Docker Configuration Documentation

This document provides comprehensive documentation for the Docker setup used in the AI Research Assistant application. The configuration implements a containerized 3-service architecture with proper isolation, networking, and development workflows.

Architecture Overview

The Docker setup manages a 3-service architecture with an nginx reverse proxy:

1. **Express DB Server** (`express-db-server`) - Port 3001 - Database operations via Supabase RPC
2. **FastAPI AI Server** (`fastapi-ai-server`) - Port 8000 - AI/ML operations and chat functionality
3. **Next.js Frontend** (`frontend`) - Port 3000 - User interface
4. **Nginx Reverse Proxy** (`nginx`) - Port 80/443 - Load balancing and routing

Project Structure

```
project-root/
├── docker-compose.yml          # Main orchestration file
├── Dockerfile.frontend         # Next.js frontend container
├── Dockerfile.backend          # FastAPI AI server container
├── express-db-server/
│   └── Dockerfile             # Express DB server container
├── nginx.conf                 # Nginx configuration
├── .env                       # Environment variables
├── .env.example              # Environment template
├── start-services.sh         # Production startup script
├── start.sh                  # Development startup script
└── ssl/                      # SSL certificates directory
```

PROF

Docker Compose Configuration

Main Services Definition

The `docker-compose.yml` file defines all services with their dependencies, networking, and health checks.

Express DB Server Service

```
express-db-server:
  build:
    context: ./express-db-server
    dockerfile: Dockerfile
  container_name: express-db-server
```

```
    ports:
      - "3001:3001"
    env_file:
      - .env
    environment:
      - SUPABASE_URL=${SUPABASE_URL}
      - SUPABASE_SERVICE_ROLE_KEY=${SUPABASE_SERVICE_ROLE_KEY}
      - SUPABASE_ANON_KEY=${SUPABASE_ANON_KEY}
      - NODE_ENV=development
      - PORT=3001
      - ALLOWED_ORIGINS=${ALLOWED_ORIGINS}
      - JWT_SECRET=${JWT_SECRET}
    volumes:
      - ./express-db-server:/app
      - /app/node_modules
      - ./.env:/app/.env:ro
    restart: unless-stopped
    networks:
      - app-network
      - supabase_network_Ai-Research-Assistant-local
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:3001/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s
```

Key Features:

- **External Network**: Connects to Supabase local network
- **Volume Mounts**: Live reload for development
- **Health Check**: Ensures service availability
- **Environment**: Development configuration

PROF

FastAPI AI Server Service

```
  fastapi-ai-server:
    build:
      context: ./backend
      dockerfile: ../Dockerfile.backend
    container_name: fastapi-ai-server
    ports:
      - "8000:8000"
    env_file:
      - .env
    environment:
      - PYTHONPATH=/app
      - PYTHONUNBUFFERED=1
      - EXPRESS_DB_URL=http://express-db-server:3001
      - DATA_DIR=/app/data
```

```
    volumes:
      - ./backend:/app
      - /app/.venv  # Exclude virtual environment
      - ./data:/app/data
      - ai-models:/app/models
    depends_on:
      - express-db-server
    restart: unless-stopped
    networks:
      - app-network
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s
```

Key Features:

- **Service Communication**: Uses Express DB server URL
- **AI Models Volume**: Persistent storage for ML models
- **Python Environment**: UV package manager integration
- **No Database Access**: Communicates via HTTP with Express server

Frontend Service

```
frontend:
  build:
    context: ./frontend
    dockerfile: ../Dockerfile.frontend
  container_name: react-frontend
  ports:
    - "3000:3000"
  env_file:
    - .env
  environment:
    - NODE_ENV=development
  volumes:
    - ./frontend:/app
    - /app/node_modules
  depends_on:
    - express-db-server
    - fastapi-ai-server
  restart: unless-stopped
  networks:
    - app-network
```

PROF

Key Features:

