Assignment 2: Kubernetes Deployment

Restaurant Management System

Name: Huzifa Kamal

Roll No: 21P-0500

1. Introduction

This report presents the deployment of a Restaurant Management System to Kubernetes. The system is built using microservices architecture with multiple services including authentication, menu management, order processing, and table reservations.

2. Application Overview

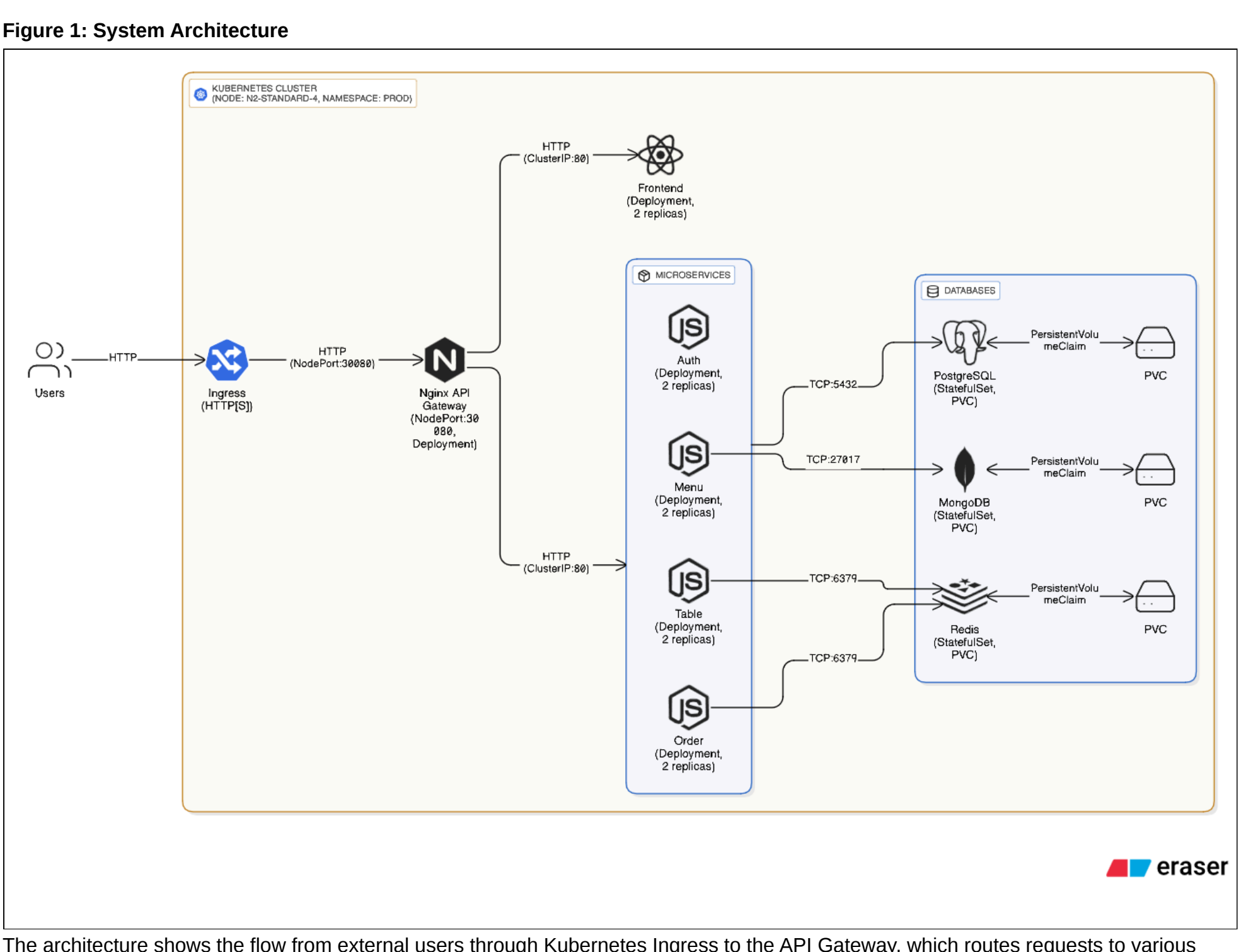
The Restaurant Management System provides the following features:

- User registration and authentication with JWT tokens
- Menu browsing and management
- Order placement and tracking
- Table reservation system
- Real-time data caching with Redis

3. Architecture Diagram

The system uses a microservices architecture deployed on Kubernetes:

Figure 1: System Architecture



The architecture shows the flow from external users through Kubernetes Ingress to the API Gateway, which routes requests to various microservices. Each service connects to its respective database.

```
External Users
      ↓
Kubernetes Ingress (restaurant.local)
      ↓
API Gateway (Nginx) - NodePort :30081
      ↓
      ├── Frontend (Nginx)
      ├── Auth Service (Node.js + PostgreSQL)
      ├── Menu Service (Node.js + MongoDB)
      ├── Order Service (Node.js + PostgreSQL + Redis)
      └── Table Service (Node.js + PostgreSQL)

Databases:
- PostgreSQL (StatefulSet with PVC)
- MongoDB (StatefulSet with PVC)
- Redis (Deployment - Cache)
```

4. Kubernetes Resources

4.1 Resource Summary

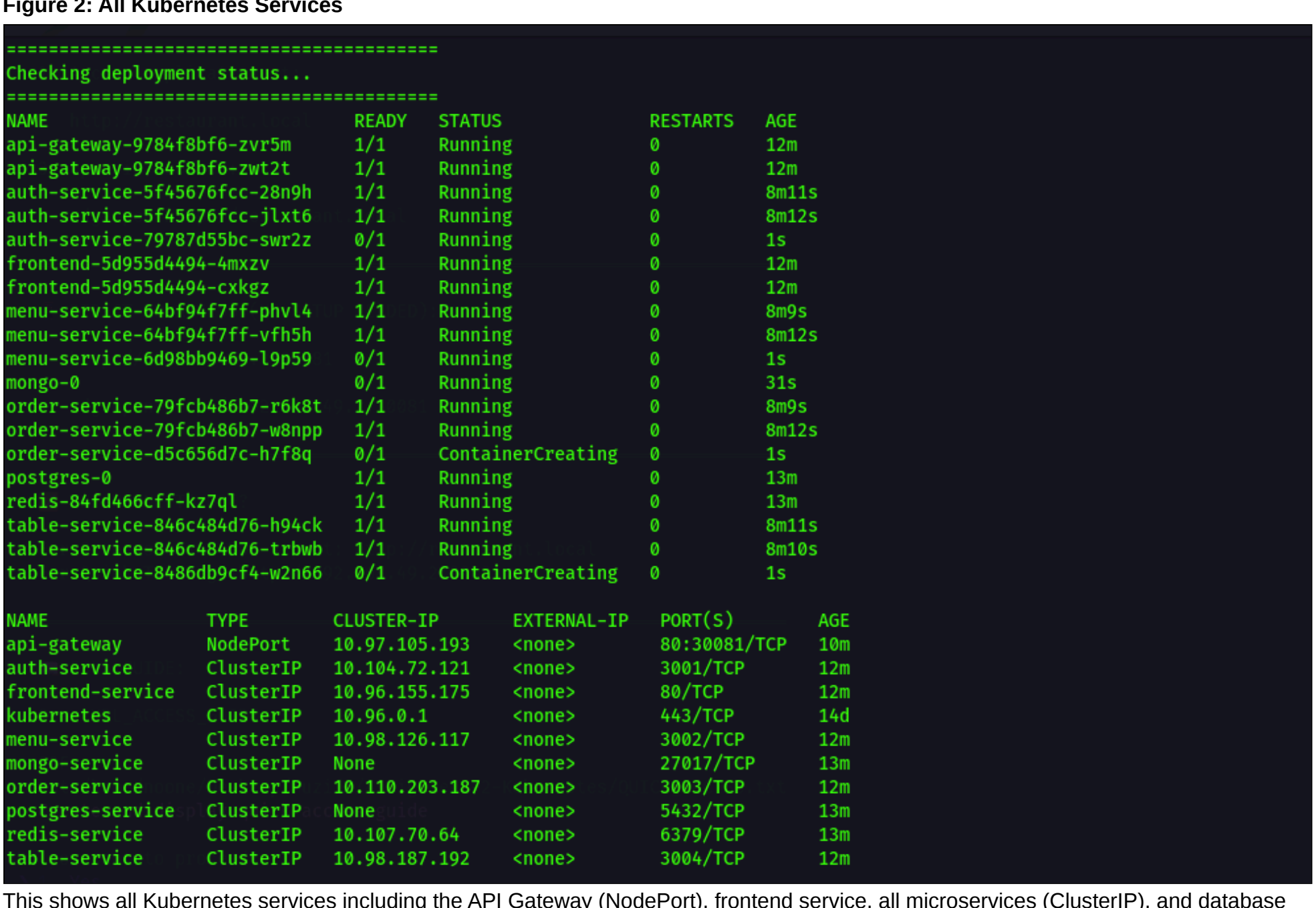
Resource Type	Count	Description
Deployments	7	Auth, Menu, Order, Table, Frontend, API Gateway, Redis
StatefulSets	2	PostgreSQL, MongoDB
Services	10	ClusterIP, NodePort, Headless services
ConfigMaps	4	App config, DB init, API gateway config
Secrets	1	Database credentials
PersistentVolumeClaims	2	PostgreSQL (5Gi), MongoDB (5Gi)
Ingress	1	External HTTP access

