Prometheus, Grafana, Python for a later stage

Yes — **Prometheus becomes very useful** as your project grows beyond Windows logs. Here's how it can support advanced TTP detection later:

**🔍 1. Behavior-Based Monitoring (e.g., for Lateral Movement)**

You can **export system-level metrics** like CPU spikes, memory usage, number of processes, unusual I/O, or inter-host communication patterns — which are all **symptoms of lateral movement** or privilege escalation. Prometheus can collect these via:

* Node Exporter (Linux)
* WMI Exporter (Windows)
* Custom exporters (e.g., for Powershell or sysmon output)

Then use **PromQL queries** and alert rules to detect abnormal patterns.

**🧠 2. Model Monitoring for ML**

If you're using machine learning to classify behavior:

* Prometheus can track **model inference times**, **detection confidence levels**, or **number of flagged anomalies per hour**.
* This helps you catch **drift** or **reduced detection quality** in real time.

**📈 3. Detection Engineering Feedback Loops**

Prometheus can **track events triggered by your detection rules**, so you can:

* Visualize **which TTPs occur most frequently**
* Correlate TTPs across systems over time
* Identify **redundant or noisy alerts** (e.g., false positives)

**🔄 4. Security Observability**

You can integrate Prometheus with:

* **Falco** or **OpenTelemetry** for runtime security events
* **eBPF**-based observability tools to catch suspicious low-level network behavior

This enables richer **telemetry** to track attacker dwell time, propagation paths, or persistence attempts.

**🔐 5. Infrastructure Security Posture**

Track:

* System uptime
* Prometheus alert firing frequency
* Resource exhaustion (often a sign of DoS or crypto-mining attacks)

**Summary sentence for your mind map:**

In advanced TTP detection, Prometheus enables real-time monitoring of system behavior, ML model quality, and attack indicators across infrastructure, making it a valuable telemetry layer for threat detection and response.

Let me know if you'd like example PromQL queries or diagrams for any of these.

### ****2. Grafana Integration****

**Ease: ★★★★★**

* Go to **Grafana UI → Add Data Source → Prometheus**
* Set URL to http://prometheus:9090
* Done.

You can immediately start visualizing Docker and custom metrics.

### ✅ ****3. Python Integration****

Expose a /metrics endpoint:

python

CopyEdit

from prometheus\_client import start\_http\_server, Counter

import time

requests = Counter('app\_requests\_total', 'Total requests')

def process\_request():

requests.inc()

if \_\_name\_\_ == '\_\_main\_\_':

start\_http\_server(8011) # Prometheus will scrape here

while True:

process\_request()

time.sleep(1)

Prometheus can scrape this like:

yaml

CopyEdit

- job\_name: 'python-app'

static\_configs:

- targets: ['python-app:8011']

### ✅ ****4. ELK Coexistence****

**Ease: ★★★★☆**

* You **don’t need to modify ELK** — just run it in parallel.
* Keep ELK for log data, and Prometheus for structured metrics.
* Grafana bridges both, giving you **logs + metrics in one place**.

Great question!

If you're adding a **Python service or metrics exporter** to your existing ELK + Prometheus + Grafana stack, the best place to put your Python file depends on what that Python script does.

Here are a few options tailored to your project structure (as seen in your screenshot):

### 🧩 ****Option 1: For a Standalone Exporter App (e.g., exposes**** /metrics****)****

If your Python script is a microservice that exposes Prometheus metrics (e.g., via Flask or FastAPI with prometheus\_client), place it in:

📁 src/metrics\_exporter/  
(Create this folder if it doesn’t exist yet.)

**Example structure:**

src/

metrics\_exporter/

exporter.py

requirements.txt

Dockerfile

### 🧩 ****Option 2: For an Existing App You’re Instrumenting****

If you're instrumenting an **existing Python app** (e.g., a log processor, ML model, etc.), then:

* Place the script alongside the rest of that module under src/
* Add the Prometheus instrumentation code inside the app logic

Example:

src/

my\_app/

app.py

metrics.py # ← Prometheus metric definitions

### 🧩 ****Option 3: For Data Collection or Processing Scripts****

If you're using Python scripts to **parse Windows logs** or run inference for lateral movement detection, and want to add Prometheus metrics (e.g., counters for processed logs), you can:

* Place those in scripts/ if they are not services
* Or move to src/telemetry/ if you want a cleaner structure

### 🐍 Sample Exporter Code (Flask-based)

If you want to expose basic metrics on localhost:8000/metrics, here's a simple example:

# src/metrics\_exporter/exporter.py

from flask import Flask, Response

from prometheus\_client import Counter, generate\_latest

app = Flask(\_\_name\_\_)

log\_counter = Counter('log\_lines\_processed', 'Number of log lines processed')

@app.route("/process")

def process():

log\_counter.inc() # simulate processing

return "Processed"

@app.route("/metrics")

def metrics():

return Response(generate\_latest(), mimetype='text/plain')

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host="0.0.0.0", port=8000)

### 📦 Dockerfile for Exporter

# src/metrics\_exporter/Dockerfile

FROM python:3.11-slim

WORKDIR /app

COPY exporter.py .

RUN pip install flask prometheus\_client

CMD ["python", "exporter.py"]

### 📄 docker-compose.yml Addition

Add this to your existing docker-compose.yml:

metrics\_exporter:

build: ./src/metrics\_exporter

ports:

- "8000:8000"

networks: [elk]

Then update prometheus.yml:

- job\_name: 'python\_exporter'

static\_configs:

- targets: ['metrics\_exporter:8000']

Let me know if you'd like a working example dropped into your project!