- **Hot Reload**: Development mode with live updates
- **Service Dependencies**: Waits for backend services
- **Node Modules**: Cached for faster builds

Nginx Reverse Proxy

```
nginx:
  image: nginx:alpine
  container_name: nginx-proxy
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf:ro
    - ./ssl:/etc/nginx/ssl:ro
  depends_on:
    - frontend
    - express-db-server
    - fastapi-ai-server
  restart: unless-stopped
  networks:
    - app-network
  healthcheck:
    test: ["CMD", "wget", "-qO-", "http://localhost/health"]
    interval: 30s
    timeout: 10s
    retries: 3
    start_period: 10s
```

Key Features:

- **Alpine Image**: Lightweight nginx image
- **SSL Ready**: SSL directory mounted for certificates
- **Configuration Mount**: Read-only nginx config

PROF

Volumes and Networks

Volumes

```
volumes:
  ai-models:
    driver: local
```

- **ai-models**: Persistent storage for AI/ML models across container restarts

Networks

```
networks:
  app-network:
    driver: bridge
  supabase_network_Ai-Research-Assistant-local:
    external: true
```

- **app-network**: Internal bridge network for service communication
- **supabase_network**: External network for Supabase local integration

Individual Dockerfiles

Frontend Dockerfile (`Dockerfile.frontend`)

```
FROM node:20

WORKDIR /app

# Copy package.json and lock first (cache layer)
COPY package*.json ./

# Install dependencies using npm ci for reproducible builds
RUN npm ci

# Copy frontend source code
COPY . .

# Expose Next.js dev port
EXPOSE 3000

# Run Next.js dev server
CMD ["npm", "run", "dev"]
```

PROF

Optimization Features:

- **Layer Caching**: Package files copied first for better caching
- **Reproducible Builds**: Uses `npm ci` instead of `npm install`
- **Development Mode**: Runs with hot reload

Backend Dockerfile (`Dockerfile.backend`)

```
FROM python:3.12-slim-trixie

# Install system dependencies for AI/ML operations
RUN apt-get update && apt-get install -y --no-install-recommends \
    curl gcc g++ make ca-certificates \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*
```

```
# Install uv
COPY --from=ghcr.io/astral-sh/uv:latest /uv /bin/uv

WORKDIR /app

# Create non-root user for security with home directory first
RUN groupadd -r aiuser && useradd -r -g aiuser -m aiuser

# Create and set permissions for directories
RUN mkdir -p /app/data /app/models /app/temp /home/aiuser/.cache
RUN chown -R aiuser:aiuser /app /home/aiuser/.cache

# Switch to non-root user for dependency installation
USER aiuser

# Set environment variables for uv to use user-owned directories
ENV UV_CACHE_DIR=/home/aiuser/.cache/uv
ENV UV_PROJECT_ENVIRONMENT=/app/.venv

# Copy dependency definitions
COPY --chown=aiuser:aiuser pyproject.toml uv.lock* ./

# Install dependencies using uv as non-root user
RUN uv sync --frozen --no-dev

# Copy backend source code (AI/ML focused)
COPY --chown=aiuser:aiuser . .

# Expose port
EXPOSE 8000

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8000/health || exit 1

# Run FastAPI AI server (database endpoints removed)
CMD ["uv", "run", "uvicorn", "app.main:app", "--host", "0.0.0.0", "--
port", "8000", "--reload"]
```

PROF

Security & Performance Features:

- **Non-root User**: Runs as `aiuser` for security
- **UV Package Manager**: Fast Python dependency management
- **Multi-stage Optimization**: Minimal runtime dependencies
- **Health Check**: Container-level health monitoring
- **AI/ML Dependencies**: Includes compilation tools for ML libraries

