OwnTracks Setup

1. Install the OwnTracks Server and a Metrics Export Script

Set up the OwnTracks Server and a metrics export script using Docker. The script will push location metrics from the OwnTracks Server to InfluxDB.

/var/lib/docker/volumes
├── owntracks-server_store/
├── downtracks-server_scripts/
│ └── _data/
│ └── owntracks_to_influx.ps1
│ └── owntracks_last.json
├── owntracks-server_config/

docker-compose.yml

```
services:
  otrecorder:
    container_name: owntracks-server
    image: owntracks/recorder
    networks:
      - network2
    ports:
      - 8083:8083
    volumes:
      - config:/config
      - store:/store
    restart: unless-stopped
    environment:
      OTR_PORT: 0

  owntracks-tracker:
```

```
    image: mcr.microsoft.com/powershell:latest
    container_name: owntracks-exporter-ps1
    restart: always
    networks:
      - network2
    volumes:
      - scripts:/scripts
    entrypoint: pwsh scripts/owntracks_to_influx.ps1



volumes:
  scripts: {}
  store: {}
  config: {}

networks:
  network2:
    name: services
    external: true
```

owntracks_to_influx.ps1

```
# =========================
# Configuration
# =========================
# InfluxDB connection settings
$InfluxHost = "InfluxDBv2 IP"
$Bucket = ""
$Org = ""
$Token = "API Key"

# OwnTracks API endpoint and credentials
$OwnTracksURL = "http://IP:8083/api/0/last"
$User = ""
$Device = ""

# File to store last known locations (we only keep 2 entries: current + previous)
$LastFile = "scripts/owntracks_last.json"

# Interval between location updates (seconds)
```

```
$SleepSeconds = 60

#Apply distance filter (meters)
$DistanceMetrs = 0.05

# ==========================
# Helper Functions
# ==========================

# Convert degrees to radians (needed for Haversine formula)
function Deg2Rad($deg) {
    return $deg * [math]::PI / 180
}

# Haversine formula – calculates great-circle distance between two lat/lon points
# Returns distance in kilometers
function Haversine($lat1, $lon1, $lat2, $lon2) {
    $R = 6371  # Earth radius in km
    $dLat = Deg2Rad($lat2 - $lat1)
    $dLon = Deg2Rad($lon2 - $lon1)
    $lat1Rad = Deg2Rad($lat1)
    $lat2Rad = Deg2Rad($lat2)

    # Formula for great-circle distance
    $a = [math]::Sin($dLat/2) * [math]::Sin($dLat/2) +
         [math]::Cos($lat1Rad) * [math]::Cos($lat2Rad) *
         [math]::Sin($dLon/2) * [math]::Sin($dLon/2)
    $c = 2 * [math]::Atan2([math]::Sqrt($a), [math]::Sqrt(1-$a))
    return $R * $c
}

# ==========================
# Main Loop
# ==========================
while ($true) {
    try {
        # ------------------------------------------------
        # 1. Fetch the latest location from OwnTracks API
        # ------------------------------------------------
        $response = Invoke-RestMethod -Method POST -Uri $OwnTracksURL -Body @{user=$User;
```

```
device=$Device}
        $data = $response[0]  # Take the first (latest) record

        # ------------------------------------------------
        # 2. Convert data to correct numeric types
        # ------------------------------------------------
        $lat = [double]$data.lat
        $lon = [double]$data.lon
        $acc = [double]$data.acc
        $vel = [double]$data.vel
        $alt = [double]$data.alt
        $batt = [double]$data.batt
        $tst = [int][double]::Parse($data.tst)  # Unix timestamp

        # ------------------------------------------------
        # 3. Load previous location from file (if available)
        # ------------------------------------------------
        $lastLocations = @()
        if (Test-Path $LastFile) {
            $loaded = Get-Content $LastFile | ConvertFrom-Json
            if ($loaded -is [System.Collections.IEnumerable]) {
                $lastLocations = $loaded
            } else {
                $lastLocations = @($loaded) # ensure array format
            }
        }

        # Default distance value (if no previous location exists)
        $distance_km = 0

        # ------------------------------------------------
        # 4. If we have a previous location, calculate the distance
        # ------------------------------------------------
        if ($lastLocations.Count -ge 1) {
            $prev = $lastLocations[-1]  # last stored entry
            $distance_km = Haversine ([double]$prev.lat) ([double]$prev.lon) $lat $lon
        }

        # ------------------------------------------------
        # 4a. Apply distance filter (meters)
```

```
        # ------------------------------------------------
        if ($distance_km -lt $DistanceMetrs) {
            $distance_km = 0
        }

        # ------------------------------------------------
        # 5. Update stored history (keep only the last 2 locations)
        # ------------------------------------------------
        $lastLocations += ,$data  # add current location
        if ($lastLocations.Count -gt 2) {
            $lastLocations = $lastLocations[-2..-1]  # trim to last 2 entries
        }
        $lastLocations | ConvertTo-Json | Set-Content $LastFile

        # ------------------------------------------------
        # 6. Format values for InfluxDB (8 decimals for lat/lon, 10 for distance)
        # ------------------------------------------------
        $latStr = $lat.ToString("F8", [System.Globalization.CultureInfo]::InvariantCulture)
        $lonStr = $lon.ToString("F8", [System.Globalization.CultureInfo]::InvariantCulture)
        $distanceStr = $distance_km.ToString("F10",
[System.Globalization.CultureInfo]::InvariantCulture)

        # ------------------------------------------------
        # 7. Build InfluxDB line protocol string
        # ------------------------------------------------
        $epochNow = [int][Math]::Round((Get-Date).ToUniversalTime().Subtract([datetime]"1970-
01-01").TotalSeconds)
        $line = "location,user=$User,device=$Device
lat=$latStr,lon=$lonStr,acc=$acc,alt=$alt,vel=$vel,batt=$batt,distance_km=$distanceStr
$epochNow"

        # ------------------------------------------------
        # 8. Send data to InfluxDB
        # ------------------------------------------------
        $uri = "$InfluxHost/api/v2/write?bucket=$Bucket&org=$Org&precision=s"
        Invoke-RestMethod -Method POST -Uri $uri -Headers @{Authorization = "Token $Token"} -
Body $line

        # ------------------------------------------------
        # 9. Print status in console
```

