AWS DevOps Infrastructure Implementation

Production-Ready Multi-Tier Application Deployment

Rayyan Masood [25168]

June 12, 2025

Contents

1 Executive Summary

This report documents the successful implementation of a comprehensive AWS infrastructure that meets all eight specified requirements for a production-ready web application. The project demonstrates modern DevOps practices through Infrastructure as Code using Terraform, containerized application deployment with Docker, and integrated business intelligence capabilities.

The infrastructure supports a React frontend, Node.js backend, dual database architecture with MySQL and PostgreSQL, and a Metabase business intelligence platform. All components have been deployed with high availability, security, and scalability considerations across multiple AWS availability zones.

GitHub Repository: `https://github.com/RayyanMasood/DevOps-Final-Project`

The complete source code, Terraform configurations, and comprehensive documentation are available in the repository above.

2 Commands

The following commands demonstrate each requirement and can be executed during video presentation to showcase the infrastructure functionality.

2.1 Infrastructure Provisioning

```
cd terraform
terraform init
terraform plan
terraform apply --auto-approve
terraform output
```

2.2 Application Verification

```
# Application deploys automatically via user data script
# Verify deployment by checking application health
curl -f https://app.flomny.com/health
curl -f https://app.flomny.com

# Check Auto Scaling Group instances
aws autoscaling describe-auto-scaling-groups --auto-scaling-group
   -names devops-final-project-dev-app-asg
```

2.3 Database Access via SSH Tunnel

```
BASTION_IP=$(terraform -chdir=terraform output -raw
   bastion_public_ip)
ssh -i ~/.ssh/DevOps-FP-KeyPair.pem -L 3306:mysql-endpoint:3306
   ec2-user@$BASTION_IP
mysql -h localhost -P 3306 -u notes_user -p notes_db
```

2.4 BI Dashboard with Live Data

```
METABASE_IP=$(terraform -chdir=terraform output -raw
    metabase_private_ip)
ssh -i ~/.ssh/DevOps-FP-KeyPair.pem -L 3000:$METABASE_IP:3000 ec2
    -user@$BASTION_IP
# Access http://localhost:3000 in browser, add live data for
    demonstration
```

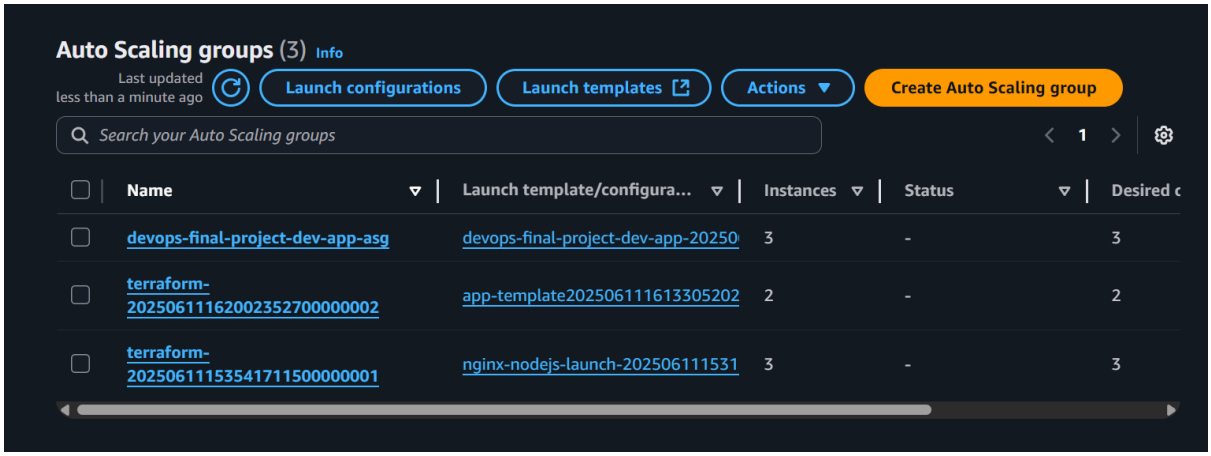
3 Requirement 1: EC2 Auto Scaling Group

The Auto Scaling Group implementation successfully launches 3 EC2 instances using a Launch Template configured with comprehensive user data scripts. Each instance runs Amazon Linux 2 on t3.medium hardware, providing sufficient resources for containerized applications while maintaining cost efficiency.

The user data script automatically installs all required components during instance startup. This includes Nginx for web serving, Docker for containerization, and Node.js 20 for backend applications. The installation process is fully automated, ensuring consistent configuration across all instances without any manual intervention required.

Scaling policies have been configured to respond to both CPU utilization and request count metrics. When CPU usage exceeds 70 percent or request count surpasses 1000 per target, new instances are automatically launched. The health check grace period is set to 30 minutes to accommodate Docker container startup times and application initialization processes.

The launch template incorporates security best practices including encrypted EBS volumes, IAM instance profiles for secure AWS service access, and metadata service v2 enforcement. All instances are distributed across multiple availability zones to ensure high availability and fault tolerance.