Express DB Dockerfile (`express-db-server/Dockerfile`)

```
FROM node:20-slim

# Install system dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
    curl ca-certificates \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN if [ -f package-lock.json ]; then npm ci --only=production; else npm
install --only=production; fi

# Copy application source code
COPY . .

# Create non-root user for security
RUN groupadd -r expressuser && useradd -r -g expressuser expressuser
RUN chown -R expressuser:expressuser /app
USER expressuser

# Expose port
EXPOSE 3001

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:3001/health || exit 1

# Start the application
CMD ["npm", "start"]
```

PROF

Features:

- **Slim Base Image**: Minimal Node.js runtime
- **Production Dependencies**: Only installs production packages
- **Security**: Non-root user execution
- **Health Monitoring**: Built-in health checks

Environment Configuration

Environment Variables Structure

The application uses a comprehensive `.env` file for configuration across all services.

Core Configuration

```
```

```
# Supabase Configuration
SUPABASE_URL=https://your-project.supabase.co
SUPABASE_URL_PUBLIC=https://your-project.supabase.co
SUPABASE_ANON_KEY=your-anon-key-here
SUPABASE_SERVICE_ROLE_KEY=your-service-role-key-here

# Database Configuration
DATABASE_URL=postgresql://postgres:password@postgres:5432/postgres
POSTGRES_DB=postgres
POSTGRES_USER=postgres
POSTGRES_PASSWORD=secure_password_here

# JWT Configuration
JWT_SECRET=your-jwt-secret-key-minimum-32-characters-long

# Server Configuration
NODE_ENV=development
FRONTEND_URL=http://localhost:3000
EXPRESS_DB_URL=http://localhost:3001
FAST_API_URL=http://localhost:8000
```

Service-Specific Variables

Express DB Server:

```
PORT=3001
ALLOWED_ORIGINS=http://localhost:3000,http://127.0.0.1:3000
RATE_LIMIT_WINDOW_MS=900000
RATE_LIMIT_MAX_REQUESTS=1000
```

PROF

FastAPI AI Server:

```
PYTHONPATH=/app
PYTHONUNBUFFERED=1
EXPRESS_DB_URL=http://express-db-server:3001
DATA_DIR=/app/data
OPENAI_API_KEY=your-openai-api-key-here
ANTHROPIC_API_KEY=your-anthropic-api-key-here
MAX_WORKERS=4
AI_MODEL_PATH=/app/models
```

Frontend:

```
NEXT_PUBLIC_API_URL=http://localhost:80/api
NEXT_PUBLIC_AI_API_URL=http://localhost:80/ai
```

Local Development Configuration

For local development with Supabase local:

```
# Supabase Local URLs
SUPABASE_URL=http://kong:8000  # For backend services
SUPABASE_URL_PUBLIC=http://127.0.0.1:54321  # For frontend
SUPABASE_ANON_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...  # Local key
SUPABASE_SERVICE_ROLE_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...  #
Local key
DATABASE_URL=postgresql://postgres:postgres@supabase_db_Ai-Research-
Assistant-local:5432/postgres
```

Startup Scripts

Production Startup (`start-services.sh`)

```
#!/bin/bash
set -e

# Define colors for better output
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
RED='\033[0;31m'
NC='\033[0m' # No Color

echo -e "${GREEN}=== Starting Research Assistant Services ===${NC}"

# Check if Docker is running
if ! docker info > /dev/null 2>&1; then
  echo -e "${RED}Docker is not running. Please start Docker and try
again.${NC}"
  exit 1
fi

# Create SSL directory if it doesn't exist
if [ ! -d "./ssl" ]; then
  echo -e "${YELLOW}Creating SSL directory for future certificate
use${NC}"
  mkdir -p ./ssl
fi

# Check if .env file exists
if [ ! -f "./.env" ]; then
  echo -e "${RED}Error: .env file not found.${NC}"
  # Creates sample .env file
  exit 1
```

```
fi

# Start services using docker-compose
echo -e "${GREEN}Starting all services with Docker Compose...${NC}"
docker-compose down
docker-compose up --build -d

# Wait for services to be ready
echo -e "${YELLOW}Waiting for services to start...${NC}"
sleep 5

# Check service status
# (Service status checks for each container)

echo -e "${GREEN}=== Setup complete! ===${NC}"
echo -e "${GREEN}Visit: http://bruhmain.3utilities.com/${NC}"
```