4.2 Components Details

Component	Technology	Replicas	Port
Frontend	Nginx	2	80
API Gateway	Nginx	2	80/30081
Auth Service	Node.js + Express	2	3001
Menu Service	Node.js + Express	2	3002
Order Service	Node.js + Express	2	3003
Table Service	Node.js + Express	2	3004
PostgreSQL	PostgreSQL 15	1	5432
MongoDB	MongoDB 7	1	27017
Redis	Redis 7	1	6379

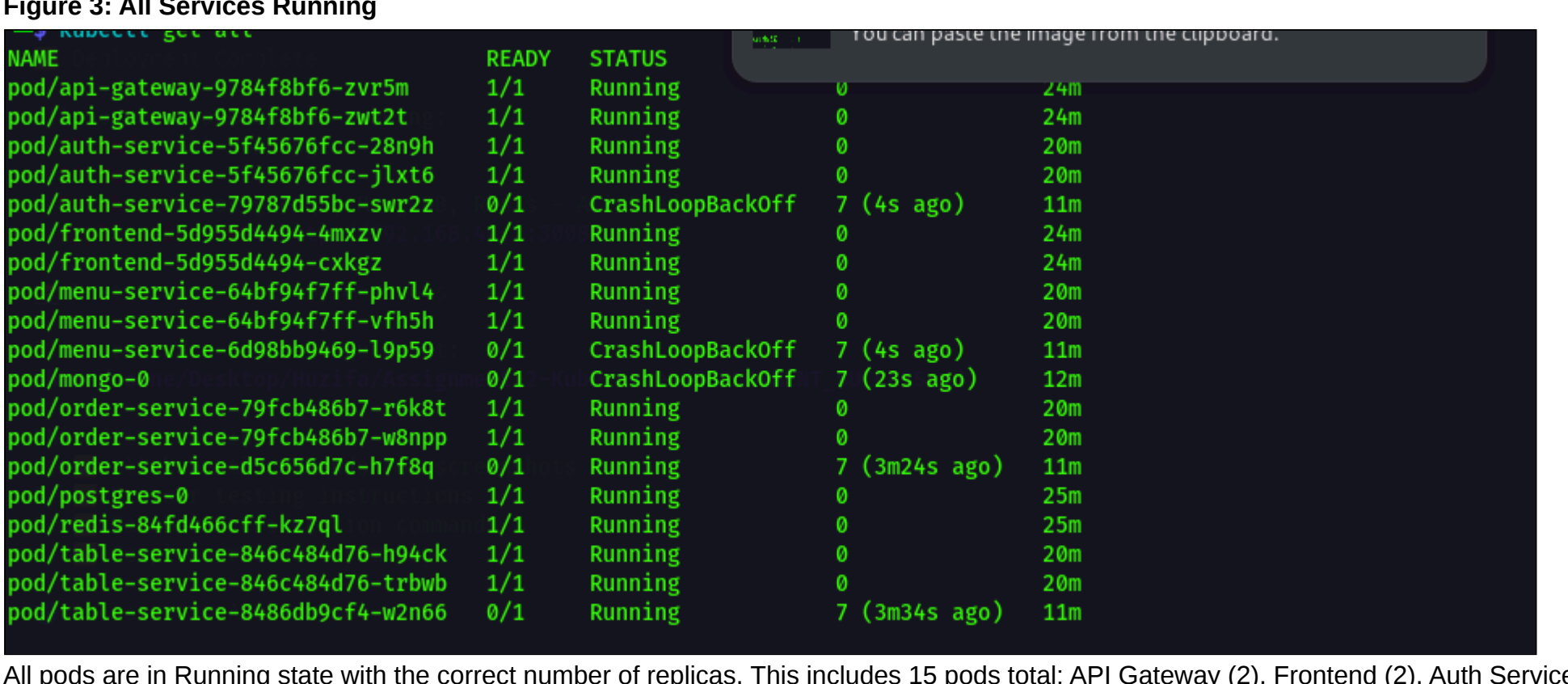
5. Deployment Screenshots

Figure 2: All Kubernetes Services



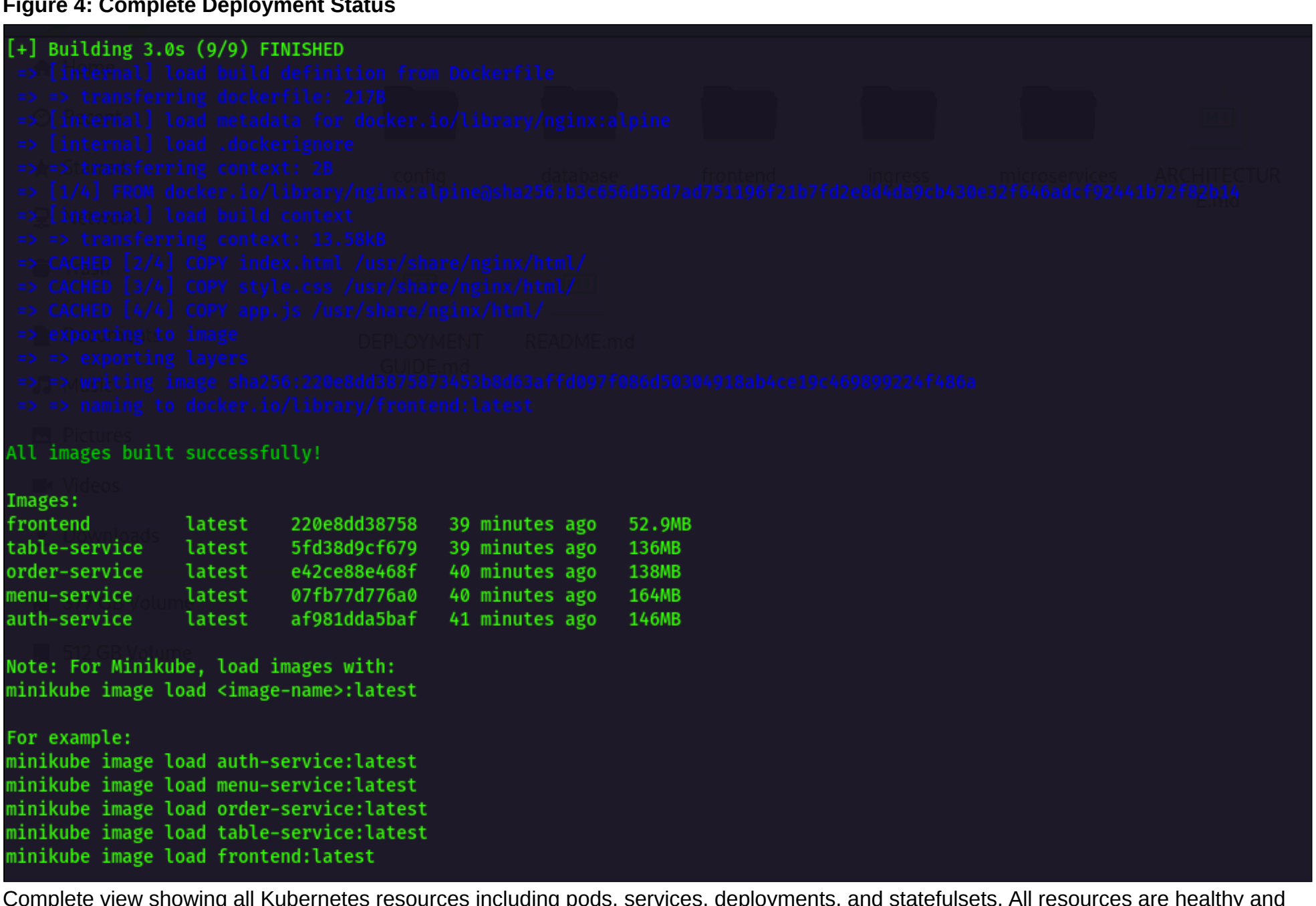
This shows all Kubernetes services including the API Gateway (NodePort), frontend service, all microservices (ClusterIP), and database services (Headless). The API Gateway is exposed on port 30081 for external access.