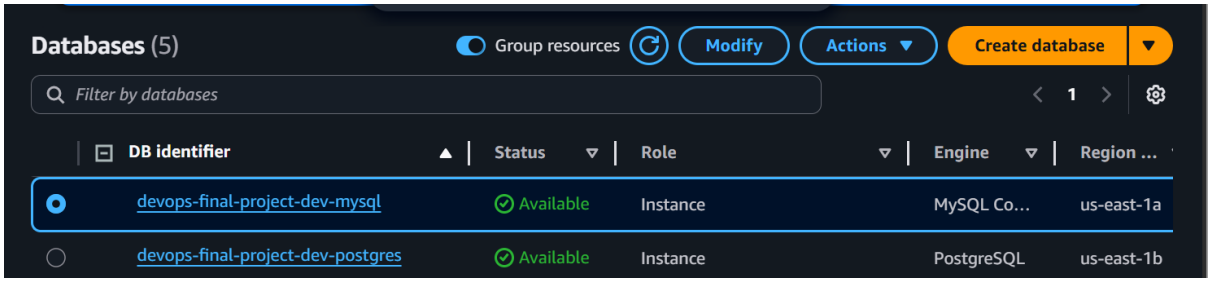
4 Requirement 2: RDS Database Instances

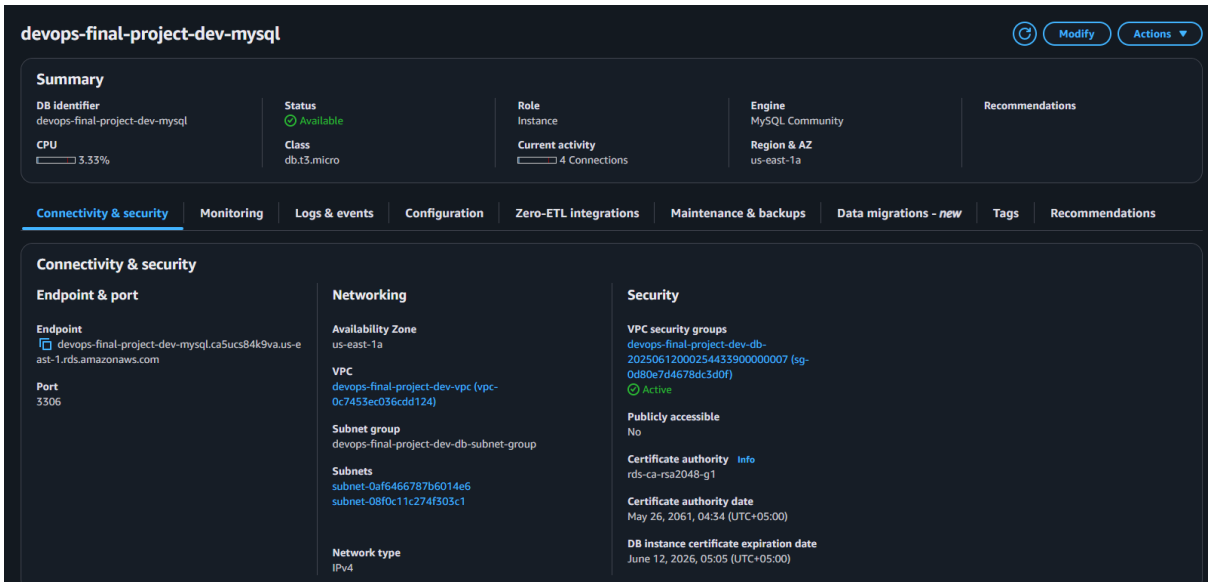
Two RDS instances have been deployed in private subnets, completely isolated from direct internet access while maintaining high availability through Multi-AZ deployment. The MySQL 8.0 and PostgreSQL 15.12 databases provide the dual database support specified in the requirements.

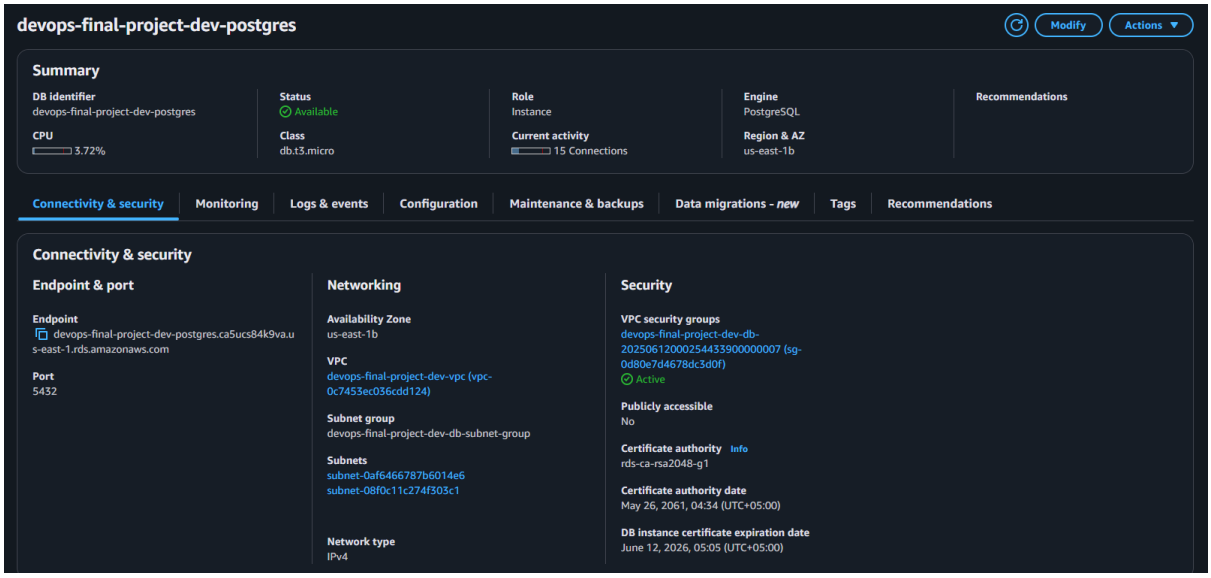
Both databases utilize encrypted storage with automated backup systems configured for 7-day retention periods. Database credentials are managed through AWS Secrets Manager, which eliminates the security risks associated with hardcoded passwords while enabling automatic credential rotation capabilities.