Development Startup (`start.sh`)

```
npx supabase db reset
docker-compose down
docker-compose up
```

Simple development workflow:

1. Reset Supabase database
2. Stop existing containers
3. Start all services with rebuild

Health Checks and Monitoring

PROF

Container Health Checks

All services implement health checks at the container level:

Express DB Server:

```
curl -f http://localhost:3001/health
```

FastAPI AI Server:

```
curl -f http://localhost:8000/health
```

Nginx:

```
wget -qO- http://localhost/health
```

Health Check Configuration

```
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost:PORT/health"]
  interval: 30s      # Check every 30 seconds
  timeout: 10s       # 10 second timeout
  retries: 3         # 3 consecutive failures = unhealthy
  start_period: 40s  # Grace period during startup
```

Monitoring Service Status

```
# Check all services
docker-compose ps

# Check specific service
docker-compose ps express-db-server

# View logs
docker-compose logs -f [service-name]

# Check health status
docker inspect --format='{{.State.Health.Status}}' express-db-server
```

Development Workflows

Local Development Setup

PROF

1. **Clone and Setup**:

   ```
   git clone <repository>
   cd Ai-Research-Assistant-local
   cp .env.example .env
   # Edit .env with your configuration
   ```

2. **Start Supabase Local** (if using):

   ```
   npx supabase start
   ```

3. **Start All Services**:

```
./start-services.sh
# OR for development with database reset:
./start.sh
```

4. **Access Services**:
    - Frontend: http://localhost:3000
    - Express DB API: http://localhost:3001
    - FastAPI AI: http://localhost:8000
    - Nginx Proxy: http://localhost:80

Development with Live Reload

The Docker setup supports live reload for development:

- **Frontend**: Next.js hot reload via volume mount
- **Express Server**: Nodemon watches for changes
- **FastAPI**: Uvicorn reload mode enabled
- **Volume Mounts**: Source code mounted for instant updates

Building Individual Services

```
# Build specific service
docker-compose build express-db-server
docker-compose build fastapi-ai-server
docker-compose build frontend

# Build all services
docker-compose build

# Force rebuild without cache
docker-compose build --no-cache
```

PROF

Production Deployment

Production Environment Variables

```
NODE_ENV=production
LOG_LEVEL=warn
DEBUG=false
VERBOSE_LOGGING=false

# Production URLs
SUPABASE_URL=https://your-project.supabase.co
FRONTEND_URL=https://your-domain.com
```

```
EXPRESS_DB_URL=https://your-domain.com/api
FAST_API_URL=https://your-domain.com/ai
```

Production Dockerfile Modifications

For production, consider these modifications:

Frontend Production Build

```
FROM node:20 AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/.next ./.next
COPY --from=builder /app/public ./public
COPY --from=builder /app/package*.json ./
RUN npm ci --only=production
EXPOSE 3000
CMD ["npm", "start"]
```

Backend Production Build

```
# Add production optimizations
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1
# Remove --reload flag
CMD ["uv", "run", "uvicorn", "app.main:app", "--host", "0.0.0.0", "--
port", "8000"]
```

PROF

SSL Certificate Setup

1. **Obtain SSL Certificates**:

   ```
   # Using Let's Encrypt
   certbot certonly --webroot -w ./ssl -d your-domain.com
   ```

2. **Update nginx.conf**:

```
server {
    listen 443 ssl http2;
    ssl_certificate /etc/nginx/ssl/fullchain.pem;
    ssl_certificate_key /etc/nginx/ssl/privkey.pem;
    # ... rest of configuration
}
```

