TMSA Integration System

ETC and BerthPlan Deployment & Configuration Documentation

Document Version:	**1.0**
Date:	February 26, 2026
Application:	TMSA ETC & BerthPlan Integration
Environment:	Pre-Production & Production
Server:	SCRBITBAMATNG02

Document History

This document and its contents have been prepared and are intended solely as information for Terminals, Global IT, SD&I and use in relation to Terminal Infrastructure Platform Refresh Project – TC1

Version No	Revision Comments	Author	Approved by	Date
1.0		BIH001		February 25, 2026

Table of Contents

- 1. Overview
- 2. System Architecture
- 2.1 ETC Handler (etc_handler_api.py)
- 2.2 BerthPlan Handler (bp_handler_api.py)
- 2.3 Berth Metric Calculator (metrics.py)
- 2.4 XML Builder (xml_builder.py)
- 2.5 Main Application (main.py)
- 3. Deployment Steps
- 3.1 Pre-requisites
- 3.2 Deployment Process
- 4. Configuration
- 4.1 Environment Variables
- 4.2 Database Configuration
- 4.3 API Configuration
- 5. File Structure
- 6. Data Flow
- 7. API Endpoints & Integration
- 8. Background Jobs & Scheduling
- 9. XML Generation Process
- 10. Troubleshooting

1. Overview

The TMSA Integration System is an automated data exchange solution that connects the APMT (APM Terminals) N4 terminal operating system with the TMSA (Terminal Management System Architecture) platform. The system handles two critical operations: ETC (Estimated Time of Completion) updates and BerthPlan synchronization.

Purpose

This system automates the bidirectional data flow between APMT's terminal operations and TMSA's port management system, ensuring real-time visibility of vessel operations, berth planning, and commercial operation timelines.

Key Features

- Automated ETC (Estimated Time of Completion) data extraction and transmission
- Real-time berth plan synchronization with vessel schedules
- OAuth2 token-based authentication for secure API access
- Dynamic berth metric calculation using bollard positioning
- SOAP/XML-based data exchange with external systems
- Automated XML file archival and cleanup (30-day retention)
- Marine agent mapping for accurate vessel agency information
- Support for both starboard and port-side berthing configurations

Technology Stack

Component	Technology
Programming Language	Python 3.x
Database	Microsoft SQL Server (N4/Sparcsn4)
API Protocol	REST (OAuth2), SOAP/XML
HTTP Client	httpx (async)
Database Driver	pyodbc (ODBC Driver 17)
Scheduling	Cron / Systemd Timer

2. System Architecture

The TMSA Integration System consists of five main components working together to extract, transform, and transmit terminal operation data:

2.1 ETC Handler (etc_handler_api.py)

Handles the extraction of Estimated Time of Completion (ETC) data from the N4 database and transmission to TMSA via SOAP/XML.

Key Responsibilities

- Database connection management using pyodbc
- SQL query execution to extract vessel commercial operation data
- XML generation with vessel voyage details, IMO, ETC timestamps
- SOAP envelope wrapping for TMSA transmission
- HTTP POST with basic authentication to TMSA endpoint
- Local XML archival in /SOAP_Archive directory

Database Query Logic

The ETC handler executes a complex SQL query that:

- Joins inv_wi (work instructions) with carrier visit tables
- Filters for LOAD/DSCH moves that are not complete
- Includes only vessels in '40WORKING' phase (actively working)
- Extracts vessel name, voyage number, IMO, and maximum ETC
- Generates XML-formatted output using SQL's FOR XML PATH

Key Methods

Method	Purpose
sqlConnection()	Establishes ODBC connection to N4 database
read_data()	Executes SQL query and fetches ETC data
get_etc_xml()	Orchestrates data retrieval and returns XML
send_xml()	Sends SOAP/XML to TMSA endpoint via HTTP POST

2.2 BerthPlan Handler (bp_handler_api.py)

Manages the retrieval of berth plan data from an external REST API and transmission to TMSA.

