

API monitoring options



I want you to act as a SOLUTION architect in a tech startup. Imagine you're working on a greenfield deployment of RESTful APIs in python.

what are the monitoring options for API deployed on on-prem VMs using FAST API library in python?

When deploying **RESTful APIs using FastAPI on on-prem VMs**, you'll need to establish **both application-level monitoring** (for API health, errors, performance) and **infrastructure-level monitoring** (for the VM, OS, network). Since this is a **greenfield project**, you have full freedom to choose modern, cloud-native tools while remaining on-prem. Let's break it down:

1. Core Monitoring Areas

1. Availability / Health

- Uptime checks (Is the API reachable? Is it returning expected responses?)
- Health endpoints (`/health` or `/ready`) built into FastAPI.

2. Performance

- Latency (95th/99th percentile)
- Throughput (requests per second)
- Slow endpoints profiling.

3. Errors

- HTTP 4xx vs 5xx distribution
- Exception tracking with stack traces.

4. Infrastructure

- CPU, RAM, disk I/O on VMs
- Network bandwidth and errors
- Process restarts and crashes.

5. Business metrics (optional but valuable)

- API call volumes by feature or client
- Custom domain-specific KPIs.

2. Monitoring Options

A. Open-Source Metrics & Observability Stack (Best for full control)

- **Prometheus + Grafana**

- Use [Prometheus FastAPI Instrumentator](#) ↗
- Exposes `/metrics` endpoint automatically with latency, request count, status codes.
- Grafana dashboards for visualization.
- Node Exporter to track VM metrics.
- Pros: No vendor lock-in, customizable, widely used.
- Cons: You must manage scaling, storage retention.

- **OpenTelemetry**

- Language-agnostic standard for tracing, metrics, logs.
- Use [opentelemetry-instrumentation-fastapi](#) ↗
- Send traces to Jaeger, Tempo, or Zipkin.
- Pros: Vendor-neutral, future-proof.
- Cons: Steeper learning curve than Prometheus alone.

B. Application Performance Monitoring (APM) Tools (Simpler to start)

- **Elastic APM (self-hosted)**

- Agent for Python to track request performance, errors.
- Kibana UI to visualize traces and logs.
- Good fit if you already plan to use Elastic Stack.

- **Sentry (self-hosted)**

- Focused on error tracking and performance monitoring.
- Lightweight setup for API exceptions + distributed tracing.

- **Other APMs (commercial but on-prem capable)**

- Dynatrace, New Relic, Datadog can run on-prem, but usually require licensing.

C. Log Aggregation

- **ELK Stack (Elasticsearch, Logstash, Kibana) or OpenSearch**

- Send structured logs from FastAPI via Python logging middleware.
- Combine with Filebeat or Fluentd for log shipping from VM.
- Useful for error investigation, audit trails.

3. FastAPI-Specific Considerations

- **Middleware for metrics**

Use `prometheus-fastapi-instrumentator` to auto-collect request durations, response sizes, and error

counts.

- **Middleware for logging**

Add a custom middleware that logs request IDs, user agents, and response codes.

- **Health checks & readiness probes**

Implement `/health` or `/ready` endpoints that check DB/cache connectivity.

- **Dependency injection for trace IDs**

Inject request IDs into logs to correlate with traces in APM tools.

4. Infrastructure Monitoring

- **Node Exporter + Grafana**

- Tracks VM metrics like CPU, RAM, disk.

- **Process supervision**

- Use `systemd`, `supervisord`, or `pm2` to ensure Uvicorn/Gunicorn stays up.

- **Alerting**

- Prometheus Alertmanager or Grafana Alerting for CPU spikes, high latency, or downtime.

5. Suggested Reference Architecture (Self-hosted and Modern)

1. **FastAPI App** → instrument with OpenTelemetry + Prometheus metrics
2. **Prometheus** → collects app + VM metrics
3. **Grafana** → dashboards + alerting
4. **Jaeger (or Tempo)** → distributed tracing
5. **ELK / OpenSearch** → centralized logs
6. **Alertmanager** → sends alerts via email, Slack, or PagerDuty

This stack is fully open-source, vendor-neutral, and runs on-prem easily.

