Next Gen IDS – Client Deployment Guide

*Version 1.0  |  August 1, 2025*

\*\*Table of Contents (update in Word → References → Table of Contents)\*\*

# 1  Overview

This guide explains how to install and run the AI‑powered “Next Gen IDS” package on \*\*client machines\*\* running Windows 10/11 or modern Linux distributions (Ubuntu 22.04+, Debian 12, Fedora 39, etc.). It covers native (Python) deployment and a fully containerised \*\*Docker Compose\*\* option, plus PostgreSQL setup, database schema creation, and service start‑up.

# 2  Prerequisites

## 2.1  Hardware

* x86‑64 CPU, 4 cores minimum (8+ recommended for busy links)
* 8 GB RAM minimum (16 GB recommended if TensorFlow CPU is used)
* Administrative privileges for packet‑capture (WinPCap/Npcap/tshark)

## 2.2  Software

* Python 3.10 – 3.11 (if using native install)
* Git (optional; or download the release ZIP)
* PostgreSQL 14 +
* Wireshark / tshark 4.x (PyShark requirement)
* Docker 20.10+ and Docker Compose v2 (optional, for container install)

# 3  Quick‑Start Options

Choose \*\*one\*\* of the following approaches:

## 3.1  Option A — Docker Compose (Cross‑Platform, Recommended)

* Install Docker and Docker Compose.
* Clone or extract the `ids` project folder.
* From the project root, run `docker compose up -d --build`.
* The stack launches two containers:
* `ids` – packet sniffer + Flask dashboard
* `db` – PostgreSQL 16 with pre‑initialised database
* Edit `docker-compose.yml` if you need to change interface name or environment variables.

## 3.2  Option B — Native Python Install

* Ensure PostgreSQL is installed and running (see § 4).
* Install Wireshark (Windows) or `sudo apt install tshark` (Linux).
* Clone/extract project → `cd ids`
* Create venv: `python -m venv venv` → activate (`venv\Scripts\activate` on Windows, `source venv/bin/activate` on Linux).
* Install dependencies: `pip install -r requirements.txt`
* Set environment variables (see § 5) or copy `.env.example`.
* Run IDS: `python ids/scripts/run\_ids.py --interface <iface>`

# 4  Installing & Configuring PostgreSQL

## 4.1  Windows

* Download \*\*EnterpriseDB PostgreSQL Installer\*\* (https://www.enterprisedb.com/downloads).
* Run installer → select components \*\*PostgreSQL Server\*\* and \*\*Command Line Tools\*\*.
* Set super‑user password and port (default 5432).
* Open \*\*pgAdmin 4\*\* → create role `ids` with password `ids` and privileges: LOGIN, CREATEDB.
* Create database `net\_analysis` owned by `ids`.

## 4.2  Linux (Ubuntu/Debian example)

* `sudo apt update && sudo apt install postgresql postgresql-contrib`
* Switch to postgres user: `sudo -iu postgres`
* `createuser --interactive` → name: \*\*ids\*\*, make role superuser? \*\*n\*\*, allow create DB? \*\*y\*\*
* `createdb -O ids net\_analysis`
* Exit back to your user account.

## 4.3  Create Tables

Run the schema script supplied with the project:

* Windows: `psql -U ids -d net\_analysis -f ids\queries\create\_tables.sql`
* Linux  : `psql -U ids -d net\_analysis -f ids/queries/create\_tables.sql`

# 5  Environment Variables

Create file \*\*.env\*\* in project root or set variables in your shell/service definition:

|  |  |
| --- | --- |
| Variable | Example Value |
| IDS\_DB\_DSN | postgres://ids:ids@localhost:5432/net\_analysis |
| IDS\_INTERFACE | Wi-Fi (Windows) or eth0 / wlan0 (Linux) |
| SMTP\_SERVER | smtp.gmail.com |
| SMTP\_USERNAME | <sender@gmail.com> |
| SMTP\_PASSWORD | <app‑password> |
| ALERT\_RECIPIENT | <your‑email>@example.com |

# 6  Running the IDS

* Activate virtualenv (`venv\Scripts\activate` | `source venv/bin/activate`).
* `python ids/scripts/run\_ids.py --interface <iface>`
* Stop with \*\*Ctrl‑C\*\*. Logs are printed to stdout.

# 7  Autostart / Daemon Mode

## 7.1  Linux (systemd unit)

Create \*\*/etc/systemd/system/ids.service\*\*:

[Unit]  
Description=Next Gen IDS  
After=network.target postgresql.service  
  
[Service]  
User=ids  
WorkingDirectory=/opt/ids  
EnvironmentFile=/opt/ids/.env  
ExecStart=/opt/ids/venv/bin/python ids/scripts/run\_ids.py --interface eth0  
Restart=on-failure  
  
[Install]  
WantedBy=multi-user.target

* `sudo systemctl daemon-reload && sudo systemctl enable --now ids`

## 7.2  Windows (Task Scheduler)

* Open \*\*Task Scheduler → Create Task\*\*.
* Run "When the computer starts", highest privileges.
* Action: `Program/script:` \*<path\to\python.exe>\*; Arguments:` ids\scripts\run\_ids.py --interface "Wi-Fi"`
* Ensure \*\*Start in\*\* points to project directory and environment variables are defined system‑wide.