Key Responsibilities

- OAuth2 token acquisition using client credentials flow
- REST API calls to fetch berth plan data (48-day window: -8 to +40 days)
- Token expiration monitoring and logging
- JSON response parsing and local storage
- XML generation via xml_builder module
- SOAP/XML transmission to TMSA endpoint
- Error handling and comprehensive logging

OAuth2 Authentication Flow

1. POST request to token endpoint with client_id and client_secret

2. Receive access_token with limited validity period

3. Include Bearer token in Authorization header for API requests

4. Monitor token expiration date (configurable via BP_TOKEN_EXP_DATA)

5. Log remaining days until token expiration

API Request Parameters

Parameter	Value
terminal	MAPTMTM
fromDate	Current date - 8 days
toDate	Current date + 40 days

2.3 Berth Metric Calculator (metrics.py)

Calculates precise berthing positions (fore and aft metric points) based on bollard positions and vessel dimensions.

Bollard Mapping System

The system uses a pre-configured bollard map with the following parameters:

Parameter	Value
Start Index	0 meters
End Index	1585 meters
Last Bollard	B81
Bollard Spacing	20 meters

Metric Calculation Logic

Two calculation modes are supported:

1. Real Metrics (Bollard-Based): When a valid bollard (e.g., B45.5) is provided:

- Parse bollard name (B45) and offset (0.5 = 10 meters)
- Look up base position from berth_map dictionary
- Calculate fore position: base + offset
- Calculate aft position: fore ± LOA (depending on berthing side)
- Starboard: aft = fore + LOA
- Port: aft = fore - LOA

2. Mock Metrics (Default): When no bollard is provided:

- Use end_index (1585m) as reference point
- Starboard: fore = 1585 - LOA, aft = 1585
- Port: fore = 1585, aft = 1585 - LOA
- Marks berth as 'mock' for visibility in reports

2.4 XML Builder (xml_builder.py)

Generates SOAP-compliant XML messages for berth plan data transmission to TMSA.

Key Features

- SOAP envelope structure with proper namespaces
- XML header with version, timestamp, and sender information
- Body section with date range and berth information
- Marine agent mapping from operator codes to agency names
- DateTime formatting to ISO 8601 with timezone (YYYY-MM-DDTHH:MM:SS.000Z)
- Berthing side conversion (1 = Starboard, 0 = Port)
- Total moves calculation (load + discharge + shifting)
- ISPS security certificate placeholder fields

Marine Agent Mapping

The system includes a hardcoded mapping of operator codes to full marine agent names. Examples:

Operator Code	Marine Agent
MSK	NOATUM (Ex MARMEDSA)
MSC	TRANSPORTS MAROCAINS
CGM	CMA CGM MAROC
HAP	ARKAS MAROC
XCL	BOLLORE TRANSPORT ET LOGISTICS MAROC

XML Structure

```
<soapenv:Envelope>
  <soapenv:Header/>
  <soapenv:Body>
    <tmsa:processBerthPlan>
      <berthPlanRequest>
        <DemandeInitiale>
          <header>
            <msgVersion>3.0</msgVersion>
            <GenerationTime>2026-02-26T10:20:00Z</GenerationTime>
            <sender>APMT</sender>
          </header>
          <body>
            <startDate>2026-02-18</startDate>
            <endDate>2026-04-07</endDate>
            <berths>
              <berthinformation>
                <!-- Vessel and berth details -->
              </berthinformation>
            </berths>
          </body>
        </DemandeInitiale>
      </berthPlanRequest>
    </tmsa:processBerthPlan>
  </soapenv:Body>
```

```
</soapenv:Envelope>
```

2.5 Main Application (main.py)

Orchestrates the complete data flow from extraction to transmission.

Execution Flow

Step 1: ETC Data Extraction: Call etc_handler.get_etc_xml() to fetch vessel ETC data

Step 2: ETC SOAP Wrapping: Wrap ETC XML in SOAP envelope with processETC operation

Step 3: ETC Transmission: Send ETC SOAP message to TMSA endpoint

Step 4: ETC Archival: Save ETC XML to /SOAP_Archive with timestamp

Step 5: Archive Cleanup: Delete XML files older than 30 days

Step 6: BerthPlan Generation: Call bp_handler.metrics_handler() to fetch and process berth data

Step 7: BerthPlan Transmission: Send BerthPlan SOAP message to TMSA endpoint

Archive Management

The system automatically manages XML archives:

- Archive directory: ./SOAP_Archive/
- Filename format: APMT_ETC_YYYYMMDD_HH:MM:SS.xml
- Retention policy: 30 days
- Cleanup runs automatically after each ETC transmission
- Cleanup based on file modification time, not creation time