Summary of Options

- **Lightweight / fast start:** Sentry (self-hosted) + Node Exporter + Grafana
- **Full observability stack:** OpenTelemetry + Prometheus + Grafana + Jaeger + ELK
- **Enterprise APM alternative:** Dynatrace, Elastic APM, or New Relic (on-prem mode)

Do you want me to **draw a full architecture diagram** showing how these tools would integrate with FastAPI on VMs, or **list concrete setup steps and Python code snippets** to instrument the APIs?



compare various monitoring options for FAST API in python in table format

Here's a **comparison table** of the main monitoring options for **FastAPI in Python**, tailored for on-prem VM deployments:

Monitoring Options for FastAPI (On-Prem VM)

Option	Type	What It Monitors	Pros	Cons	Best For
Prometheus + Grafana	Metrics + Dashboards	Request latency, throughput, status codes, VM resource metrics (with Node Exporter)	Open-source, widely adopted, highly customizable, integrates well with FastAPI via prometheus-fastapi-instrumentator	Manual setup, storage/retention management required	Teams comfortable managing infra; long-term metric storage
OpenTelemetry + Jaeger/Tempo	Distributed Tracing + Metrics	End-to-end request traces, spans, latency breakdown, service-to-service dependencies	Vendor-neutral, future-proof, integrates with many backends	Steeper learning curve, more components to run	Microservices or complex distributed apps
Elastic APM (self-hosted)	APM + Tracing + Logs	API performance, exceptions, DB calls, logs in one stack	Unified observability in Elastic Stack, powerful Kibana UI	Heavier resource use, Elastic licensing considerations	Teams already using ELK or wanting all-in-one stack
Sentry (self-hosted)	Error Tracking + Performance	Exceptions, slow endpoints, basic transaction tracing	Lightweight setup, developer-friendly UI, good alerting	Limited deep metrics compared to Prometheus/OTel	Fast error triage, smaller teams
New Relic / Dynatrace (on-prem capable)	Full APM	API performance, infra metrics, traces, logs	Turnkey dashboards, strong support, minimal code instrumentation	Licensing costs, partial vendor lock-in	Enterprises with budget for commercial support
ELK / OpenSearch Stack	Log Aggregation + Search	Structured API logs, error analysis, audit trails	Great for debugging and historical search, integrates with Filebeat/Fluentd	Limited native metrics, needs extra setup for latency stats	Deep log analysis rather than real-time metrics
Node Exporter (alone)	Infra Metrics	VM CPU, RAM, disk, network	Simple, fast to set up, works with Prometheus	No app-level visibility	Basic infrastructure health monitoring

Do you want me to **extend this table with example Python instrumentation code** (e.g., Prometheus middleware vs. OpenTelemetry tracing), or **add a cost/complexity rating to each option** for decision-making?



what third party open source monitoring & logging libraries are available for FAST API in python? print in table format

Here's a list of **third-party open-source monitoring and logging libraries** commonly used with **FastAPI in Python**, formatted as a table for quick reference:

Open-Source Monitoring & Logging Libraries for FastAPI

Library / Tool	Purpose	Key Features	Integration Effort	Notes
prometheus-fastapi-instrumentator	Metrics	Auto-collects request counts, latency, response codes; exposes /metrics endpoint	Low	Pairs with Prometheus + Grafana
OpenTelemetry (opentelemetry-instrumentation-fastapi)	Tracing + Metrics + Logs	Vendor-neutral telemetry, distributed tracing, span export to Jaeger/Tempo/Zipkin	Medium	Best for complex/microservice environments
Sentry Python SDK	Error tracking + Performance	Captures exceptions, slow endpoints, request context, supports self-hosted Sentry	Low	Great for real-time error triage
Elastic APM Python Agent	APM (Performance + Tracing)	Tracks request latency, DB queries, exceptions; integrates with Kibana	Medium	Best if using Elastic Stack
structlog	Logging	Structured JSON logs, easy correlation with request IDs	Low	Good for log pipelines like ELK or OpenSearch
Loguru	Logging	Developer-friendly logger with minimal config, colored logs, rotation	Low	Good drop-in replacement for logging
python-json-logger	Logging	Outputs logs in JSON format for ingestion by log aggregators	Low	Useful if sticking to stdlib logging
Starlette Exporter	Metrics	Similar to Prometheus instrumentator but at Starlette level (FastAPI's base)	Low	Alternative for simple metrics
Jaeger Client (Python)	Distributed Tracing	Sends trace spans to Jaeger for visualization	Medium	Typically used with OpenTelemetry
uvicorn-gunicorn-fastapi-logger	Logging	Preconfigured logger compatible with	Low	Simplifies deployment setup

Library / Tool	Purpose	Key Features	Integration Effort	Notes
		Uvicorn/Gunicorn & FastAPI		

Do you want me to **rank these by popularity and ecosystem support** (so you know which ones are safest to adopt), or **show how to combine a few into a single cohesive stack (metrics + tracing + logging together)?**