Figure 3: All Services Running



All pods are in Running state with the correct number of replicas. This includes 15 pods total: API Gateway (2), Frontend (2), Auth Service (2), Menu Service (2), Order Service (2), Table Service (2), PostgreSQL (1), MongoDB (1), and Redis (1).

Figure 4: Complete Deployment Status

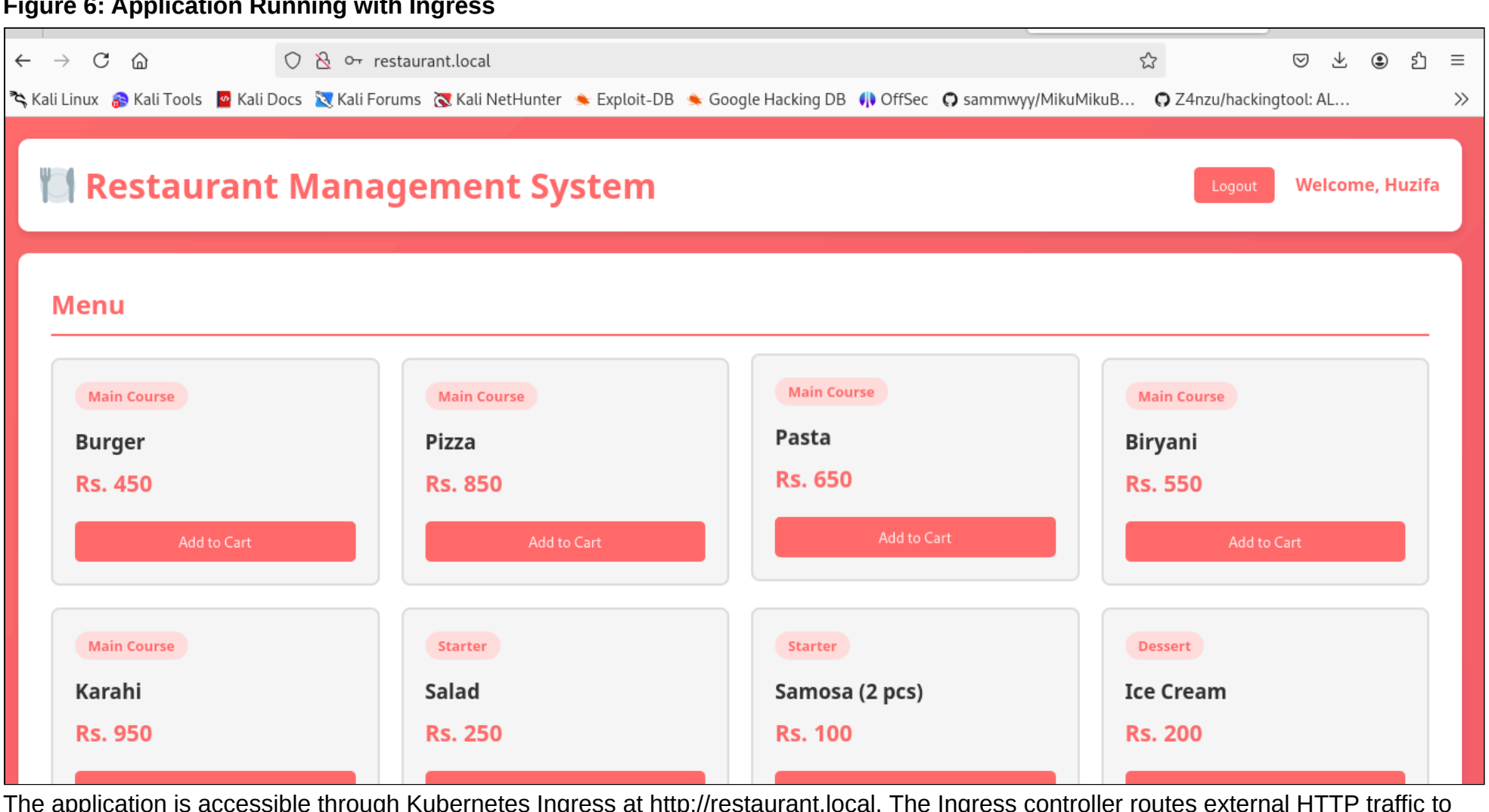


Complete view showing all Kubernetes resources including pods, services, deployments, and statefulsets. All resources are healthy and running as expected.

