Using Termux to sending your GPS

Step	Detail
Install Termux + Python	https://f-droid.org/en/packages/com.termux/
Install termux-api + setup location	https://f-droid.org/packages/com.termux.api/
Write Python MQTT script	Below code
Generate SAS token	
Send GPS to Azure IoT Hub	Replace your real (i) iot_hub_name, (ii) device_id and (iii) device_primary_key. Read more: https://learn.microsoft.com/en-us/azure/iot/iot-mqtt-connect-to-iot-hub#tls-configuration

---

👩🏻‍💻Install Termux + Python

Termux is a terminal emulator + Linux environment for Android.

Install Termux:

- Download Termux **from F-Droid** (recommended, not from Play Store): https://f-droid.org/en/packages/com.termux/

After installation, open Termux and run:

```
# Update and upgrade packages
pkg update
pkg upgrade

# Install Python and some tools
pkg install python termux-api git

# Install MQTT Python client
pip install paho-mqtt
```

👨‍💻Install termux-api + setup location

Install Termux:API app (Android part):

- Install **Termux:API** APK from F-Droid link here:
  https://f-droid.org/packages/com.termux.api/
  (⚡ Not available on Play Store.)
- **Give permission**: In Termux, run:

```
termux-setup-storage
```

Now test manually:

- If it asks permission for "location", **click Allow**.

```
termux-location -p network
>> {
  "latitude": 10.8236391,
  "longitude": 106.6246496,
  "altitude": 24.30000114440918,
  "accuracy": 17.875999450683594,
  "vertical_accuracy": 9.563785552978516,
  "bearing": 0.0,
  "speed": 0.0,
  "elapsedMs": 174,
  "provider": "network"
}

termux-location -p gps
>> {
  "latitude": 10.822995438732521,
  "longitude": 106.62443690385572,
  "altitude": -32.42641115876083,
  "accuracy": 96.0,
  "vertical_accuracy": 159.95823669433594,
  "bearing": 160.51629638671875,
  "speed": 0.17218974232673645,
  "elapsedMs": 105,
```

```
  "provider": "gps"
}
```

- If it appears as above, which means you have succeeded.

📡 "Network" provider in Termux (read more)

- "network" **uses mobile cell towers + Wi-Fi** to *guess* your location.
- If you are on **4G** or **5G** (no Wi-Fi), it uses **only cell tower triangulation**.

🇻🇳 In Vietnam (2024-2025 situation):

- **4G/5G** coverage is **good** in cities (HCM, Hanoi, Da Nang, etc.), *but*...
- **Cell tower triangulation** location is **coarse**:
  - In cities: error ~ **100 meters to 500 meters**.
  - In rural areas: error ~ **500 meters to 2 km**.
- **Wi-Fi** (if available) improves "network" accuracy a lot (~10-50 meters).

✅ Summary:

Situation	Expected Accuracy
4G/5G in City (no Wi-Fi)	~100m to 500m
4G/5G in Countryside (no Wi-Fi)	~500m to 2000m
4G/5G + Wi-Fi nearby	~10m to 50m

🔥 Therefore:

- **If you need accurate tracking (like < 10 meters) ➔ Use `gps`.**
- **If you just want general area (like 100-500 meters) ➔ `network` is OK.**
- **If indoors ➔** GPS can fail; network can still guess.
- **If moving fast (car, bike) ➔** GPS is strongly recommended.

📈 Real-world example in HCM City:

- On **4G** only: my phone with "network" gave me ~200 meters error.
- On **Wi-Fi** + **4G**: "network" gave me ~20 meters accurate location.
- With **GPS**: <5 meters accuracy.

🚀Our project:

- Let the user initially choose between `network/gps` provider.

