Deployment Model

Overview

The BRAD system is deployed as a set of Docker containers orchestrated using Docker Compose. The deployment is intended for on-premises environments, where all services run on the client's server. Users interact with the system via a web browser, accessing the frontend component.

Target Environment

- **Type:** On-premises server
- **Operating System:** Linux or Windows (with Docker support)
- **Access:** Internal network (browser access to frontend)
- **Security:** Server should be protected by a firewall and only necessary ports exposed.

Deployment Topology

The system follows a multi-tier architecture, with each major component running in its own container:

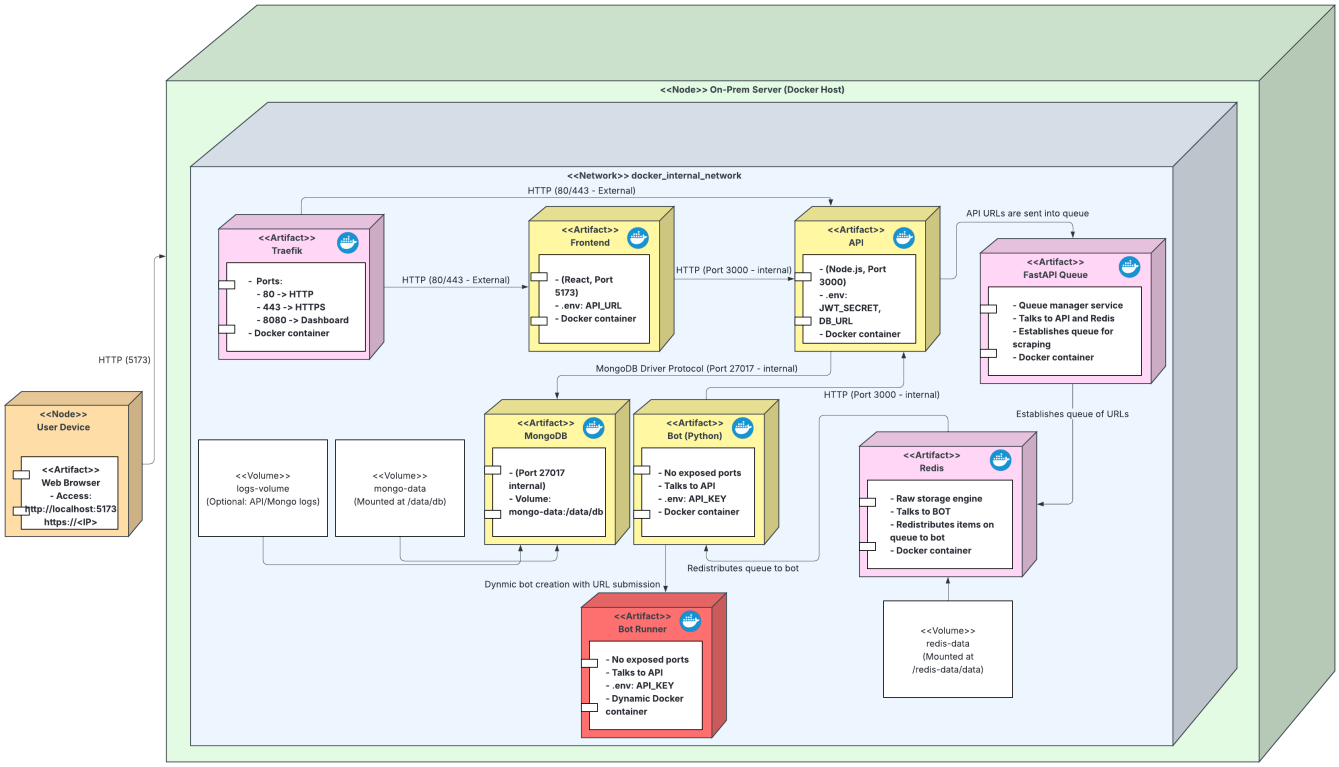
- **Frontend:** React-based web application, exposed on port 5173.
- **API:** Node.js/Express backend, exposed on port 3000.
- **Bot:** Python service for reporting abusive domains, communicates with the API.
- **Database:** MongoDB, data persisted via a Docker volume.

All containers are connected via an internal Docker network, ensuring secure communication between services. Sensitive environment variables are managed via Docker Compose, but for production, a `.env` file or Docker secrets is recommended.

Tools and Platforms

- **Docker**: Containerization of all services.
- **Docker Compose**: Orchestration and management of multi-container setup.
- **MongoDB**: Database service, running in a container.
- **React**: Frontend application.
- **Python**: Bot service.

Deployment Diagram



Quality Requirements Support

- **Security:**
  - All inter-service traffic is isolated within the Docker network.
  - API and database ports are only exposed as needed; firewall rules should restrict external access.
  - Sensitive data (JWT secrets, email credentials) should be managed via environment variables or Docker secrets.
  - Role-based access control and authentication are enforced at the API level.
  - For production, enable TLS/SSL termination at the server or via a reverse proxy.
  - MongoDB data is persisted in a Docker volume; ensure disk encryption at the OS level if required.
  - Multi-factor authentication (MFA) and audit logging are implemented in the application layer.
- **Compliance:**
  - Data handling and storage comply with GDPR and POPIA requirements.
  - Audit logs are generated by the application and should be stored securely (consider mounting a host directory for logs).
  - User consent and data minimization are enforced in the application logic.
- **Reliability:**
  - Docker Compose manages service dependencies and restarts (e.g., `restart: always` for MongoDB).
  - Application logic includes error handling and recovery for failed submissions or bot crashes.
  - Data is persisted in Docker volumes to prevent loss on container restart.
- **Scalability:**

    - The containerized architecture allows for horizontal scaling (e.g., running multiple API or bot containers if needed).
    - Services can be distributed across multiple hosts if Docker Swarm or Kubernetes is adopted in the future.

- **Maintainability:**

    - Each component is isolated in its own container, enabling independent updates and troubleshooting.
    - Docker Compose simplifies deployment, upgrades, and rollback.
    - Modular codebase supports targeted changes (e.g., updating the bot or frontend without affecting others).

Deployment Steps

1. **Install Docker and Docker Compose** on the target server.
2. **Clone the project repository** to the server.
3. **Configure environment variables** in a `.env` file (recommended for production).
4. **Run** `docker-compose up --build -d` to build and start all services.
5. **Access the system** via a browser at `http://<server-ip>:5173`.

Security and Compliance Recommendations

- Restrict MongoDB port (`27018`) to internal access only; do not expose to the public internet.
- Use strong, unique secrets for JWT and bot access keys.
- Store audit logs in a secure, tamper-evident location.
- Regularly update Docker images to patch vulnerabilities.
- For production, consider using a reverse proxy (e.g., Nginx) for TLS termination and additional access control.