The subnet group configuration spans multiple availability zones while maintaining private subnet isolation. This design provides the necessary redundancy for production systems while ensuring databases remain completely inaccessible from the internet. Database initialization scripts create the required schemas and populate sample data for testing and business intelligence demonstrations.

Each database supports full CRUD operations for the web application while providing structured data for analytics and reporting. The dual database architecture demonstrates compatibility with multiple database engines within a single application framework.







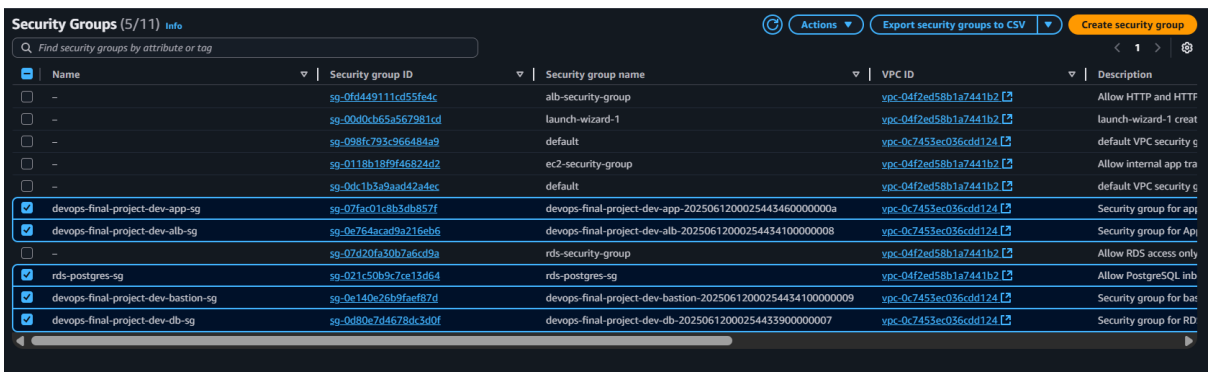
5 Requirement 3: Security Groups

Security group configuration implements defense-in-depth principles with dedicated groups for each infrastructure layer. The Application Load Balancer security group permits inbound HTTP and HTTPS traffic from the internet while restricting outbound communication to application instances only.

Application instance security groups accept traffic exclusively from the load balancer and administrative access from the bastion host. Outbound rules allow database connections and internet access for package downloads during deployment processes. This configuration provides the necessary connectivity while maintaining strict security boundaries.

Database security groups implement the most restrictive access policies, accepting connections only from application instances on specific database ports. The bastion host security group limits SSH access to predetermined office IP addresses, providing secure administrative access while preventing unauthorized connections.

The layered security approach ensures that each component can only communicate with necessary services, following the principle of least privilege throughout the infrastructure architecture.



