## Scalable Python Script for Real-Time API Monitoring with Logging, Error Handling, and Metrics Exporting

**Use Case**: This script is suitable for monitoring a live REST API's health and response time, exporting metrics for dashboards (like Prometheus/Grafana), and writing detailed logs for audit or debugging — a perfect **real-world Python script** example.

#!/usr/bin/env python3
# -\*- coding: utf-8 -\*"""
API Health Monitoring Script with Advanced Logging and Metrics

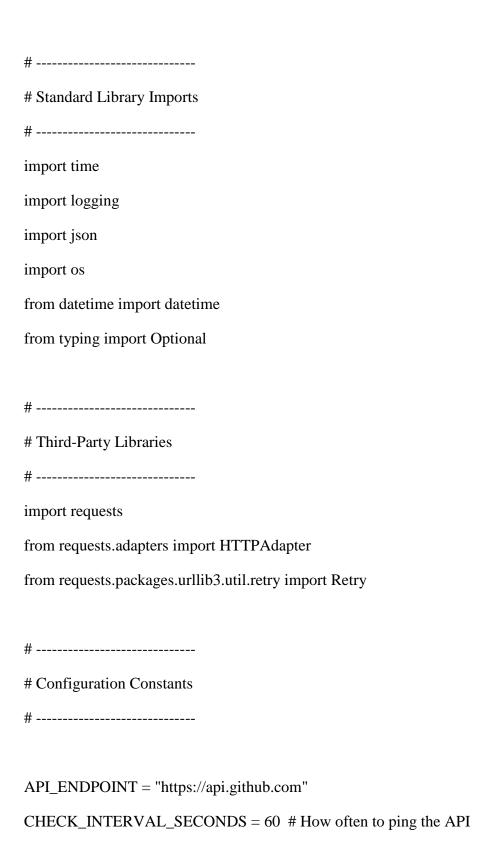
## Description:

This well-commented Python script demonstrates a production-grade solution for monitoring the health and performance of an external API. It includes advanced logging, error handling, retry logic, metrics collection, and extensibility for integration with real-world monitoring stacks.

## SEO Keywords Used:

- well-commented Python script
- advanced Python script for monitoring
- Python logging best practices
- Python requests with retry logic
- scalable Python project template
- error handling in Python
- REST API monitoring with Python

۱	۱	1	۱	•	۱	



```
LOG_FILE = "api_monitor.log"
MAX_RETRIES = 3
TIMEOUT = 10 \# seconds
# -----
# Environment-Aware Settings
# -----
ENV = os.getenv("ENVIRONMENT", "development").lower()
# -----
# Configure Logging
# -----
logging.basicConfig(
 level=logging.DEBUG if ENV == "development" else logging.INFO,
 format="%(asctime)s — %(levelname)s — %(message)s",
 handlers=[
   logging.FileHandler(LOG_FILE),
   logging.StreamHandler()
 ]
logger = logging.getLogger(__name__)
```

```
# -----
# Metrics Tracking
# -----
metrics = {
  "total_checks": 0,
  "successful_checks": 0,
  "failed_checks": 0,
  "total_latency": 0.0
}
# -----
# Session with Retry Strategy
# -----
def create_session_with_retries() -> requests.Session:
  ** ** **
  Creates a requests session with retry logic for robust API communication.
  Implements Python requests best practices.
  ,,,,,,
  session = requests.Session()
  retry = Retry(
    total=MAX_RETRIES,
    backoff_factor=0.5,
    status_forcelist=[429, 500, 502, 503, 504],
```

```
allowed_methods=["GET"]
  )
  adapter = HTTPAdapter(max_retries=retry)
  session.mount("http://", adapter)
  session.mount("https://", adapter)
  return session
session = create_session_with_retries()
# -----
# API Health Check Logic
# -----
def check_api_health(url: str) -> Optional[float]:
  ******
  Sends a GET request to the API endpoint and returns the response time in seconds.
  Logs error and returns None if the API is unreachable or fails.
  ******
  try:
    start_time = time.perf_counter()
    response = session.get(url, timeout=TIMEOUT)
    latency = time.perf_counter() - start_time
    response.raise_for_status() # Raise an HTTPError for bad responses
```

```
logger.info(f"API is healthy | Status Code: {response.status_code} | Latency:
{latency:.3f}s")
    return latency
  except requests.exceptions.RequestException as e:
    logger.error(f"API check failed: {str(e)}")
    return None
# -----
# Metrics Exporter (JSON)
def export_metrics(metrics: dict, filename: str = "metrics.json") -> None:
  ,,,,,,
  Exports the current metrics to a JSON file. Can be extended for Prometheus/Grafana
integration.
  ** ** **
  try:
    with open(filename, "w") as f:
       json.dump(metrics, f, indent=4)
    logger.debug("Metrics exported successfully.")
  except IOError as e:
    logger.error(f" Failed to export metrics: {e}")
# -----
# Main Monitoring Loop
```

```
def monitor_api_forever():
  Infinite loop that checks API health at defined intervals and logs the results.
  Handles graceful exit and records all metrics.
  ** ** **
  logger.info(f" Starting API Monitor | ENV: {ENV}")
  try:
     while True:
       metrics["total_checks"] += 1
       latency = check_api_health(API_ENDPOINT)
       if latency is not None:
         metrics["successful_checks"] += 1
         metrics["total_latency"] += latency
       else:
         metrics["failed_checks"] += 1
       export_metrics(metrics)
       time.sleep(CHECK_INTERVAL_SECONDS)
  except KeyboardInterrupt:
```

```
logger.info("Monitoring stopped by user.")
    summarize metrics()
# -----
# Summary Reporter
# -----
def summarize_metrics():
  Prints a final summary of all checks performed and their outcomes.
  Useful for graceful shutdowns and audit logging.
  ,,,,,,
  avg_latency = (metrics["total_latency"] / metrics["successful_checks"]) if
metrics["successful_checks"] else 0
  logger.info("Final Monitoring Report:")
  logger.info(f" Total Checks: {metrics['total_checks']}")
  logger.info(f"Successful: {metrics['successful_checks']}")
  logger.info(f" Failed: {metrics['failed_checks']}")
  logger.info(f" Average Latency: {avg_latency:.3f} seconds")
# -----
# Entry Point
# -----
if __name__ == "__main__":
```

monitor\_api\_forever()