3. Deployment Steps

3.1 Pre-requisites

Item	Requirement	Notes
Operating System	Linux (Ubuntu 24.04 LTS)	Server: SCRBITBAMATNG02
Python Version	Python 3.8+	With asyncio support
Database Access	SQL Server 2016+	ODBC Driver 17 for SQL Server
Network Access	HTTPS outbound	Access to TMSA endpoints
User Permissions	admin	Write access to /home/Projects/TMSA
Disk Space	Minimum 10GB	For XML archives and logs

Required System Packages

```
# Install ODBC driver for SQL Server
curl https://packages.microsoft.com/keys/microsoft.asc | sudo apt-key add -
curl https://packages.microsoft.com/config/ubuntu/24.04/prod.list | \
    sudo tee /etc/apt/sources.list.d/mssql-release.list
sudo apt update
sudo ACCEPT_EULA=Y apt install -y msodbcsql17 unixodbc-dev

# Verify ODBC installation
odbcinst -j
```

Required Python Packages

```
pip install pyodbc httpx python-dotenv
```

3.2 Deployment Process

Step 1: Create Project Directory

Set up the application directory structure.

```
sudo mkdir -p /home/Projects/TMSA
sudo chown root:root /home/Projects/TMSA
cd /home/Projects/TMSA
```

Step 2: Deploy Application Files

Copy all Python scripts to the project directory.

```
# Copy files:
# - main.py
# - etc_handler_api.py
# - bp_handler_api.py
# - metrics.py
# - xml_builder.py
```

Step 3: Create Archive Directories

Set up directories for XML storage.

```
mkdir -p SOAP_Archive
mkdir -p ETC_Archive
mkdir -p log
```

Step 4: Configure Environment Variables

Create .env file with credentials.

```
cat > .env << 'EOF'
# Database Configuration
DB_DATA_SOURCE=10.212.12.68
DB_PORT=1433
DB_INITIAL_CATALOG=Sparcsn4
DB_USER_ID=BI_END_USER
DB_PASSWORD=********

# ETC Configuration
ETC_URI=https://tmsa-endpoint/etc
ETC_AUTH_USER=<etc_user_to be provided by TC1 Port Authority>
ETC_AUTH_PASSWORD=<etc_password to be provided by TC1 Port Authority>

# BerthPlan OAuth2 Configuration
CLIENT_ID=your_client_id
CLIENT_SECRET=your_client_secret
SCOPE=your_scope
CONSUMER_KEY=your_consumer_key
TOKEN_URL=https://oauth-endpoint/token
BP_TOKEN_EXP_DATA=yyyy-mm-dd

# BerthPlan API Configuration
BP_URL=https://api-endpoint/berthplan
BP_XML_SEND_URL=https://tmsa-endpoint/berthplan
BP_AUTH_USER=<etc_user_to be provided by TC1 Port Authority>
BP_AUTH_PASSWORD=<etc_password to be provided by TC1 Port Authority>
EOF

chmod 600 .env
```

Step 5: Test Database Connectivity

Verify connection to N4 database.

```
python3 -c "import pyodbc; from dotenv import load_dotenv; import os;
load_dotenv(); conn = pyodbc.connect(f'DRIVER={{ODBC Driver 17 for SQL
Server}};SERVER={os.getenv("DB_DATA_SOURCE")};DATABASE={os.getenv("DB_INITIAL_C
ATALOG")};UID={os.getenv("DB_USER_ID")};PWD={os.getenv("DB_PASSWORD")}');
print('Database connection successful!')"
```

Step 6: Test Individual Components

Test each handler independently.

```
# Test ETC handler
python3 etc_handler_api.py

# Test BerthPlan handler
python3 bp_handler_api.py
```

Step 7: Test Complete Flow

Run the main application manually.

```
python3 main.py
```

Step 8: Configure Cron Job

Schedule automated execution.

```
# Edit crontab
crontab -e

# Add entry (runs every 4 hours)
0 */4 * * * cd /home/Projects/TMSA && /usr/bin/python3 main.py >>
cron_output.log 2>&1
```

4. Configuration

4.1 Environment Variables

All configuration is managed through environment variables stored in the .env file. The application validates all required variables on startup.