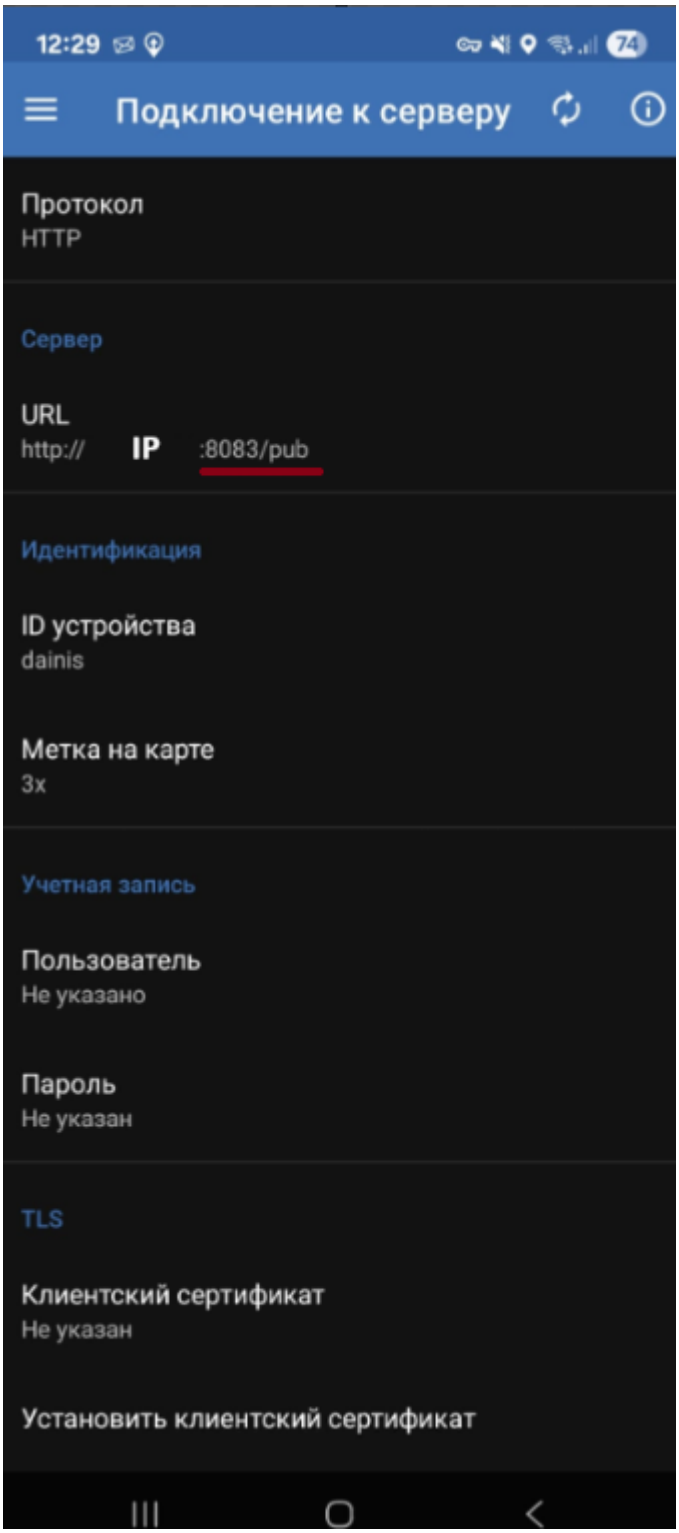
```
        # ------------------------------------------------
        Write-Host "Saved location $latStr,$lonStr (Distance: $distanceStr km) at $tst"
    }
    catch {
        # Handle errors gracefully (e.g., network/API issues)
        Write-Warning "Error: $($_.Exception.Message)"
    }

    # ------------------------------------------------
    # 10. Wait before fetching the next location
    # ------------------------------------------------
    Start-Sleep -Seconds $SleepSeconds
}
```