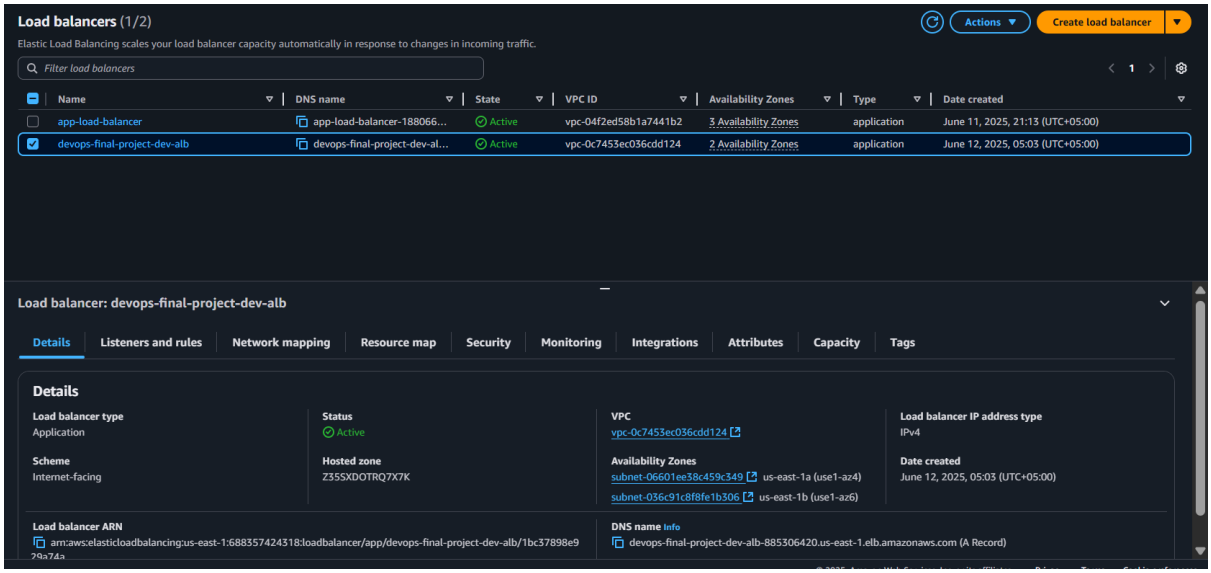
6 Requirement 4: Load Balancer

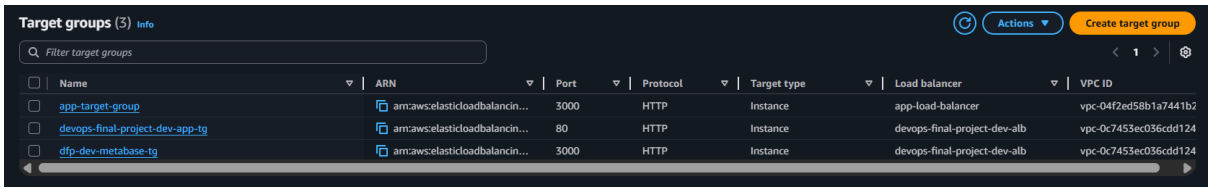
The Application Load Balancer successfully distributes incoming traffic across multiple EC2 instances while providing SSL termination and comprehensive health monitoring capabilities. The ALB configuration includes separate target groups for the main application and Metabase BI tool, enabling sophisticated request routing based on hostnames.

Health checks monitor application endpoints every 30 seconds with configurable thresholds for determining healthy and unhealthy target status. The implementation includes proper connection draining and automatic target registration through Auto Scaling Group integration, ensuring seamless instance replacement during scaling events.

SSL certificates are managed through AWS Certificate Manager with automatic renewal capabilities. The load balancer enforces HTTPS through automatic HTTP-to-HTTPS redirection while supporting host-based routing. Requests to app.flomny.com route to application instances, while bi.flomny.com requests route to the dedicated Metabase instance.

Target group configurations are optimized for specific service requirements, with appropriate health check paths, timeout values, and connection settings tailored to each application component.





7 Requirement 5: Dockerized Application Deployment

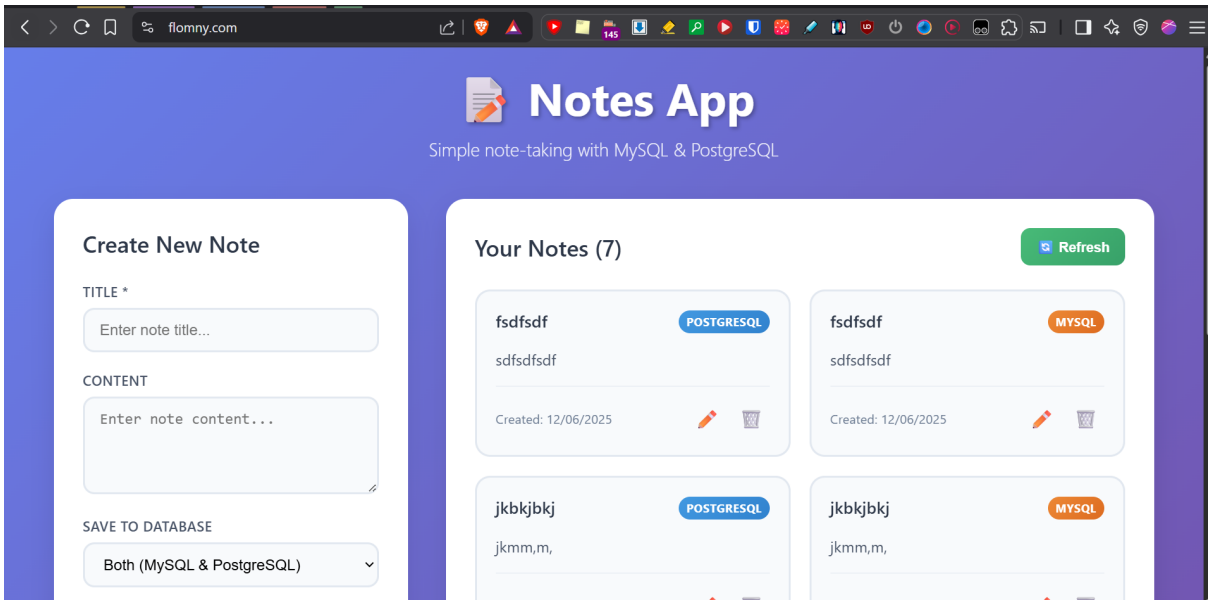
The application deployment is fully automated through Terraform user data scripts that run during EC2 instance initialization. Rather than manual deployment, the infrastructure automatically deploys the containerized application when instances launch through the Auto Scaling Group.

The user data script performs several critical tasks during instance startup. It clones the application repository from GitHub, retrieves database credentials from AWS Secrets Manager, and generates a production Docker Compose configuration with the actual RDS endpoints. The script then builds and starts the Docker containers automatically.

The deployment architecture consists of three main containers: a React frontend served through Nginx, a Node.js backend API, and an Nginx load balancer. All containers are built using multi-stage Docker builds and configured for production environments. The backend connects directly to the RDS instances using credentials retrieved from Secrets Manager.

