

# Docker Deployment Guide (Standalone)

This guide details how to deploy the **WhatsApp eTIMS** application using Docker and Docker Compose. It consolidates architectural insights, security requirements, and operational steps from our technical documentation.

---

## 1. Project Overview

Item	Details
<b>Application Name</b>	WhatsApp eTIMS
<b>Technology Stack</b>	Next.js 16, React 19, TypeScript, TailwindCSS 4
<b>Delivery Channel</b>	WhatsApp WebView
<b>Target Users</b>	Non-VAT Registered Taxpayers
<b>Backend API</b>	<a href="https://kratest.pesaflow.com/api/ussd">https://kratest.pesaflow.com/api/ussd</a>
<b>Container Format</b>	Docker Image (Linux/Node.js)

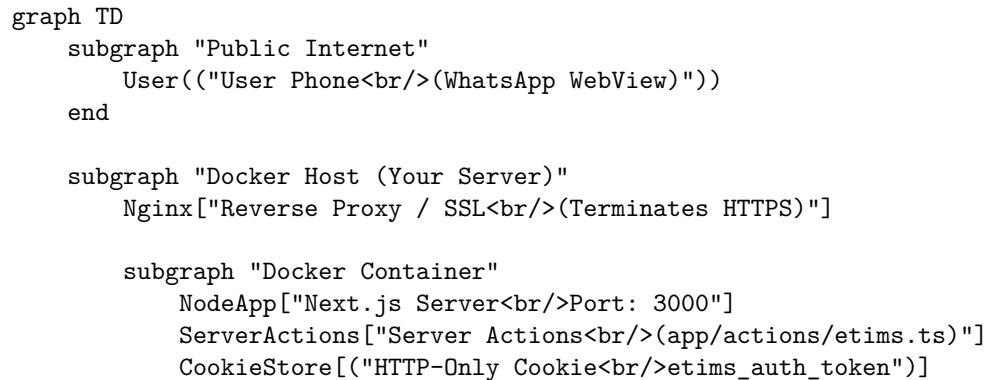
### Core Features

- **eTIMS Sales Invoicing** - Create and send tax invoices via WhatsApp
  - **Credit Notes** - Full and partial credit note processing
  - **Buyer-Initiated Invoices** - Buyer creates, Seller approves workflow
- 

## 2. System Architecture

The application operates as a **Secure Gateway** (BFF Pattern) between WhatsApp users and the PesaFlow APIs. It is designed to be stateless and containerized.

### High-Level Data Flow



```

        end
    end

    subgraph "PesaFlow API"
        subgraph "Authentication"
            InitAPI【"/init<br/>Check Registration"】
            OtpAPI【"/otp<br/>Generate OTP"】
            ValidateAPI【"/validate-otp<br/>Returns Token"】
        end

        subgraph "Business Operations"
            LookupAPI【"/buyer-initiated/lookup"】
            PostSaleAPI【"/post-sale"】
            CreditNoteAPI【"/credit-note"】
            FetchInvoicesAPI【"/fetch-invoices"】
        end
    end

    %% User Request Flow
    User -->|"1. HTTPS Request<br/>?phone=254..."| Nginx
    Nginx -->|"2. Proxy to :3000"| NodeApp

    %% Internal Server Flow
    NodeApp -->|"3. Invoke"| ServerActions
    ServerActions -->|"4. Read/Write Token"| CookieStore

    %% Auth Flow (no token required)
    ServerActions -.->|"No Auth Required"| InitAPI
    ServerActions -.->|"No Auth Required"| OtpAPI
    ServerActions -.->|"No Auth Required"| ValidateAPI

    %% Business Flow (token required)
    ServerActions -->|"Bearer Token"| LookupAPI
    ServerActions -->|"Bearer Token"| PostSaleAPI
    ServerActions -->|"Bearer Token"| CreditNoteAPI
    ServerActions -->|"Bearer Token"| FetchInvoicesAPI

    style User fill:#ffe0b2,stroke:#f57c00,stroke-width:2px
    style NodeApp fill:#d4f1f4,stroke:#00796b,stroke-width:2px
    style ServerActions fill:#c8e6c9,stroke:#388e3c,stroke-width:2px
    style CookieStore fill:#fff9c4,stroke:#fbc02d,stroke-width:2px
    style ValidateAPI fill:#c8e6c9,stroke:#388e3c,stroke-width:2px

