  │ epicbooks   │ default     │ 1.0.0   │ fork    │ 12345    │ online │
└────┴─────────────┴─────────────┴─────────┴─────────┴──────────┴────────┘
```

Status meanings:

- `online` : Running normally
- `stopping` : Gracefully shutting down
- `stopped` : Not running
- `errored` : Crashed

Step 9.3: Configure PM2 for Production

Create PM2 ecosystem config:

```
nano ecosystem.config.js
```

Add configuration:

```
module.exports = {
  apps: [{
    // Application name
    name: 'epicbooks',

    // Script to start
    script: './src/index.js',

    // Number of instances
    // 0 = all CPU cores
    // max = all CPU cores
    // 1 = single instance
    instances: 'max',

    // Execution mode
    // 'cluster' = load balancing across instances
    // 'fork' = single process
    exec_mode: 'cluster',

    // Auto-restart on file changes (development only)
    watch: false,

    // Max memory before auto-restart (prevents memory leaks)
    max_memory_restart: '500M',

    // Environment variables
    env: {
      NODE_ENV: 'production',
      PORT: 8080
    },

    // Error log file
    error_file: '/var/www/epicbooks/logs/pm2-error.log',

    // Output log file
    out_file: '/var/www/epicbooks/logs/pm2-output.log',

    // Log date format
    log_date_format: 'YYYY-MM-DD HH:mm:ss Z',

    // Merge logs from all instances
    merge_logs: true,

    // Auto-restart on crash
    autorestart: true,

    // Max number of restarts in 1 minute
    max_restarts: 10,

    // Delay between restarts
    min_uptime: '10s',

    // Time to wait before killing process
    kill_timeout: 3000
  }]
};
```

Understanding cluster mode:

Single Instance (fork mode):

```
[Request 1] → Node.js Instance
[Request 2] → (waits)
[Request 3] → (waits)
```

- One request at a time
- Doesn't use all CPU cores
- Lower performance

Cluster Mode:

```
[Request 1] → Node.js Instance 1
[Request 2] → Node.js Instance 2
[Request 3] → Node.js Instance 3
[Request 4] → Node.js Instance 4
```

- Multiple instances (one per CPU core)
- Load balanced automatically
- Better performance
- High availability (if one crashes, others continue)

Create logs directory:

```
mkdir -p /var/www/epicbooks/logs
```

Save file and exit:

- Press `Ctrl + X`
- Press `Y`
- Press `Enter`

Step 9.4: Start with Ecosystem Config

```
# Stop previous PM2 process
pm2 stop all
pm2 delete all

# Start with ecosystem config
pm2 start ecosystem.config.js

# Save PM2 configuration
pm2 save

# Generate startup script (auto-start on server boot)
pm2 startup

# Copy and run the command shown in output
```

PM2 startup output example:

```
[PM2] You have to run this command as root. Execute the following command:
sudo env PATH=$PATH:/usr/bin /usr/lib/node_modules/pm2/bin/pm2 startup systemd -u ubuntu --hp /home/ubuntu
```

Copy and run that command:

```
sudo env PATH=$PATH:/usr/bin /usr/lib/node_modules/pm2/bin/pm2 startup systemd -u ubuntu --hp /home/ubuntu
```

What this does:

- Creates systemd service
- PM2 starts automatically on server reboot
- Application survives server crashes

Verify PM2 status:

```
pm2 status
```

Expected output:

```
┌─────┬──────────────┬─────────────┬─────────┬─────────┬─────────┬────────┐
| id  | name         | namespace   | version | mode    | pid     | status |
├─────┼──────────────┼─────────────┼─────────┼─────────┼─────────┼────────┤
└── 0   | epicbooks    | default     | 1.0.0   | cluster | 12345   | online |
| 1   | epicbooks    | default     | 1.0.0   | cluster | 12346   | online |
| 2   | epicbooks    | default     | 1.0.0   | cluster | 12347   | online |
| 3   | epicbooks    | default     | 1.0.0   | cluster | 12348   | online |
└─────┴──────────────┴─────────────┴─────────┴─────────┴─────────┴────────┘
```

t2.micro has 1 CPU core, so you'll see 1 instance. If you upgrade to larger instance type, you'll see more instances.

Step 9.5: PM2 Monitoring

View logs:

```
# All logs (combined)
pm2 logs

# Logs for specific app
pm2 logs epicbooks

# Only errors
pm2 logs --err

# Last 100 lines
pm2 logs --lines 100
```

Monitor in real-time:

```
pm2 monit
```

Shows:

- CPU usage per instance
- Memory usage per instance
- Log output in real-time

Detailed information:

```
pm2 show epicbooks
```

Output:

```
 Describing process with id 0 - name epicbooks
┌────────────────────┬──────────────────────────────────┐
| status             | online                           |
| name               | epicbooks                        |
| version            | 1.0.0                            |
| restarts           | 0                                |
| uptime             | 5m                               |
| script path        | /var/www/epicbooks/src/index.js  |
| script args        | N/A                              |
| error log path     | /var/www/epicbooks/logs/...      |
| out log path       | /var/www/epicbooks/logs/...      |
| pid path           | /home/ubuntu/.pm2/pids/...       |
| mode               | cluster_mode                     |
| node v8 arguments  | []                               |
| watch & reload     | ✘                                |
| interpreter        | node                             |
| restarts           | 0                                |
| unstable restarts  | 0                                |
| created at         | 2023-12-15T10:00:00.000Z         |
└────────────────────┴──────────────────────────────────┘
```

Useful PM2 commands:

```
# Restart application (graceful)
pm2 restart epicbooks

# Reload (zero-downtime restart)
pm2 reload epicbooks

# Stop application
pm2 stop epicbooks

# Delete from PM2
pm2 delete epicbooks

# Restart all apps
pm2 restart all

# View process list
pm2 list

# Clear logs
pm2 flush

# Install PM2 log rotation (prevents huge log files)
pm2 install pm2-logrotate
```

Phase 10: Final Testing

Step 10.1: Comprehensive Functional Testing

1. Homepage:

```
URL: http://<your-ec2-ip>
Expected: Book listings, search bar, navigation
```

2. Search Functionality:

```
Action: Enter book title in search
Expected: Filtered results
```

3. User Registration:

```
URL: http://<your-ec2-ip>/register
Action: Create new account
Expected: Success message, redirect to login
```

4. User Login:

```
URL: http://<your-ec2-ip>/login
Action: Login with created account
Expected: Redirect to homepage, user name displayed
```

5. Browse Books:

```
URL: http://<your-ec2-ip>/books
Expected: All books with details
```

6. View Book Details:

```
Action: Click on a book
Expected: Full details, "Add to Cart" button
```

7. Add to Cart:

```
Action: Click "Add to Cart"
Expected: Item added, cart count updates
```

8. View Cart:

```
URL: http://<your-ec2-ip>/cart
Expected: Cart items, quantities, total price
```

9. Checkout:

```
Action: Click "Proceed to Checkout"
Expected: Order confirmation
```

10. Admin Login:

```
URL: http://<your-ec2-ip>/admin/login
Credentials: admin / admin123
Expected: Admin dashboard
```

11. Admin - Manage Books:

```
Action: Add/edit/delete books
Expected: Changes reflected on site
```

12. Admin - View Orders:

```
Action: View order list
Expected: All orders with details
```

Step 10.2: Performance Testing

Load testing with Apache Bench (ab):

```
# Install Apache Bench
sudo apt install -y apache2-utils

# Test with 100 requests, 10 concurrent
ab -n 100 -c 10 http://localhost/

# Test with 1000 requests, 100 concurrent
ab -n 1000 -c 100 http://localhost/
```

Expected results (good performance):

```
Requests per second:    500.00 [#/sec] (mean)
Time per request:       2.000 [ms] (mean)
Time per request:       20.000 [ms] (mean, across all concurrent requests)
Transfer rate:          150.00 [Kbytes/sec] received

Percentage of the requests served within a certain time (ms)
  50%      2
  66%      3
  75%      4
  80%      5
  90%      8
  95%     10
  98%     15
  99%     20
 100%     50 (longest request)
```

If performance is poor:

- Enable PM2 cluster mode (already done)
- Add Nginx caching

- Optimize database queries
- Add CDN for static files

Step 10.3: Security Verification

1. Check Port Security:

```
# Check open ports
sudo netstat -tlnp
```

Should only see:

```
Port 22   (SSH)
Port 80   (HTTP)
Port 3306 (MySQL, but only localhost)
Port 8080 (Node.js, but only localhost)
```

2. Verify Database Access:

```
# Try connecting from outside (should fail)
mysql -h epicbooks-db.xxx.rds.amazonaws.com -u admin -p

# Try connecting from EC2 (should work)
mysql -h epicbooks-db.xxx.rds.amazonaws.com -u admin -p
```

3. Test SQL Injection Prevention:

```
Try entering in search: ' OR '1'='1
Expected: Sanitized, no SQL error
```

4. Test XSS Prevention:

```
Try entering in comment: <script>alert('XSS')</script>
Expected: Escaped, not executed
```

5. Check HTTPS Redirect (future):

