Restaurant Reservation System

Mezgar Hazem

May 11, 2025

Abstract

This document provides comprehensive technical documentation for the Restaurant Reservation System, a microservices-based solution deployed with Docker, composed of an API Gateway, Restaurant Service, and Booking Service, utilizing gRPC for inter-service communication, Kafka for event streaming, and MongoDB for persistent data storage.

Contents

1 System Architecture

The system is composed of several containerized microservices:

- **API Gateway**: Node.js/Express with GraphQL
- **Restaurant Service**: gRPC service with MongoDB
- **Booking Service**: gRPC service with MongoDB
- **MongoDB**: Primary data store (separate databases for each service)
- **Kafka**: Event streaming platform
- **Zookeeper**: Kafka dependency

2 Component Details

2.1 API Gateway

Port	3000 (exposed)
Image	`api-gateway:latest`
Environment	RESTAURANT_SERVICE=restaurant-service:50051 BOOKING_SERVICE=booking

2.2 Restaurant Service

Port	50051 (internal)
Image	`restaurant-service:latest`
MongoDB	Database: `restaurants_db` Collection: `restaurants` Indexes: `_id`, `id`, `name`

2.3 Booking Service

Port	50052 (internal)
Image	`booking-service:latest`
MongoDB	Database: `bookings_db` Collection: `bookings` Indexes: `_id`, `id`, `restaurant_id`

3 Database Architecture

3.1 MongoDB Configuration

- **Container**: `mongodb:latest`
- **Port**: 27017 (internal)
- **Volumes**:
  - `/data/db` for persistent storage
  - Separate databases for services
- **Data Durability**:

- Journaling enabled
- WiredTiger storage engine

3.2 Data Models

```
// Restaurant Model
{
  _id: ObjectId,
  id: String,        // Unique business ID
  name: String,
  cuisine: String,
  createdAt: ISODate
}

// Booking Model
{
  _id: ObjectId,
  id: String,        // Unique booking ID
  restaurant_id: String,
  user_id: String,
  guests: Number,
  createdAt: ISODate
}
```

4 Docker Deployment

4.1 Requirements

- Docker Engine 20.10+
- Docker Compose 2.4+
- 4GB RAM minimum
- 2 CPU cores minimum

4.2 Deployment Steps

1. Clone the repository
2. Run `docker-compose build`
3. Start services: `docker-compose up -d`
4. Verify containers: `docker-compose ps`

4.3 Service Dependencies

Service	**Dependencies**
API Gateway	Kafka, Restaurant Service, Booking Service
Restaurant Service	MongoDB, Kafka
Booking Service	MongoDB, Kafka
Kafka	Zookeeper

5 Environment Variables

Key configuration parameters:

Service	Variable	Purpose
All	MONGO_URI	MongoDB connection string
All	KAFKA_BROKER	Kafka broker address
Gateway	RESTAURANT_SERVICE	Restaurant service endpoint
Gateway	BOOKING_SERVICE	Booking service endpoint

6 Data Flow

6.1 Create Restaurant

1. Client POST to API Gateway
2. Gateway gRPC call to Restaurant Service
3. Service validates and publishes to Kafka
4. Kafka consumer persists to MongoDB
5. Response returned through chain

6.2 Create Booking

1. Client POST to API Gateway
2. Gateway verifies restaurant exists
3. Booking Service processes via gRPC
4. Event published to Kafka
5. Consumer persists booking to MongoDB

7 Troubleshooting

7.1 Common Issues

- **MongoDB connection failures**: Verify `MONGO_URI` and container health
- **Kafka timeouts**: Check Zookeeper and Kafka logs
- **gRPC errors**: Validate service ports and health checks

7.2 Logging

- All services log to stdout
- Use `docker-compose logs <service>` to view
- Key log patterns:
  - MongoDB connection established
  - Kafka producer/consumer events
  - gRPC method calls