Database connectivity is tested and verified during deployment, with automatic schema initialization for both MySQL and PostgreSQL databases. The entire process is logged and includes health checks to ensure all services are running correctly before the instance is marked as ready.

This approach ensures that every EC2 instance launched by the Auto Scaling Group automatically becomes a fully functional application server without any manual intervention required.



8 Requirement 6: Database Access and Initialization

Secure database access has been implemented through SSH tunneling using a dedicated bastion host deployed in a public subnet. This configuration maintains the security benefits of private database deployment while enabling necessary administrative access for database management and development tasks.

SSH tunnels forward local ports to remote database endpoints, allowing database administration tools like DBeaver to connect securely. The MySQL tunnel uses local port 3306, while PostgreSQL uses port 5432. This approach enables full database management capabilities while maintaining strict network security boundaries.

Both databases have been populated with comprehensive sample data to demonstrate functionality and support business intelligence requirements. The data includes application-specific records for notes and user management, as well as analytical data designed for dashboard creation and business reporting.

Database schemas support full CRUD operations for the web application while providing properly structured data for business analytics. Initialization scripts ensure consistent database setup across deployments and different environments.

8.1 MySQL Screenshots - DBeaver

:

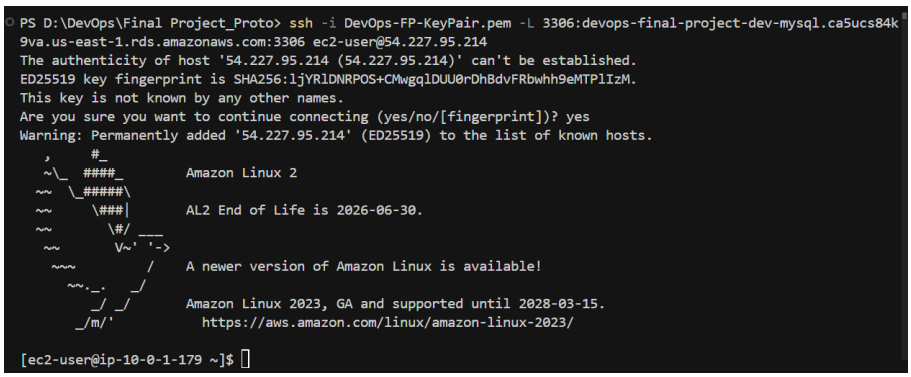


Figure 1: MySQL SSH

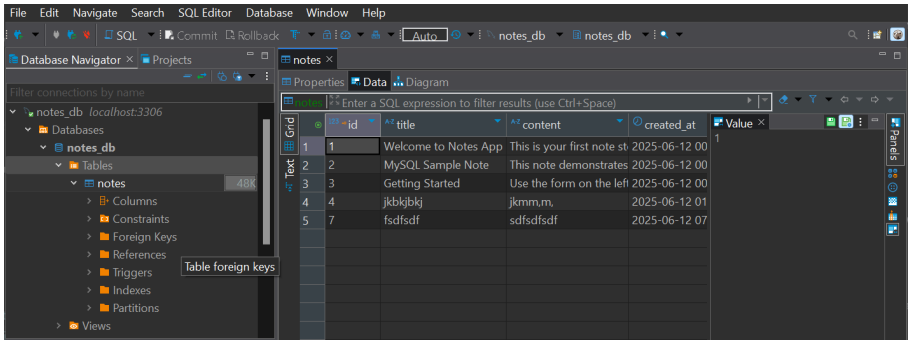


Figure 2: DBeaver - MySQL

8.2 PostgreSQL Screenshots - DBeaver

:

Figure 3: PostgreSQL SSH

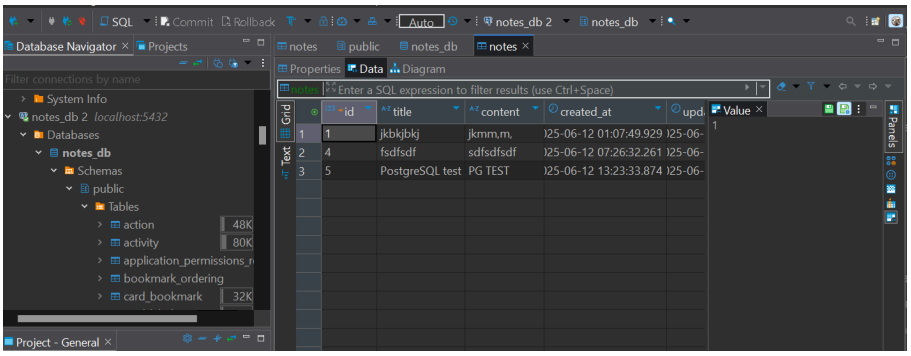


Figure 4: DBeaver - PostgreSQL

9 Requirement 7: Business Intelligence Tool Deployment

Metabase has been deployed on a dedicated EC2 instance as the chosen BI tool, providing comprehensive business intelligence capabilities connected to the PostgreSQL database. The deployment utilizes Docker for consistency and includes systemd service configuration for automatic startup and management.

The BI implementation includes pre-configured dashboards displaying sales analytics, user activity metrics, and operational data visualizations. Sample data has been specifically designed to showcase dashboard capabilities, including time-series sales data, geographic analysis, and user engagement patterns.