# 8  Testing & Sample Data

* Use scripts under `ids/scripts` (e.g. `generate\_dummy\_data.py`) to back‑fill packets and alerts for dashboard testing.
* Run unit tests with `pytest`.

# 9  Troubleshooting

* \*\*psycopg2 OperationalError:\*\* verify DB credentials and that `pg\_hba.conf` allows local connections.
* \*\*PermissionError capturing packets:\*\* run shell as Administrator (Windows) or with `sudo` / setcap on tshark (Linux).
* \*\*TShark not found:\*\* ensure Wireshark is installed and `tshark` is in PATH.

# 10  Appendix – Database Schema

The key tables are created by `queries/create\_tables.sql`. A shortened excerpt is shown below for reference:

CREATE TABLE packets (  
 packet\_id SERIAL PRIMARY KEY,  
 ts TIMESTAMP,  
 src\_ip INET,  
 ...  
);  
  
CREATE TABLE alerts (  
 alert\_id SERIAL PRIMARY KEY,  
 ts TIMESTAMP,  
 alert\_type VARCHAR(50),  
 ...  
);  
  
CREATE TABLE host\_stats (  
 stats\_id SERIAL PRIMARY KEY,  
 interval\_start TIMESTAMP,  
 interval\_end TIMESTAMP,  
 host\_ip INET,  
 ...  
);

Here’s the high-level directory layout for the **Next Gen IDS** package

ids/ ← project root

├── DEPLOYMENT.md ← markdown copy of the install guide

├── Dockerfile ← container build recipe

├── docker-compose.yml ← multi-service stack (IDS + PostgreSQL)

├── README.md ← quick-start & architecture notes

├── requirements.txt ← pip dependencies (runtime)

├── dev-requirements.txt ← linting / test tools (dev only)

├── core/ ← packet-capture & alerting engine

│ ├── \_\_init\_\_.py

│ ├── alert.py ← Alert dataclass

│ ├── capture.py ← TShark-based sniffer wrapper

│ ├── config.py ← env / CLI config helpers

│ ├── db.py ← PostgreSQL access layer

│ └── detector.py ← base rule/ML detector interface

├── ml/ ← machine-learning model wrappers

│ ├── \_\_init\_\_.py

│ ├── base.py

│ ├── autoencoder.py

│ ├── isolation\_forest.py

│ ├── one\_class\_svm.py

│ └── factory.py ← picks detector based on config

├── ml\_models/ ← pre-trained model binaries

│ ├── autoencoder.h5

│ ├── autoencoder.keras

│ ├── autoencoder\_scaler.pkl

│ ├── isolation\_forest.pkl

│ └── one\_class\_svm.pkl

├── data/ ← demo CSVs for seeding the DB / tests

│ ├── normal\_traffic\_baseline.csv

│ ├── synthetic\_alerts.csv

│ └── synthetic\_host\_stats.csv

├── queries/ ← hand-written SQL for reports & schema

│ ├── create\_tables.sql

│ ├── alerts\_by\_hour.sql

│ └── ddos\_last\_10m.sql

├── scripts/ ← one-off utilities & entry points

│ ├── run\_ids.py ← \*\*main CLI\*\* - starts sniffer + engine

│ ├── generate\_dummy\_data.py ← back-fill packets/alerts for the UI

│ ├── train\_models.py ← retrains ML detectors (offline)

│ ├── load\_csv\_data\_to\_database.py

│ ├── icmp\_flood\_win.py ← Windows test attack script

│ ├── syn\_flood.py ← Linux/Unix SYN flood generator

│ └── … (other small helpers)

├── web/ ← Flask dashboard

│ ├── \_\_init\_\_.py

│ ├── app.py ← Blueprint + route registration

│ ├── extensions.py ← Flask-SQLAlchemy, CORS, etc.

│ ├── templates/ ← Jinja2 views

│ │ ├── index.html

│ │ ├── dashboard.html

│ │ ├── train\_model.html

│ │ ├── login.html

│ │ └── admin.html

│ └── static/ ← CSS, JS, logos, saved plots

│ ├── css/

│ ├── img/

│ └── training\_results/ ← latest 3-D model-training PNGs

├── tests/ ← pytest unit tests

│ ├── \_\_init\_\_.py

│ └── test\_detector.py

└── log/ (optional) ← runtime logs if LOG\_DIR env set

**Key take-aways**

* **core/** – networking & rule/ML detection engine; minimal external deps.
* **ml/ & ml\_models/** – plug-and-play anomaly detectors (Isolation Forest, Autoencoder, One-Class SVM).
* **web/** – lightweight Flask UI for real-time stats, model training, and admin.
* **scripts/run\_ids.py** – single entry-point for CLI or service wrapper (systemd / Task Scheduler).
* **Dockerfile + docker-compose.yml** – spin up the IDS and PostgreSQL in seconds.