```

## Authentication Flow (OTP → Token)

This diagram shows the login process: check status → send OTP → verify OTP → get token.

```
sequenceDiagram
    participant Browser as User Browser
    participant NextJS as Next.js Server
    participant Cookie as Cookie Store
    participant API as PesaFlow API

    Note over Browser, API: Step 1: Check User Registration
    Browser->>NextJS: checkUserStatus("254712345678")
    NextJS->>API: POST /init<br/>{ "msisdn": "254712345678" }
    API-->>NextJS: { "code": 1, "has_etims": true, "has_vat": false, "name": "John Doe" }
    NextJS-->>Browser: { success: true, isRegistered: true }

    Note over Browser, API: Step 2: Request OTP
    Browser->>NextJS: generateOTP("254712345678")
    NextJS->>API: POST /otp<br/>{ "msisdn": "254712345678" }
    API-->>NextJS: { "message": "OTP sent successfully" }
    NextJS-->>Browser: { success: true }

    Note over Browser, API: Step 3: Verify OTP & Get Token
    Browser->>NextJS: verifyOTP("254712345678", "ABC123")
    NextJS->>API: POST /validate-otp<br/>{ "msisdn": "254712345678", "otp": "ABC123" }
    API-->>NextJS: { "code": 1, "token": "eyJhbGciOiJI... ", "message": "OTP verified" }

    NextJS->>Cookie: Set 'etims_auth_token' (HTTP-Only)
    NextJS-->>Browser: { success: true }

    Note over Browser: User is now authenticated
```

## Business Operation Flow (with Token)

This diagram shows an authenticated API call to submit an invoice.

```
sequenceDiagram
    participant Browser as User Browser
    participant NextJS as Next.js Server
    participant Cookie as Cookie Store
    participant API as PesaFlow API

    Note over Browser, NextJS: User clicks "Submit Invoice"
    Browser->>NextJS: submitInvoice({ msisdn, items, total_amount })

    NextJS->>Cookie: cookies().get('etims_auth_token')
```

```
Cookie-->>NextJS: "eyJhbGciOiJI..."  
  
NextJS->>NextJS: Build headers:<br/>Authorization: Bearer {token}<br/>x-source-for: what  
  
NextJS->>API: POST /post-sale<br/>Headers: Authorization: Bearer eyJhbGc...<br/>Body: {  
  API-->>NextJS: { "code": 8, "invoice_no": "INV-001", "invoice_pdf_url": "https://...." }  
  
NextJS-->>Browser: { success: true, invoice_id: "INV-001" }  
  
Note over Browser: UI updates with invoice PDF link
```

---

### 3. API Endpoints Integrated

All API calls route through the upstream API.

[IMPORTANT] **Test Environment:** <https://kratest.pesaflow.com/api/ussd>  
**Production Environment:** <https://ecitizen.kra.go.ke/api/ussd>

Ensure the correct URL is set in NEXT\_PUBLIC\_API\_BASE\_URL for your deployment.

Endpoint	Description
lookup	Customer/PIN verification
submit_invoice	Create sales invoice
credit_note	Submit credit notes
fetch_invoices	Retrieve buyer-initiated invoices
process_buyer_invoice	Accept/reject buyer invoices
check_user_status	Verify eTIMS registration
register_taxpayer	Register for eTIMS service
generate_otp / verify_otp	Authentication

#### Request Headers

All requests from Server Actions include:

Authorization: Bearer <etims\_auth\_token>  
x-source-for: whatsapp  
Content-Type: application/json

---

### 4. WhatsApp Integration

#### Notification Types

Type	Trigger	Purpose
<code>etims_invoice</code>	Sales invoice created	Send Invoice PDF to seller/buyer
<code>etims_credit_note</code>	Credit note submitted	Send Credit Note PDF
<code>etims_buyer_pending</code>	Buyer invoice created	Notify buyer for approval
<code>etims_buyer_action</code>	Buyer accepts/rejects	Notify seller of decision

### WhatsApp API Configuration

- Uses Meta WhatsApp Business Cloud API
  - Sends PDFs as document attachments
  - Sends text notifications for status updates
- 

## 5. Application Routes

Route	Description
/	Main services dashboard
/etims	eTIMS invoicing module
/etims/auth/*	Authentication (login, OTP)
/etims/sales-invoice/*	Sales invoice wizard
/etims/credit-note/*	Credit note flow
/etims/buyer-initiated/*	Buyer/Seller invoice flow

---

## 6. Security Considerations

Feature	Implementation
<b>HTTP-Only Cookies</b>	Auth tokens stored in <code>etims_auth_token</code> cookie, inaccessible to JavaScript.
<b>Server-Side Secrets</b>	API keys and base URLs are only available on the server ( <code>process.env</code> ).
<b>VAT Restriction</b>	Hard block prevents VAT-registered taxpayers from using the service.
<b>Token Injection</b>	All API requests are authenticated server-side in <code>app/actions/</code> .
<b>Context Injection</b>	Phone number passed via URL query parameter ( <code>?phone=254...</code> ).