delete_all_records_from_DB.ps1

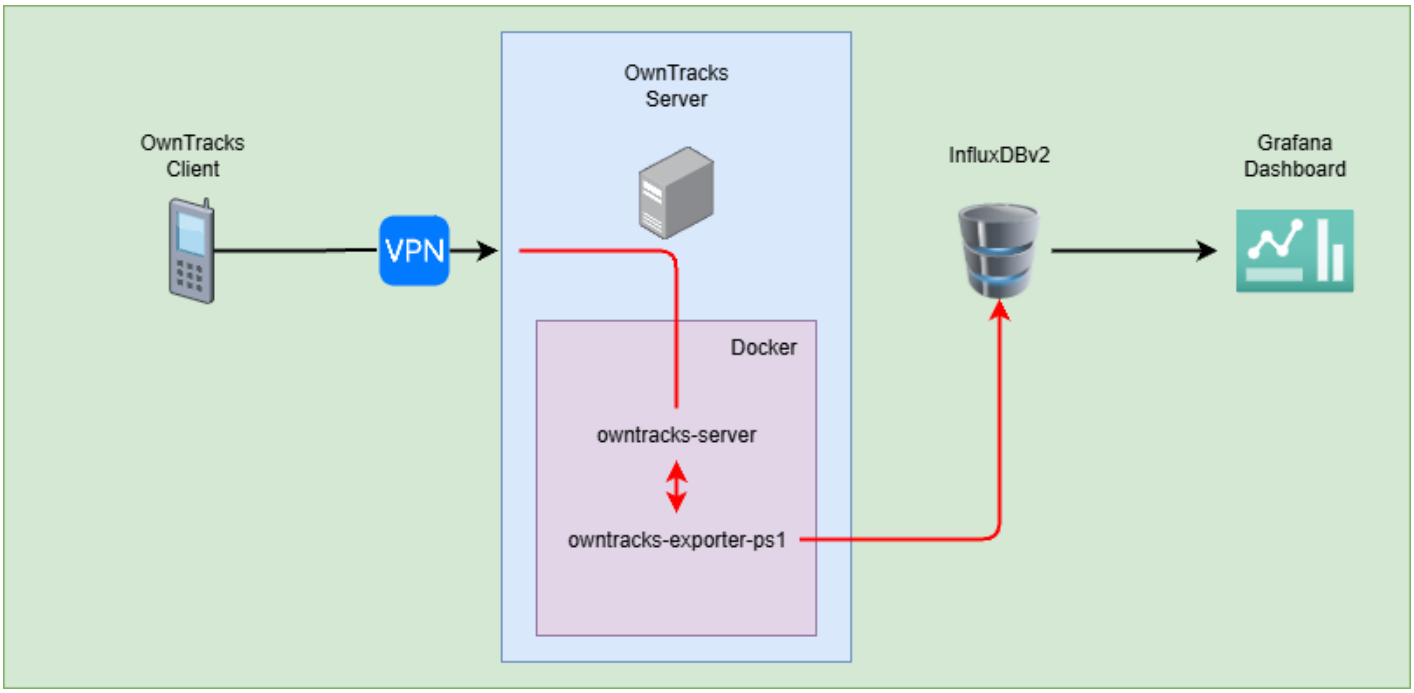
```
$InfluxHost = "InfluxDBv2 IP"
$Bucket     = ""
$Org        = ""
$Token      = "API"

$stop = (Get-Date).ToUniversalTime().ToString("yyyy-MM-ddTHH:mm:ssZ")
$json = "{""start"":""1970-01-01T00:00:00Z"",""stop"":""$stop""}"
$bytes = [System.Text.Encoding]::UTF8.GetBytes($json)

Invoke-WebRequest -Method Post `
  -Uri "$InfluxHost/api/v2/delete?org=$Org&bucket=$Bucket" `
  -Headers @{ Authorization = "Token $Token" } `
  -ContentType 'application/json; charset=utf-8' `
  -Body $bytes `
  -UseBasicParsing
```

2. Install OwnTracks on Your Phone

- **Protocol:** Select **HTTP**
- **Server URL:** Enter the URL of the Docker container running the OwnTracks server.


12:29
74
Подключение к серверу
Протокол
HTTP
Сервер
URL
http://
IP
:8083/pub
Идентификация
ID устройства
dainis
Метка на карте
3x
Учетная запись
Пользователь
Не указано
Пароль
Не указан
TLS
Клиентский сертификат
Не указан
Установить клиентский сертификат

3. Layout



Revision #10
Created 3 October 2025 10:24:58 by Dainis
Updated 4 October 2025 14:06:18 by Dainis