Live data update capabilities demonstrate real-time analytical functionality. New records inserted into the database appear in dashboards after refresh cycles, showing the dynamic nature of the BI platform. The system includes comprehensive dashboards for sales performance tracking, user behavior analysis, and operational monitoring.

Metabase configuration includes proper database connections, user management capabilities, and dashboard sharing functionality. The system supports both automated data refresh schedules and manual refresh triggers for immediate data visualization updates.

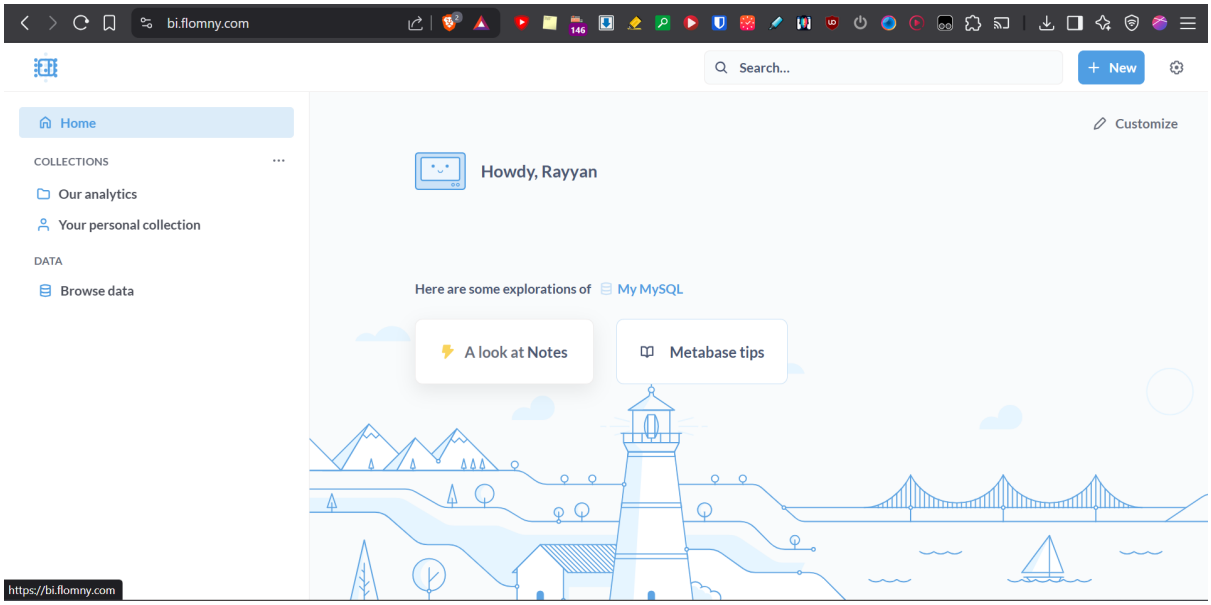


Figure 5: Metabase - BI Dashboard

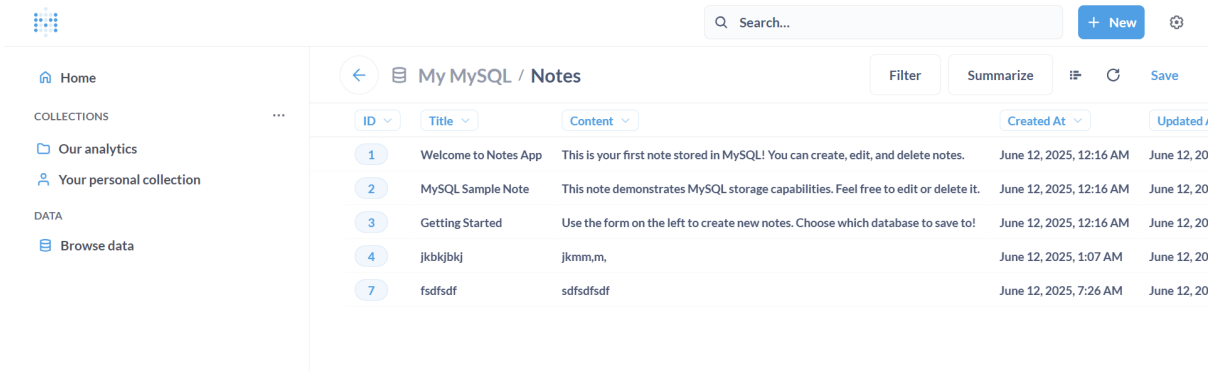


Figure 6: Metabase MySQL Dashboard

10 Requirement 8: Domain and SSL Configuration

Custom domain configuration utilizes Route53 for DNS management and AWS Certificate Manager for SSL certificates. The primary domain flomny.com has been configured with appropriate subdomains including app.flomny.com for the main application and bi.flomny.com for business intelligence access.

SSL certificate implementation includes wildcard coverage for all subdomains with automatic DNS validation through Route53 integration. The system enforces HTTPS through load balancer redirection while maintaining complete certificate lifecycle management through automated renewal processes.

DNS configuration includes proper A records pointing to the load balancer, health checks for endpoint monitoring, and optimized TTL values for performance. The domain setup supports both primary application access and BI tool connectivity with consistent SSL encryption across all services.