Database Configuration

Variable	Purpose	Example
DB_DATA_SOURCE	N4 database server IP	10.212.12.68
DB_PORT	Database port	1433
DB_INITIAL_CATALOG	Database name	Sparcsn4
DB_USER_ID	Database username	DB_USER
DB_PASSWORD	Database password	******

ETC Configuration

Variable	Purpose	Example
ETC_URI	TMSA ETC endpoint URL	https://tmsa.example.com/etc
ETC_AUTH_USER	Basic auth username	etc_service
ETC_AUTH_PASSWORD	Basic auth password	******

BerthPlan OAuth2 Configuration

Variable	Purpose	Example
CLIENT_ID	OAuth2 client identifier	client_abc123
CLIENT_SECRET	OAuth2 client secret	[REDACTED]
SCOPE	OAuth2 requested scope	api.read api.write
CONSUMER_KEY	Additional API key	consumer_xyz789
TOKEN_URL	OAuth2 token endpoint	https://auth.example.com/token
BP_TOKEN_EXP_DATA	Token expiration date	yyyy-mm-dd

BerthPlan API Configuration

Variable	Purpose	Example
BP_URL	BerthPlan data API endpoint	https://api.example.com/berthplan
BP_XML_SEND_URL	TMSA BerthPlan endpoint	https://tmsa.example.com/berthplan
BP_XML_USER	BerthPlan auth username	bp_service
BP_XML_PASSWORD	BerthPlan auth password	[REDACTED]

5. File Structure

The application follows a modular structure with clear separation of concerns:

```
/home/Projects/TMSA/
├── main.py                          # Application entry point & orchestrator
├── etc_handler_api.py               # ETC data extraction & transmission
├── bp_handler_api.py                # BerthPlan API integration
├── metrics.py                       # Berth metric calculator
├── xml_builder.py                   # XML/SOAP message generator
├── .env                             # Environment configuration (credentials)
├── agentMap.json                    # Marine agent mapping reference
├── BerthPlan_data.json              # Latest BerthPlan API response
├── xml_file.xml                     # Latest generated BerthPlan XML
├── cron_output.log                  # Cron execution logs
├── ETC_SOAP_Archive/                # ETC XML archives (30-day retention)
│   └── APMT_ETC_YYYYMMDD_HH:MM:SS.xml
├── BP_SOAP_Archive/                # ETC XML archives (30-day retention)
│   └── APMT_BP_YYYYMMDD_HH:MM:SS.xml
├── log/                             # Application logs
└── __pycache__/                     # Python bytecode cache
```

Key Files Description

File	Purpose
main.py	Orchestrates ETC and BerthPlan workflows, manages archival
etc_handler_api.py	Database queries, ETC XML generation, SOAP transmission
bp_handler_api.py	OAuth2 authentication, BerthPlan API calls, XML transmission
metrics.py	Bollard mapping, metric point calculations
xml_builder.py	SOAP envelope generation, marine agent mapping
BerthPlan_data.json	Cached BerthPlan API response for troubleshooting
xml_file.xml	Latest generated BerthPlan XML for verification
cron_output.log	Cron job execution logs

6. Data Flow

The TMSA Integration System implements two parallel data flows:

6.1 ETC Data Flow

1. **Query Execution:** N4 Database (Sparcsn4) ← SQL Query with JOINs
2. **Data Extraction:** Extract vessel voyage, IMO, name, ETC timestamp
3. **XML Generation:** Convert SQL result set to XML format
4. **SOAP Wrapping:** Wrap XML in SOAP envelope with processETC operation
5. **Transmission:** HTTP POST to TMSA ETC endpoint with Basic Auth
6. **Local Archival:** Save timestamped XML to SOAP_Archive/
7. **Cleanup:** Delete XML files older than 30 days

6.2 BerthPlan Data Flow

1. **OAuth2 Authentication:** POST to token endpoint → Receive access_token
2. **API Request:** GET BerthPlan data with Bearer token (48-day window)
3. **JSON Storage:** Save API response to BerthPlan_data.json
4. **Metric Calculation:** Calculate fore/aft positions for each berth
5. **XML Generation:** Build SOAP envelope with berth information
6. **Agent Mapping:** Map operator codes to full marine agent names
7. **Transmission:** HTTP POST to TMSA BerthPlan endpoint with Basic Auth
8. **Local Storage:** Save generated XML to xml_file.xml

6.3 System Integration Points

The system integrates with multiple external systems:

System	Protocol	Direction	Purpose
N4 Database	ODBC/SQL	Inbound	Extract vessel and work instruction data
BerthPlan API	REST/OAuth2	Inbound	Retrieve berth schedules
TMSA ETC	SOAP/HTTP	Outbound	Send vessel completion estimates
TMSA BerthPlan	SOAP/HTTP	Outbound	Send berth planning data

8. Background Jobs & Scheduling

The TMSA Integration System runs as a scheduled job, not as a continuous service.