💻Write Python MQTT script & Connect with Azure IoT Hub

If you currently don't have an IoT Hub and Device, please follow the guidance below:
—-
(i) Create and manage Azure IoT hubs:
https://learn.microsoft.com/en-us/azure/iot-hub/create-hub?tabs=portal
(ii) Create and manage device identities:
https://learn.microsoft.com/en-us/azure/iot-hub/create-connect-device?tabs=portal
(iii) Also, read more about using MQTT protocol directly, as well as connecting over TLS 1.2:

a. https://learn.microsoft.com/en-us/azure/iot/iot-mqtt-connect-to-iot-hub#use-the-mqtt-protocol-directly-from-a-device
b. https://learn.microsoft.com/en-us/azure/iot/iot-mqtt-connect-to-iot-hub#tls-configuration

—-

Copied this (1) advanced python script to your code (GPT generated), replace with your real (i) `iot_hub_name`, (ii) `device_id` and (iii) `device_primary_key`:

```
import subprocess
import json
import paho.mqtt.client as mqtt
import ssl
import time
import hmac
import hashlib
import base64
import urllib.parse
import warnings

# ===================== Configuration =====================
iot_hub_name = "YOUR-IOT-HUB-NAME"
device_id = "YOUR-IOT-HUB-DEVICE"
device_primary_key = "YOUR-DEVICE-PRIMARY-KEY"  # 🔐 Base64 key

mqtt_host = f"{iot_hub_name}.azure-devices.net"
mqtt_port = 8883

client_id = device_id
username =
f"{iot_hub_name}.azure-devices.net/{device_id}/?api-version=2021-04-12"
resource_uri = f"{iot_hub_name}.azure-devices.net/devices/{device_id}"

# ===================== SAS Token Generator =====================
def generate_sas_token(uri, key, expiry_in_secs=3600):
    expiry = int(time.time()) + expiry_in_secs
    sign_key = f"{uri}\n{expiry}".encode("utf-8")
    key_bytes = base64.b64decode(key)
    signature = base64.b64encode(
        hmac.new(key_bytes, sign_key, hashlib.sha256).digest()
    )
    sig_encoded = urllib.parse.quote(signature, safe="")
    return f"SharedAccessSignature sr={uri}&sig={sig_encoded}&se={expiry}",
expiry
```

```
# Initial token
sas_token, token_expiry = generate_sas_token(resource_uri, device_primary_key)

# ===================== Choose Initial Provider =====================
initial_provider = input("📡 Choose location provider to use (network/gps): ").strip().lower()
if initial_provider not in ["network", "gps"]:
    print("❌ Invalid choice. Defaulting to 'network'.")
    initial_provider = "network"
fallback_provider = "gps" if initial_provider == "network" else "network"

# ===================== Location Fetcher =====================
def get_location(provider):
    try:
        output = subprocess.check_output(["termux-location", "-p", provider])
        return json.loads(output)
    except Exception as e:
        print(f"❌ {provider} failed: {e}")
        return None

# ===================== MQTT Setup =====================
warnings.filterwarnings("ignore", category=DeprecationWarning)

def on_connect(client, userdata, flags, rc):
    print(f"✅ Connected with result code {rc}")

client = mqtt.Client(client_id=client_id, protocol=mqtt.MQTTv311)
client.username_pw_set(username=username, password=sas_token)
client.tls_set(cert_reqs=ssl.CERT_REQUIRED, tls_version=ssl.PROTOCOL_TLSv1_2)
client.on_connect = on_connect
client.connect(mqtt_host, mqtt_port, 60)
client.loop_start()

# ===================== SAS Refresh Logic =====================
def refresh_sas_token_if_needed():
    global sas_token, token_expiry
    if time.time() > token_expiry - 60:
        print("🔁 SAS token expired or about to expire. Regenerating...")
        sas_token, token_expiry = generate_sas_token(resource_uri,
```

```
device_primary_key)
        client.username_pw_set(username=username, password=sas_token)

# ==================== Send GPS Logic ====================
def send_gps():
    refresh_sas_token_if_needed()

    location = get_location(initial_provider)
    if not location:
        print(f"⚠️ {initial_provider} failed, trying {fallback_provider}...")
        location = get_location(fallback_provider)
        if not location:
            print("❌ Both providers failed. Skipping...")
            return

    speed_mps = location.get("speed", 0)
    speed_kph = speed_mps * 3.6
    accuracy = location.get("accuracy", None)

    print(f"📡 Provider: {initial_provider}")
    print(f"🚦 Speed: {speed_kph:.1f} km/h")
    print(f"🎯 Accuracy: {accuracy} meters" if accuracy is not None else "🎯
Accuracy: N/A")

    payload = json.dumps({
        "latitude": location["latitude"],
        "longitude": location["longitude"],
        "altitude": location.get("altitude", 0),
        "timestamp": int(time.time())
    })

    topic = f"devices/{device_id}/messages/events/"
    print("📤 Sending GPS:", payload)
    client.publish(topic, payload, qos=1)

# ==================== Main Loop ====================
try:
    while True:
        send_gps()
```

```
        time.sleep(3)
except KeyboardInterrupt:
    print("🚫 Program interrupted. Exiting...")
    client.loop_stop()
    client.disconnect()
```