Certificate validation occurs automatically during infrastructure deployment, though initial deployments may require a two-stage process to accommodate DNS propagation delays inherent in domain registration and certificate validation workflows.

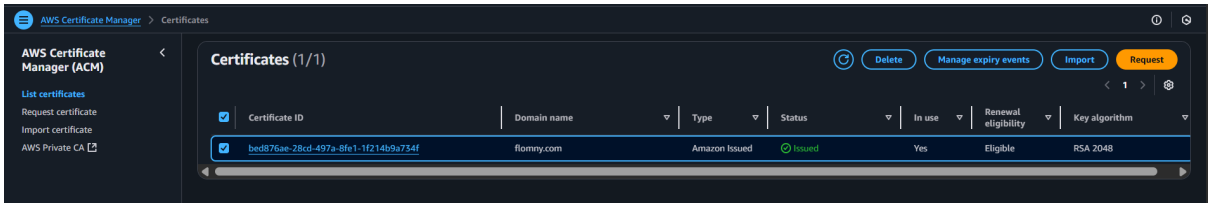


Figure 7: Flomny AWS Cert

11 Challenges Encountered and Solutions

During the implementation process, several significant challenges were encountered and systematically resolved. The most complex issue involved PostgreSQL SSL connections, where RDS instances required SSL but used self-signed certificates causing application authentication failures. This was resolved by configuring the application to accept SSL connections while disabling strict certificate verification for RDS endpoints.

DNS propagation delays caused initial deployment failures during certificate validation processes. The solution involved implementing a two-stage deployment approach where infrastructure creation precedes certificate validation, allowing sufficient time for DNS propagation between deployment stages.

Application architecture complexity initially caused deployment failures due to intricate service dependencies and communication patterns. The resolution involved simplifying the overall architecture while implementing robust health checks and proper dependency management through Docker Compose configuration.

SSH tunneling required optimization to maintain stable database connections for administration tasks. Enhanced SSH configuration with keep-alive settings and proper timeout management resolved connection stability issues encountered during database management operations.

12 Key Technical Implementations

The Terraform implementation follows a modular design pattern with separate modules for networking, security, compute, database, load balancing, and monitoring components. This approach enables infrastructure reusability across environments while maintaining clear separation of concerns and facilitating maintenance.

Docker multi-stage builds optimize container images for production deployment while preserving development workflow efficiency. The build process significantly reduces final image sizes through staged compilation and dependency management, improving deployment speed and reducing storage costs.

Security implementation follows defense-in-depth principles with multiple layers of access control. Each infrastructure tier has specific security requirements implemented through security groups, IAM policies, and network access controls that work together to provide comprehensive protection.

13 Conclusion

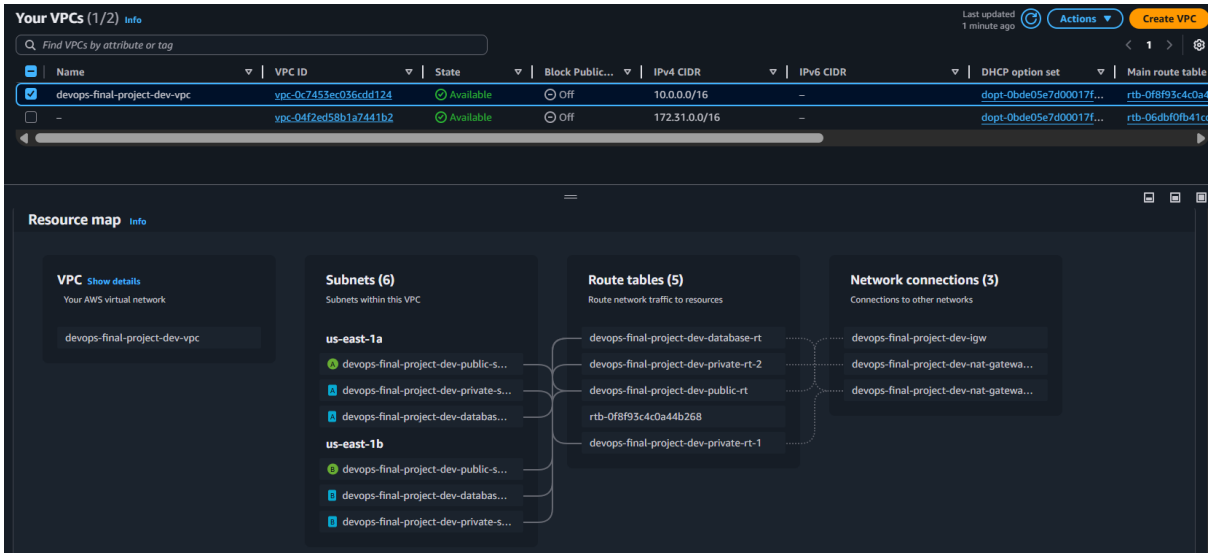
This implementation successfully fulfills all eight specified requirements while demonstrating advanced DevOps practices and AWS architectural best practices. The infrastructure provides a production-ready foundation that supports scalability, security, and operational excellence for modern web applications.

The modular Terraform design enables easy replication and environment-specific customization across development, staging, and production environments. The containerized application architecture supports modern deployment practices while maintaining operational simplicity and comprehensive monitoring capabilities.