3. **Mount Certificates**:

```
volumes:
  - ./ssl:/etc/nginx/ssl:ro
  - /etc/letsencrypt:/etc/letsencrypt:ro
```

Performance Optimization

Resource Limits

Add resource limits for production:

```
services:
  express-db-server:
    deploy:
      resources:
        limits:
          cpus: '0.5'
          memory: 512M
        reservations:
          cpus: '0.25'
          memory: 256M

  fastapi-ai-server:
    deploy:
      resources:
        limits:
          cpus: '2.0'
          memory: 2G
        reservations:
          cpus: '1.0'
          memory: 1G
```

PROF

Caching Strategies

1. **Docker Layer Caching**: Dependencies installed before code copy
2. **Node Modules**: Anonymous volumes for better performance
3. **AI Models**: Persistent volume for model caching

Network Optimization

1. **Internal Communication**: Services use internal Docker network
2. **External Access**: Only nginx exposes public ports
3. **Service Discovery**: Docker DNS for service resolution

Troubleshooting

Common Issues

Service Won't Start

```
# Check logs
docker-compose logs [service-name]

# Check container status
docker-compose ps

# Restart specific service
docker-compose restart [service-name]
```

Port Conflicts

```
# Check port usage
netstat -tlnp | grep :3000
lsof -i :3000

# Kill process using port
sudo kill -9 $(lsof -t -i:3000)
```

PROF

Permission Issues

```
# Fix ownership (Linux/Mac)
sudo chown -R $USER:$USER ./

# Fix permissions for volumes
docker-compose exec fastapi-ai-server chown -R aiuser:aiuser /app
```

Network Issues

```
# Recreate networks
docker-compose down
docker network prune
```

```
docker-compose up

# Check network connectivity
docker-compose exec express-db-server ping fastapi-ai-server
```

Debugging Commands

```
# Enter container shell
docker-compose exec express-db-server bash
docker-compose exec fastapi-ai-server bash

# Check environment variables
docker-compose exec express-db-server env

# Monitor resource usage
docker stats

# View container processes
docker-compose top
```

Log Management

```
# View all logs
docker-compose logs

# Follow logs in real-time
docker-compose logs -f

# View specific service logs
docker-compose logs -f express-db-server

# View last N lines
docker-compose logs --tail=50 fastapi-ai-server
```

PROF

Security Considerations

Container Security

1. **Non-root Users**: All services run as non-root users
2. **Read-only Mounts**: Configuration files mounted as read-only
3. **Resource Limits**: Prevents resource exhaustion attacks
4. **Health Checks**: Early detection of compromised containers

Network Security

1. **Internal Network**: Services communicate via private network

2. **Minimal Exposure**: Only nginx exposed to public internet
3. **Service Isolation**: Each service in separate container

Secrets Management

1. **Environment Files**: Sensitive data in `.env` files
2. **Docker Secrets**: For production deployment
3. **File Permissions**: Restrict access to configuration files

Best Practices

Development

1. **Use Volume Mounts**: For live reload during development
2. **Environment Separation**: Different configs for dev/prod
3. **Health Checks**: Monitor service health continuously
4. **Resource Limits**: Prevent runaway processes

Production

1. **Multi-stage Builds**: Optimize image sizes
2. **Security Scanning**: Regular vulnerability scans
3. **Backup Strategies**: Data persistence and recovery
4. **Monitoring**: Comprehensive logging and metrics
5. **Rolling Updates**: Zero-downtime deployments

Maintenance

1. **Regular Updates**: Keep base images updated
2. **Image Cleanup**: Remove unused images and containers
3. **Log Rotation**: Prevent disk space issues
4. **Performance Monitoring**: Track resource usage

PROF This Docker configuration provides a robust, scalable, and maintainable foundation for the AI Research Assistant application, supporting both development workflows and production deployments.