8.1 Cron Configuration

The application is typically scheduled via cron to run at regular intervals:

```
# Edit crontab
crontab -e

# Agreed schedules:

# */10 * * * * cd /home/Projects/TMSA/ && sudo /home/Projects/venv/bin/python
main.py  >> /home/Projects/TMSA/cron_output.log 2>&1
```

8.2 Job Execution Flow

Each cron execution performs the following:

1. Load environment variables from .env file
2. Initialize EtcHandler and BerthPlanHandler instances
3. Execute ETC workflow (extract, transform, send, archive)
4. Execute BerthPlan workflow (authenticate, fetch, calculate, send)
5. Clean up old XML archives (30-day retention)
6. Log all operations to cron_output.log
7. Exit with status code (0 = success, non-zero = error)

8.3 Monitoring & Alerting

Recommended monitoring practices:

- Monitor cron_output.log for ERROR or CRITICAL level messages
- Check BerthPlan API key expiration date
- Alert on HTTP 401/403 errors (authentication failures)
- Alert on HTTP 500 errors (server-side failures)
- Monitor OAuth2 token expiration date (BP_TOKEN_EXP_DATA)
- Alert when token has < 30 days remaining
- Monitor SOAP_Archive/ directory size (should not exceed 1GB)
- Verify XML files are being created in SOAP_Archive/
- Check database connectivity regularly

8.4 Background Jobs Summary Table

Job Name	Category	Description	Frequency
ETC Data Sync	Application	Extract vessel ETC from N4 and send to TMSA	Every 10 minutes
BerthPlan Sync	Application	Fetch berth plans and send to TMSA	Every 10 minutes
XML Archive Cleanup	Maintenance	Delete XML files older than 30 days	After each run
OAuth2 Token Refresh	Authentication	Automatic token acquisition on each run	Every run
Log Rotation	Maintenance	Rotate cron_output.log when > 10MB	Manual

9. XML Generation Process

The system generates two types of XML messages:

9.1 ETC XML Generation

ETC XML is generated directly from SQL using FOR XML PATH syntax:

- Header section with version, timestamp, sender
- Body section with commercial operation details per vessel
- Voyage number, vessel name, vessel code, IMO
- Maximum ETC timestamp from all pending work instructions
- Automatic XML formatting by SQL Server

9.2 BerthPlan XML Generation

BerthPlan XML is generated programmatically using Python's ElementTree:

- SOAP envelope with proper namespaces
- Header with message version (3.0), generation time, sender
- Body with date range (start_date, end_date)
- Berth information array with vessel and operational details
- Dynamic metric calculation for each berth
- Marine agent name lookup from operator code
- DateTime formatting to ISO 8601 with timezone

9.3 XML Validation

The system performs basic XML validation:

- Verify XML is well-formed before transmission
- Check required fields are not empty
- Validate DateTime formats
- Ensure numeric fields contain valid numbers
- Log any validation errors before sending

10. Troubleshooting

Common Issues and Solutions

Issue: Database connection failure

Solutions:

1. Verify DB_DATA_SOURCE, DB_USER_ID, DB_PASSWORD in .env file
2. Test connectivity: telnet 10.212.12.68 1433
3. Check ODBC driver: odbcinst -j
4. Verify user has SELECT permissions on Sparcsn4 database
5. Check firewall rules allow outbound TCP/1433

Issue: OAuth2 authentication failure (401 Unauthorized)

Solutions:

6. Verify CLIENT_ID and CLIENT_SECRET are correct
7. Check token expiration date (BP_TOKEN_EXP_DATA)
8. Test token endpoint manually: curl -X POST <TOKEN_URL>
9. Ensure SCOPE matches API requirements
10. Verify CONSUMER_KEY is valid
11. Check if token has expired and needs renewal

Issue: SOAP transmission failure (HTTP 500)

Solutions:

12. Verify ETC_URI and BP_XML_SEND_URL are correct
13. Check if TMSA endpoints are accessible
14. Validate XML syntax before sending
15. Review TMSA error logs for rejection reasons
16. Verify authentication credentials (ETC_AUTH_USER, BP_XML_USER)
17. Check if required XML fields are populated

Issue: No ETC data generated

Solutions:

18. Verify vessels are in '40WORKING' phase in N4
19. Check if work instructions exist with move_kind LOAD/DSCH
20. Ensure work instructions are not in COMPLETE stage
21. Query database manually to verify data availability
22. Check SQL query syntax in etc_handler_api.py

Issue: BerthPlan API returns empty data

Solutions:

23. Verify date range parameters (fromDate, toDate)
24. Check if terminal parameter is correct (MAPTMTM)
25. Ensure OAuth2 token has required scope
26. Verify Bearer token is included in Authorization header
27. Check API endpoint URL (BP_URL)
28. Review API response in BerthPlan_data.json

Issue: Incorrect berth metric calculations

Solutions:

29. Verify bollard naming convention (e.g., B45)
30. Check bollard_spacing parameter (default: 20m)
31. Ensure vessel LOA is provided in API response
32. Review berth_map dictionary for bollard positions
33. Verify isStarboardBerth flag is correct (1 or 0)
34. Check if planned_bollard starts with 'B'

Diagnostic Commands

Use these commands to diagnose application issues:

```
# Check cron job status
crontab -l | grep TMSA

# View recent logs
tail -100 /home/Projects/TMSA/cron_output.log

# Test database connectivity
python3 -c "import pyodbc; from dotenv import load_dotenv; import os;
load_dotenv(); conn = pyodbc.connect(f'DRIVER={{ODBC Driver 17 for SQL
Server}};SERVER={os.getenv("DB_DATA_SOURCE")};DATABASE={os.getenv("DB_INITIAL_C
ATALOG")};UID={os.getenv("DB_USER_ID")};PWD={os.getenv("DB_PASSWORD")}');
print('Connected successfully')"

# Test ETC handler
cd /home/Projects/TMSA
python3 etc_handler_api.py

# Test BerthPlan handler
python3 bp_handler_api.py

# Check OAuth2 token expiration
cd /home/Projects/TMSA
python3 -c "from bp_handler_api import BerthPlanHandler; from datetime import
datetime; handler = BerthPlanHandler(); print(handler.TOKEN_MSG)"

# Verify XML archives
ls -lh /home/Projects/TMSA/SOAP_Archive/ | tail -10

# Check disk space
df -h /home/Projects/TMSA

# Monitor real-time execution
tail -f /home/Projects/TMSA/cron_output.log
```

Log File Locations

Log Type	Location
Cron Output	/home/Projects/TMSA/cron_output.log
ETC XML Archives	/home/Projects/TMSA/SOAP_Archive/
Application Logs	/home/Projects/TMSA/log/

Support Contacts

For additional support, contact the following teams:

Application Team: APMT IT Development Team (BIH001)

Database Team: N4 Database Administrators

Network Team: Terminal Network Operations

TMSA Support: TMSA Platform Support Team (TC)

Appendix: Quick Reference

Manual Execution

```
# Navigate to project directory
cd /home/Projects/TMSA

# Run complete flow
python3 main.py

# Run only ETC
python3 -c "import asyncio; from etc_handler_api import EtcHandler; handler =
EtcHandler(); asyncio.run(handler.get_etc_xml())"

# Run only BerthPlan
python3 -c "import asyncio; from bp_handler_api import BerthPlanHandler;
handler = BerthPlanHandler(); asyncio.run(handler.metrics_handler())"

# Test metric calculator
python3 -c "from metrics import BerthMetricCalculator; calc =
BerthMetricCalculator(); print(calc.get_metrics('B45.5', 250.0, True))"

# View generated XML
cat xml_file.xml | head -50
```

Environment Variables Reference

```
# Complete .env file template
DB_DATA_SOURCE=10.212.12.68
DB_PORT=1433
DB_INITIAL_CATALOG=Sparcsn4
DB_USER_ID=db_user
DB_PASSWORD=secure_password

ETC_URI=https://tmsa-endpoint/etc
ETC_AUTH_USER=etc_user
ETC_AUTH_PASSWORD=etc_password

CLIENT_ID=client_id_here
CLIENT_SECRET=client_secret_here
SCOPE=required_scope
CONSUMER_KEY=consumer_key_here
TOKEN_URL=https://auth-endpoint/token
BP_TOKEN_EXP_DATA=2027-12-31

BP_URL=https://api-endpoint/berthplan
BP_XML_SEND_URL=https://tmsa-endpoint/berthplan
BP_XML_USER=bp_user
BP_XML_PASSWORD=bp_password
```