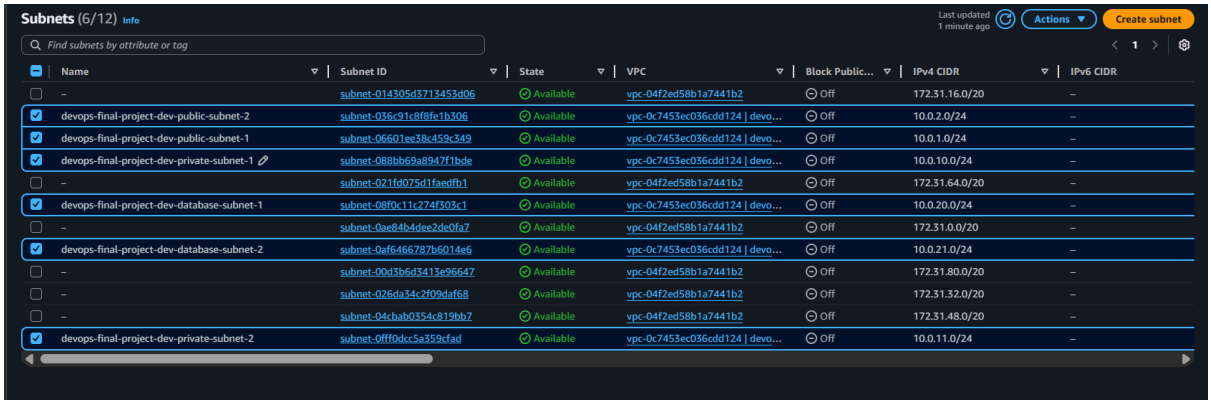
The project serves as a practical demonstration of cloud-native principles including auto-scaling, load balancing, database management, and business intelligence integration. The infrastructure represents a reference implementation suitable for enterprise-grade AWS deployments with complete documentation and established operational procedures.

14 Extra Screenshots

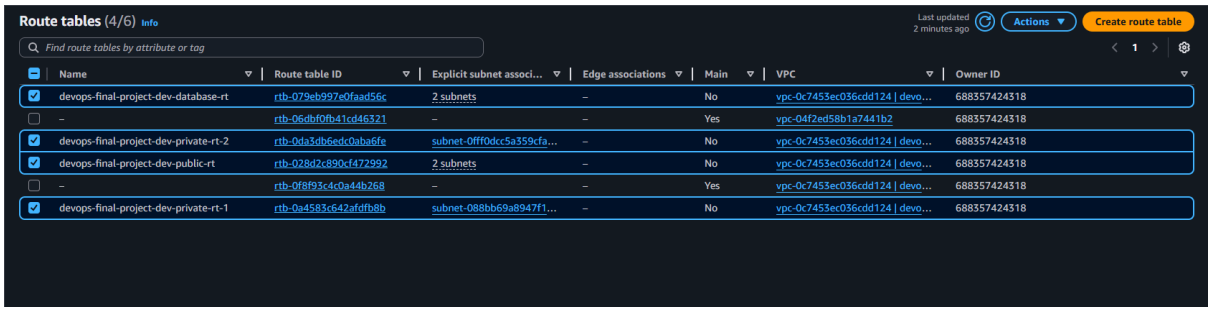
14.1 VPC



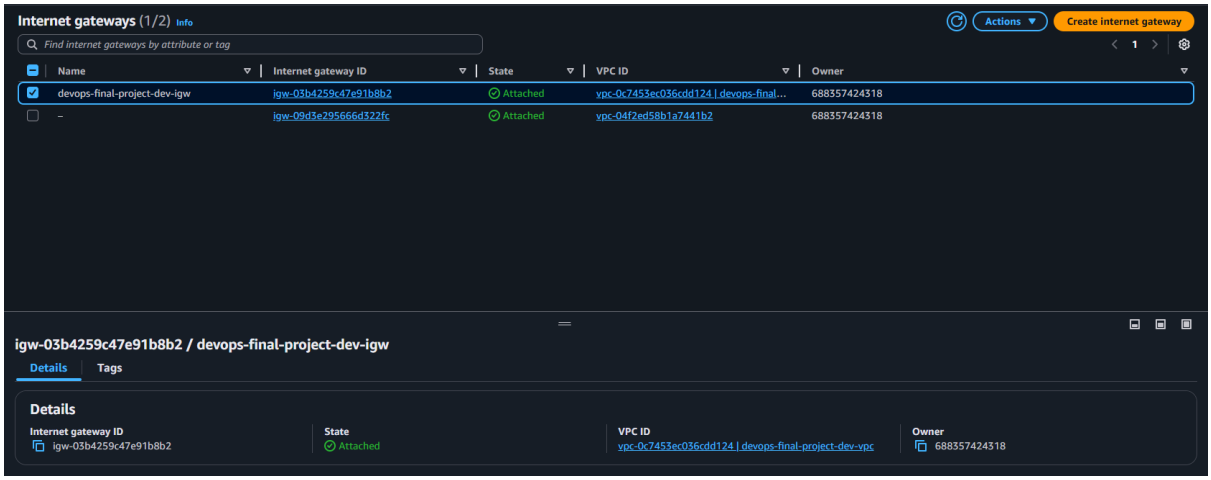
14.2 Subnets



14.3 VPC



14.4 Internet Gateways



14.5 NAT Gateways