2. Deployment Strategy (Production-Ready)

2.1 Containerization

- **Dockerization of Microservices**:
  - Each microservice has its own Dockerfile, ensuring separation of concerns.
  - Containers encapsulate all dependencies and runtime configurations.
- **Multi-Stage Builds**:
  - Use **multi-stage Docker builds** to keep images small and secure.
  - Final image built on a **distroless base image**, minimizing attack surface and removing unnecessary tools (e.g., bash, curl).
- **Best Practices**:
  - Use .dockerignore to prevent leaking sensitive files.
  - Regularly scan images using tools like **Trivy** or **Grype**.

2.2 Orchestration

- **Development Environment**:
  - Use **Docker Compose** to spin up multiple services locally.
  - Supports volume mounting, service dependencies, and .env injection.
- **Production Environment**:
  - Use **Kubernetes (K8s)** for orchestration.
  - Deployment manifests located under deploy/k8s/
  - Supports horizontal scaling, self-healing, service discovery, and resource limits.
- **Compatible Platforms**:
  - Can be deployed on **Amazon EKS**, **Google GKE**, **Azure AKS**, or self-hosted K8s clusters.
- **Namespace Management**:
  - Use namespaces to isolate environments (e.g., dev, staging, prod).

2.3 CI/CD

- **CI/CD Tools**:
  - Use **GitHub Actions**, **GitLab CI**, or **CircleCI** for automation.
- **Pipeline Stages**:

1. **Linting**: Enforce code style using golangci-lint
2. **Testing**: Run unit and integration tests
3. **Build**: Compile and build Docker images
4. **Push**: Push versioned images to Docker Hub / ECR / GCR
5. **Deploy**: Roll out changes to Kubernetes using Helm or Kustomize

- **Deployment Strategies**:
  - Support for **Blue/Green** or **Canary Deployments** using **ArgoCD** or **Flux**.
  - Enable automatic rollback on health check failure.

2.4 Secrets Management

- **Storage Options**:
  - **Kubernetes Secrets**: For environment-specific secrets mounted at runtime.
  - **AWS Secrets Manager** or **HashiCorp Vault**: For dynamic and secure secret injection.
- **Security Measures**:
  - Never commit .env or .secret files to version control.
  - Rotate secrets periodically and enforce least privilege access.

---

3. Non-Functional Requirements (NFRs)

3.1 Logging

- **Structured Logging**:
  - Use **uber-go/zap** or **rs/zerolog** for structured logs.
  - Logs are in **JSON format** for compatibility with aggregators.
- **Log Aggregation**:

    - Logs are collected using **FluentBit** and forwarded to:
        - **ElasticSearch** (via EFK stack: Elasticsearch, FluentBit, Kibana)
        - **Grafana Loki** for a lightweight, cost-effective solution
- **Example Log Entry**:

```
{
 "level": "info",
 "service": "auth",
 "msg": "user login successful",
 "userID": "123",
 "timestamp": "2025-08-03T12:00:00Z"
}
```

3.2 Authentication & Authorization

- **Authentication**:
    - Use **JWTs** for access and refresh tokens
    - Middleware extracts and validates token, injecting claims into request context
- **Authorization**:
    - Use **Role-Based Access Control (RBAC)**:
        - Roles: Admin, User, Vendor
        - Configurable via database or environment variables
- **Future Integration Options**:
    - Support for **OAuth2** / **OpenID Connect** via **Google**, **Auth0**, **Keycloak**

3.3 Monitoring & Observability

- **Metrics Collection**:
    - Use **Prometheus** to scrape service metrics
    - Dashboards built with **Grafana**

- Key Metrics:
    - HTTP request count & duration
    - DB query performance
    - Memory/CPU usage per container
- **Distributed Tracing**:
  - Use **OpenTelemetry (OTEL)** SDKs for tracing
  - Backend options: **Jaeger**, **Tempo**
- **Health Probes**:
  - Implement /healthz endpoints for:
    - **Readiness probe**: Determines if app is ready to serve
    - **Liveness probe**: Determines if app should be restarted

---

3.4 Rate Limiting & Throttling

- **Enforcement Points**:
  - API Gateway (e.g., **Kong**, **Nginx Ingress**) applies global rate limits
  - Per-user or per-IP rate limiting via middleware
- **Implementation**:
  - Use **Redis-backed** token bucket using libs like ulule/limiter
- **Benefits**:
  - Prevents DDoS and abuse
  - Ensures fair usage across users

3.5 Caching

- **Cache Layer**:
  - Use **Redis** for:
    - Session and token storage
    - Frequently queried data (e.g., restaurant list, menu items)
- **Caching Strategy**:

    - Set **TTL (Time-to-Live)** on keys
    - Manual invalidation on updates (e.g., restaurant updated)
- **Optional**: In-memory LRU caching for performance-sensitive reads

---

3.6 Secure Communication

- **Ingress Security**:
    - TLS termination at **Ingress Controller** (e.g., Nginx, Traefik)
    - Use **Let's Encrypt** or **cert-manager** for auto-renewed certs
- **Internal Security**:
    - gRPC/HTTP traffic restricted via **Kubernetes Network Policies**
    - Enable **mTLS** for service-to-service communication if required

---

3.7 Database & Storage

- **Transactional Database**:
    - **MySQL** used for persistent relational data
    - Ensure regular **automated backups** using MySQL dump, Percona XtraBackup, or managed snapshots (e.g., AWS RDS)
    - Tune queries and indexes for high throughput under load
- **Migrations**:
    - Use **Golang-migrate** or **goose** for version-controlled schema changes
    - Run migrations as part of CI/CD pipeline before deployment
- **Object Storage**:
    - Use **AWS S3**, **MinIO**, or compatible service
    - For user uploads (e.g., images, documents)
    - Ensure access via signed URLs and configure proper IAM policies