Or this (2) basic python script, which is easier to understand (no error handling and provider choosing logic):

```
import subprocess
import json
import paho.mqtt.client as mqtt
import ssl
import time
import hmac
import hashlib
import base64
import urllib.parse

# ===================== Configuration =====================
iot_hub_name = "YOUR-IOT-HUB-NAME"
device_id = "YOUR-DEVICE-ID"
device_primary_key = "REPLACE_WITH_YOUR_DEVICE_PRIMARY_KEY"  # 🔐 Base64 key

mqtt_host = f"{iot_hub_name}.azure-devices.net"
mqtt_port = 8883

client_id = device_id
username =
f"{iot_hub_name}.azure-devices.net/{device_id}/?api-version=2021-04-12"
resource_uri = f"{iot_hub_name}.azure-devices.net/devices/{device_id}"

# ===================== SAS Token Generator =====================
def generate_sas_token(uri, key, expiry_in_secs=3600):
    expiry = int(time.time()) + expiry_in_secs
    sign_key = f"{uri}\n{expiry}".encode("utf-8")
    key_bytes = base64.b64decode(key)
    signature = base64.b64encode(
```

```
        hmac.new(key_bytes, sign_key, hashlib.sha256).digest()
    )
    sig_encoded = urllib.parse.quote(signature, safe="")

    return f"SharedAccessSignature sr={uri}&sig={sig_encoded}&se={expiry}",
expiry

# Initial token
sas_token, token_expiry = generate_sas_token(resource_uri, device_primary_key)

# ===================== GPS Fetcher =====================
def get_gps_location():
    providers = ["network", "gps"]  # Try network first, then GPS
    for provider in providers:
        try:
            print(f"Trying provider: {provider}")
            output = subprocess.check_output(["termux-location", "-p", provider])
            data = json.loads(output)
            return {
                "latitude": data["latitude"],
                "longitude": data["longitude"],
                "altitude": data.get("altitude", 0),
                "timestamp": int(time.time())
            }
        except Exception as e:
            print(f"Provider {provider} failed: {e}")
    print("No location providers worked!")
    return None

# ===================== MQTT Setup =====================
def on_connect(client, userdata, flags, rc):
    print(f"Connected with result code {rc}")

client = mqtt.Client(client_id=client_id, protocol=mqtt.MQTTv311)
client.username_pw_set(username=username, password=sas_token)
client.tls_set(
    ca_certs=None,
    certfile=None,
    keyfile=None,
```

```
    cert_reqs=ssl.CERT_REQUIRED,
    tls_version=ssl.PROTOCOL_TLSv1_2
)
client.on_connect = on_connect
client.connect(mqtt_host, mqtt_port, 60)
client.loop_start()

# ===================== SAS Refresh Logic =====================
def refresh_sas_token_if_needed():
    global sas_token, token_expiry
    if time.time() > token_expiry - 60:  # Refresh 1 minute before expiry
        print("🔁 SAS token expired or about to expire. Regenerating...")
        sas_token, token_expiry = generate_sas_token(resource_uri,
device_primary_key)
        client.username_pw_set(username=username, password=sas_token)

# ===================== Send GPS =====================
def send_gps():
    refresh_sas_token_if_needed()
    location = get_gps_location()
    if location:
        payload = json.dumps(location)
        topic = f"devices/{device_id}/messages/events/"
        print("Sending GPS:", payload)
        client.publish(topic, payload, qos=1)

# ===================== Main Loop =====================
try:
    while True:
        send_gps()
        time.sleep(3)
except KeyboardInterrupt:
    print("Program interrupted. Exiting...")
    client.loop_stop()
    client.disconnect()
```

📼Video demo:

Sending your GPS to Azure IoT Hub - Termux + Python