Deployment Infrastructure Documentation

Overview

This document describes the deployment architecture, services, environments, and tools used to deploy and manage the messaging application. It ensures that all components work seamlessly across environments with high scalability, security, and performance.

Deployment Architecture

1. Components

The app is composed of the following key components:

- **Frontend**:
  - Flutter app for Android, iOS, and Web.
- **Backend**:
  - API built using Node.js with NestJS.
- **Database**:
  - PostgreSQL for structured data.
  - Redis for caching and session management.
- **Storage**:
  - S3-compatible object storage for media (e.g., images, videos).
- **Messaging**:
  - WebSocket server for real-time communication.
- **Authentication**:
  - Firebase Authentication for user management.
- **Monitoring**:
  - Prometheus and Grafana for metrics.
  - ELK Stack (Elasticsearch, Logstash, Kibana) for log aggregation and analysis.

Deployment Infrastructure

1. Environments

- **Development**:
  - Used for feature development and debugging.
  - Hosted on a local machine or cloud-based staging server.
- **Staging**:
  - Mirrors production for testing before deployment.
  - Hosted on a separate cloud server.
- **Production**:
  - The live environment used by end-users.
  - Deployed on a scalable cloud infrastructure.

2. Cloud Provider

- Preferred: **AWS (Amazon Web Services)**.
- Alternatives: Google Cloud Platform (GCP), Microsoft Azure.

3. Load Balancer

- **Purpose**: Distributes incoming traffic to backend services.
- Tool: AWS Elastic Load Balancer (ELB) or NGINX.

4. CI/CD Pipeline

- **Purpose**: Automates code integration, testing, and deployment.
- Tools:
  - GitHub Actions for CI/CD workflows.
  - Docker for containerization.
  - Kubernetes (K8s) for container orchestration.

5. Networking

- **Firewall**:
  - Restricts unauthorized access using security groups and IP whitelisting.
- **DNS**:
  - Cloudflare for domain management and DDoS protection.
- **SSL/TLS**:
  - Ensures secure HTTPS communication using Let's Encrypt or AWS ACM.

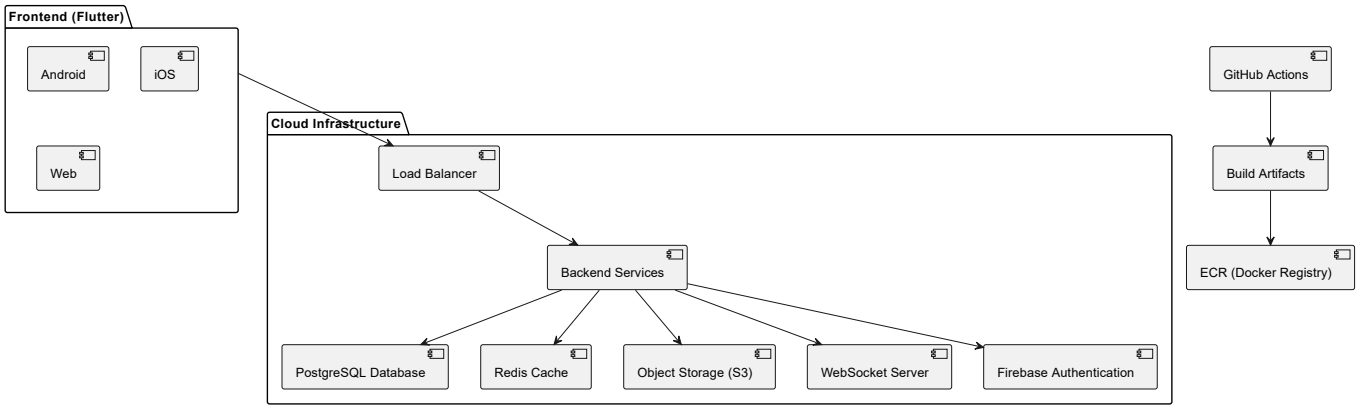
Deployment Workflow

Step-by-Step Pipeline

1. **Code Commit**:
   - Developers push changes to the GitHub repository.
2. **Continuous Integration (CI)**:
   - Run automated tests (unit, integration).
   - Build Docker images for frontend and backend.
3. **Artifact Storage**:
   - Push Docker images to AWS Elastic Container Registry (ECR).
4. **Continuous Deployment (CD)**:
   - Deploy new versions to the staging environment.
   - Run automated end-to-end tests in staging.
   - Manually approve deployment to production.
5. **Production Deployment**:

- Deploy to Kubernetes clusters using Helm charts.
- Update DNS records to point to the new version.

High-Level Diagram of Deployment Infrastructure



Scalability and Reliability

1. Horizontal Scaling

- Use Kubernetes to add or remove pods dynamically based on load.

2. Database Scaling

- Read replicas for PostgreSQL to distribute query load.
- Indexing for optimized search queries.

3. Caching

- Use Redis to cache frequently accessed data (e.g., user profiles, chat lists).

4. Disaster Recovery

- Automated backups of databases and storage.
- Multi-region deployment to ensure availability.

5. Monitoring

- Real-time logs and metrics using:
  - Grafana for dashboards.
  - Prometheus for alerts.

Security Practices

1. **API Security**:
   - Use JWT for authenticated API requests.
   - Rate limiting with tools like NGINX or API Gateway.

2. **Data Encryption**:
    - Encrypt data at rest using AES-256.
    - Use HTTPS (TLS) for secure data in transit.
3. **Access Control**:
    - Restrict database access to backend services.
    - Implement role-based access control (RBAC).
4. **Environment Secrets**:
    - Store sensitive data (e.g., API keys) in AWS Secrets Manager or Vault.

Deployment Checklist

- [ ] Code passes all automated tests.
- [ ] Docker images are built and pushed to the registry.
- [ ] Helm charts are updated and reviewed.
- [ ] Staging environment tests are successful.
- [ ] Approval received for production deployment.
- [ ] Production deployment completed successfully.