---

## 7. Deployment Artifacts & Prerequisites

### Required Software

- **Docker Engine:** (v20.10+)
- **Docker Compose:** (v2.0+)

### Environment Variables

Create a `.env.production` file with these values. An example template is provided in the project:

**Template:** `env.example`

Variable	Description	Example Value
HOST_PORT	The external port to expose the application on.	3000
WHATSAPP_PHONE_NUMBER	Meta Business ID for the phone number.	5896221609...
WHATSAPP_ACCESS_TOKEN	System User Token with messaging permissions.	EAA...
NEXT_PUBLIC_WHATSAPP_NUMBER	Display number (no + sign).	254708427694
NEXT_PUBLIC_API_BASE_URL	Base URL for the upstream API.	<a href="https://ecitizen.kra.go.ke/api/ussd">https://ecitizen.kra.go.ke/api/ussd</a>

```
# Copy the template to create your production config
cp env.example .env.production

[!NOTE] For testing, use https://kratest.pesaflow.com/api/ussd
as the NEXT_PUBLIC_API_BASE_URL. For production, use
https://ecitizen.kra.go.ke/api/ussd.
```

### Network Requirements

Category	Requirement
<b>Subdomain</b>	A-Record for <code>whatsapp.kra.go.ke</code> (or your domain) pointing to your server.
<b>Firewall (Outbound)</b>	Allow HTTPS (Port 443) to <code>kratest.pesaflow.com</code> .
<b>Firewall (Inbound)</b>	Allow HTTPS (Port 443) from public internet to Nginx/Load Balancer.

---

Category	Requirement
<b>TLS Certificate</b>	Valid SSL certificate for your domain (e.g., via Let's Encrypt).

---

## 8. Docker Compose Setup

A `docker-compose.yml` file is provided in the project.

The port mapping uses `HOST_PORT:-3000:3000`. This means: use the value of `HOST_PORT` from `.env.production`, or default to 3000 if not set. Change `HOST_PORT` in your `.env.production` file to use a different port (e.g., 8080).

---

## 9. Operational Workflow

```
sequenceDiagram
    participant Admin as System Admin
    participant Server as Docker Host
    participant App as App Container

    Note over Admin, Server: 1. Setup Phase
    Admin->>Server: Copy source code or pull image
    Admin->>Server: Create .env.production
    Admin->>Server: Create docker-compose.yml

    Note over Admin, Server: 2. Build & Deploy
    Admin->>Server: docker compose build --no-cache
    Server->>Server: Building Docker Image...
    Admin->>Server: docker compose up -d
    Server->>App: Start Container

    Note over App: 3. Startup Verification
    App->>App: Load Environment Vars
    App->>App: Start Next.js Server

    Note over Admin, App: 4. Monitoring
    Admin->>App: docker compose logs -f
```

---

## 10. Step-by-Step Deployment Instructions

### Method 1: Using Pre-built Image

#### Step 1: Prepare the Environment

```
mkdir etims  
cd etims
```

#### Step 2: Configure Secrets

```
nano .env.production  
# Paste the variables listed in Section 7
```

#### Step 3: Pull and Run the Image

```
# Pull the latest image  
docker pull ghcr.io/chatnationwork/etims-app:latest  
  
# Run the container with environment variables  
docker run -d \  
  -p 3000:3000 \  
  --name etims-app \  
  --env-file .env.production \  
  ghcr.io/chatnationwork/etims-app:latest
```

#### Step 4: Verify Deployment

```
# Check if container is running  
docker ps  
  
# View application logs  
docker logs etims-app  
  
# Test the application  
curl http://localhost:3000/etims
```

#### Step 5: Stop/Restart

```
# Stop the application  
docker stop etims-app  
  
# Start the application  
docker start etims-app  
  
# Remove the container  
docker rm etims-app
```

## Method 2: Build from Source

### Step 1: Prepare the Environment

```
git clone https://github.com/chatnationwork/etims.git  
cd etims
```

### Step 2: Configure Secrets

```
nano .env.production  
# Paste the variables listed in Section 7 above.
```

### Step 3: Build and Run

```
docker compose build  
docker compose up -d
```

---

## 11. File Structure

```
etims/  
  app/  
    _components/      # Shared UI components  
    actions/          # Server actions (API proxy)  
      etims.ts        # Main API proxy logic  
    etims/             # eTIMS invoicing module  
    docs/              # Documentation  
    scripts/           # Test scripts  
    Dockerfile         # Container build instructions  
    env.example        # Environment template  
    package.json       # Dependencies
```

---