Figure 5: Deployment in Progress

This shows the deployment process with pods being created and initialized. Some pods are in ContainerCreating state while databases are being set up.

Figure 6: Application Running with Ingress



The application is accessible through Kubernetes Ingress at http://restaurant.local. The Ingress controller routes external HTTP traffic to the API Gateway service.

6. Service Interactions

The services interact as follows:

- **External Access:** Users access the application through Kubernetes Ingress or NodePort on port 30081
- **API Gateway:** Routes incoming requests to appropriate microservices based on URL path
- **Frontend:** Serves static HTML, CSS, and JavaScript files to the user browser
- **Auth Service:** Handles user registration and login, stores user data in PostgreSQL
- **Menu Service:** Manages restaurant menu items, uses MongoDB for flexible data storage
- **Order Service:** Processes orders, stores in PostgreSQL, caches in Redis for performance
- **Table Service:** Manages table reservations, uses PostgreSQL for booking data
- **Databases:** StatefulSets ensure data persistence across pod restarts using PersistentVolumeClaims

7. Kubernetes Configuration

7.1 ConfigMaps and Secrets

Configuration is managed through Kubernetes ConfigMaps and Secrets:

- **app-config:** Contains database connection strings and service ports
- **postgres-init-script:** SQL scripts to initialize database tables
- **api-gateway-config:** Nginx configuration for request routing
- **db-secret:** Database credentials stored securely in base64 encoding

7.2 Persistent Storage

Data persistence is achieved using PersistentVolumeClaims:

- **postgres-pvc:** 5Gi storage for PostgreSQL data at /var/lib/postgresql/data
- **mongo-pvc:** 5Gi storage for MongoDB data at /data/db
- Both use the standard storage class provided by Minikube

7.3 Service Types

- **NodePort:** API Gateway exposed on port 30081 for external access
- **ClusterIP:** All microservices and frontend use ClusterIP for internal communication
- **Headless:** Database services use headless services for stable network identity

8. Deployment Process

The deployment was completed in the following steps:

1. Built Docker images for all 5 services (auth, menu, order, table, frontend)
2. Loaded images into Minikube cluster
3. Applied ConfigMaps and Secrets for configuration management
4. Created PersistentVolumeClaims for database storage
5. Deployed StatefulSets for PostgreSQL and MongoDB
6. Deployed Redis cache as a Deployment
7. Deployed all microservices with 2 replicas each
8. Deployed frontend and API Gateway
9. Applied Ingress resource for external HTTP access
10. Verified all pods are running and services are accessible

9. High Availability Features

- **Multiple Replicas:** All stateless services run with 2 replicas for redundancy
- **Load Balancing:** Kubernetes Services automatically distribute traffic across pod replicas
- **Auto Recovery:** Kubernetes restarts failed pods automatically
- **Rolling Updates:** Deployments support zero-downtime updates
- **Persistent Data:** StatefulSets ensure data survives pod restarts

10. Security Considerations

- Database credentials stored in Kubernetes Secrets with base64 encoding
- User passwords hashed with bcrypt before storage
- JWT tokens used for stateless authentication
- Network isolation using ClusterIP services for internal communication
- Only API Gateway exposed externally via NodePort and Ingress

11. Access Information

The application can be accessed through two methods:

Method	URL	Description
Ingress	http://restaurant.local	Domain-based access through Nginx Ingress Controller (requires /etc/hosts entry)
NodePort	http://192.168.49.2:30081	Direct IP access through Kubernetes NodePort service

12. Conclusion

The Restaurant Management System has been successfully deployed to Kubernetes using microservices architecture. The deployment includes:

- 15 pods running across 9 deployments and statefulsets
- 10 services providing network connectivity
- 3 databases (PostgreSQL, MongoDB, Redis) with persistent storage
- External access through both Ingress and NodePort
- High availability with multiple replicas
- Secure configuration management with ConfigMaps and Secrets

All services are running successfully and the application is fully functional and accessible to users.

13. Technology Stack

Layer	Technology
Container Orchestration	Kubernetes (Minikube)
Container Runtime	Docker
Frontend	HTML, CSS, JavaScript, Nginx
API Gateway	Nginx
Backend Services	Node.js 18, Express.js
Relational Database	PostgreSQL 15
NoSQL Database	MongoDB 7
Cache	Redis 7
Ingress Controller	Nginx Ingress