```
Note: Currently HTTP only. Add SSL certificate later.
```

Step 10.4: Monitoring Setup

Set up PM2 web monitoring:

```
pm2 install pm2-server-monit
```

Check application health:

```
curl http://localhost/health
```

Expected response:

```
{
  "status": "healthy",
  "database": "connected",
  "uptime": 3600,
  "timestamp": "2023-12-15T11:00:00.000Z"
}
```

Set up log rotation:

```
pm2 install pm2-logrotate
pm2 set pm2-logrotate:max_size 10M
pm2 set pm2-logrotate:retain 7
```

This rotates logs:

- When they reach 10MB
- Keeps last 7 log files
- Prevents disk space issues

---

8. Troubleshooting

Common Issues and Solutions

Issue 1: Cannot Connect to EC2 via SSH

Symptoms:

```
ssh: connect to host 54.123.45.67 port 22: Connection timed out
```

Possible causes:

1. Security group doesn't allow your IP on port 22
2. Key pair permissions incorrect
3. Wrong IP address

Solutions:

```
# Check security group in AWS Console
# Ensure inbound rule: SSH (22) from Your IP

# Fix key permissions
chmod 400 epicbooks-key.pem

# Verify EC2 is running
# Check "Instance State" in AWS Console
```

Issue 2: Application Not Accessible from Browser

Symptoms:

```
This site can't be reached
```

Possible causes:

1. Security group doesn't allow port 80
2. Nginx not running
3. PM2 application stopped

Solutions:

```
# Check security group allows port 80

# Check Nginx status
sudo systemctl status nginx
sudo systemctl start nginx

# Check PM2 status
pm2 status
pm2 start ecosystem.config.js

# Check if port 80 is listening
sudo netstat -tlnp | grep :80
```

Issue 3: Database Connection Failed

Symptoms:

```
Error: connect ETIMEDOUT
Access denied for user 'admin'
```

Possible causes:

1. RDS security group doesn't allow EC2 security group
2. Wrong credentials in .env
3. RDS not in "Available" state

Solutions:

```
# Verify RDS security group
# Inbound: MySQL (3306) from EC2 security group

# Check .env file
cat .env
# Verify DB_HOST, DB_USER, DB_PASSWORD

# Test connection manually
mysql -h $DB_HOST -u $DB_USER -p$DB_PASSWORD

# Check RDS status in AWS Console
```

Issue 4: Static Files Not Loading

Symptoms:

```
CSS not applied
Images not showing
404 errors for /static/*
```

Possible causes:

1. Nginx configuration incorrect
2. Files in wrong directory
3. Permissions issue

Solutions:

```
# Check Nginx config
sudo nginx -t

# Verify static files directory
ls -la /var/www/epicbooks/public

# Fix permissions
sudo chown -R ubuntu:ubuntu /var/www/epicbooks

# Check Nginx error log
sudo tail -f /var/log/nginx/epicbooks_error.log
```

Issue 5: PM2 Application Keeps Restarting

Symptoms:

```
pm2 status
# Shows "errored" or high restart count
```

Possible causes:

1. Application crashes on startup
2. Database connection issues
3. Port already in use

Solutions:

```
# View error logs
pm2 logs epicbooks --err

# Check what's using port 8080
sudo lsof -i :8080

# Try starting without PM2 to see errors
cd /var/www/epicbooks
npm start

# Check .env configuration
cat .env
```

Issue 6: Out of Memory

Symptoms:

```
pm2 status
# Shows high memory usage
# Server becomes slow
```

Possible causes:

1. Memory leak in application
2. Too many PM2 instances for available RAM
3. Large file uploads

Solutions:

```
# Restart PM2
pm2 restart epicbooks

# Reduce PM2 instances
# Edit ecosystem.config.js
# Change instances: 'max' to instances: 1

# Check memory usage
free -h
pm2 monit

# Set max memory restart in ecosystem.config.js
max_memory_restart: '300M'
```

Issue 7: Nginx 502 Bad Gateway

Symptoms:

```
502 Bad Gateway
nginx/1.18.0
```

Possible causes:

1. Node.js application not running
2. Port mismatch in Nginx config
3. Firewall blocking localhost communication

Solutions:

```
# Check PM2 status
pm2 status
pm2 start epicbooks

# Test Node.js directly
curl http://localhost:8080

# Check Nginx upstream config
sudo nano /etc/nginx/sites-available/epicbooks
# Verify: proxy_pass http://127.0.0.1:8080;

# Restart Nginx
sudo systemctl restart nginx
```

Issue 8: Database Migrations Failed

Symptoms:

```
Error: Table 'epicbooks.users' doesn't exist
```

Possible causes:

1. Migrations not run
2. Wrong database name
3. Migration files missing

Solutions:

```
# Check database exists
mysql -h $DB_HOST -u $DB_USER -p$DB_PASSWORD -e "SHOW DATABASES;"

# Run migrations manually
cd /var/www/epicbooks
npx sequelize-cli db:migrate

# Check migration status
npx sequelize-cli db:migrate:status

# Verify tables created
mysql -h $DB_HOST -u $DB_USER -p$DB_PASSWORD -e "USE epicbooks; SHOW TABLES;"
```

---

9. Maintenance & Updates

Regular Maintenance Tasks

1. Update Application Code

```
# Navigate to project directory
cd /var/www/epicbooks

# Stop PM2
pm2 stop epicbooks

# Pull latest code from Git
git pull origin main

# Install new dependencies
npm install

# Run database migrations (if any)
npx sequelize-cli db:migrate

# Restart with PM2 (zero-downtime)
pm2 reload epicbooks

# Or restart (brief downtime)
pm2 restart epicbooks
```

2. Update System Packages

```
# Update package list
sudo apt update

# Upgrade packages
sudo apt upgrade -y

# Reboot if kernel updated
# PM2 will auto-start on boot
sudo reboot
```

3. Monitor Disk Space

```
# Check disk usage
df -h

# Check largest directories
du -sh /var/www/epicbooks/*

# Clean PM2 logs
pm2 flush

# Clean system logs
sudo journalctl --vacuum-time=7d
```

4. Database Backups

```
# Manual backup
mysqldump -h $DB_HOST -u $DB_USER -p$DB_PASSWORD $DB_NAME > backup_$(date +%Y%m%d).sql

# Automated backup (RDS)
# AWS RDS automatically backs up daily
# Restore from AWS Console if needed
```

5. Monitor Application Performance

```
# View PM2 dashboard
pm2 monit

# Check logs for errors
pm2 logs --err

# View Nginx access patterns
sudo tail -f /var/log/nginx/epicbooks_access.log
```

---

10. Next Steps & Improvements

Recommended Enhancements

1. Add SSL/HTTPS

Why:

- Security (encrypted data)
- SEO ranking boost
- User trust
- Required for payment processing

How:

```
# Install Certbot
sudo apt install -y certbot python3-certbot-nginx

# Get SSL certificate (free from Let's Encrypt)
sudo certbot --nginx -d yourdomain.com -d www.yourdomain.com

# Auto-renewal
sudo certbot renew --dry-run
```

2. Set Up Domain Name

Steps:

1. Buy domain from Namecheap, GoDaddy, etc.
2. Point domain to EC2 Elastic IP (static IP)
3. Update Nginx `server_name` directive
4. Add SSL certificate

3. Implement CDN (CloudFront)

Benefits:

- Faster content delivery worldwide
- Reduced server load
- Lower bandwidth costs
- Better user experience

4. Add Redis for Caching

Benefits:

- Faster response times
- Reduced database queries
- Session storage
- Rate limiting

Install:

```
sudo apt install -y redis-server
npm install redis
```

5. Implement CI/CD Pipeline

Tools:

- GitHub Actions

- Jenkins
- GitLab CI

Benefits:

- Automated testing
- Automated deployment
- Faster releases
- Fewer errors

6. Add Monitoring & Alerting

Tools:

- AWS CloudWatch
- Datadog
- New Relic
- Sentry (error tracking)

Monitor:

- Server metrics (CPU, memory, disk)
- Application errors
- Response times
- Database performance

7. Implement Load Balancer

When needed:

- Traffic exceeds single EC2 capacity
- Need high availability
- Want zero-downtime deployments

AWS Solution:

- Application Load Balancer (ALB)
- Multiple EC2 instances
- Auto Scaling Group

Conclusion

You now have a fully functional, production-ready EpicBooks application running on AWS with:

Secure Architecture

- VPC with public/private subnets
- Security groups controlling access
- Database isolated from internet

High Availability

- PM2 process management
- Auto-restart on crash
- Cluster mode for performance

Production Web Server

- Nginx reverse proxy
- Static file caching
- Load balancing ready

Professional Deployment

- Environment-based configuration
- Database migrations
- Structured logging

Scalability

- Easy to add more servers
- Database on managed RDS
- Ready for load balancer

Next learning topics:

- SSL/TLS certificates
- Domain configuration
- Continuous deployment
- Advanced monitoring
- Database optimization
- Caching strategies

Congratulations on deploying your application!