Hoang Tuan Anh
Technical Interview

1. Application Study & Requirement Summary

After studying the web & application, I have made a draft version of the requirements, which contains the main functionalities of the system

- Python
    - Function Metrics Collection.
    - Asynchronous Task Management.
    - Periodic Metrics Saving.
    - Auto Metrics Saving.

2. Application Architecture Design

The essential modules that need to be done are

- Python
    - **Function Metrics Collection**: Automatically collects execution time, call count, and error count for decorated functions.
    - **Asynchronous Task Management**: Uses Celery for asynchronous task processing and periodic metrics saving.
    - **Periodic Metrics Saving**: Periodically saves metrics to the database even if the limit is not reached.
    - **Auto Metrics Saving**: Autosaves metrics to the database even if the limit is reached. After removing the record that was saved on Redis.
    - **Add calculate**: Calculate the Number of calls, Average execution time, and Number of errors on Redis Key app:metrics:{func_name}.
    - **Add record metrics**: Add new metrics entry on Redis. Key metrics_list.
    - **Get Metrics**: **Number of calls**, **Average execution time**, **Number of errors** of a Function
    - **Dockerized**: Easily deployable using Docker and Docker Compose for all components (Redis, Celery, and the application).

DB Design

I made simplified ERDs that contain only essential fields. I ignore the tables related to our internal operation system.
Assume we had some table., So I concentrated only on creating a **metrics** table.

metrics:

- id: str

- func_name: str
- execution_time: float
- error_occurred: boolean
- created_at: integer

3. High-level technology choices (Productions)

Technology Choices

Infrastructure

I recommend we use the infrastructure provided by a popular Cloud Provider (PaaS), based for the following reasons:

- Less up-front cost, we only need to pay for the resources we need, instead of spending money on big servers
- As our system grows bigger, we can easily scale the infrastructure.

Infrastructure technology choice (Assuming that we're using AWS GCP, another):

Technology	**AWS**	**Google Cloud Platform**	**Other**
Application	Elastic Beanstalk, EC2,..	Google App Engine	
Load Balancer	Elastic Load Balancing	Cloud Load Balancing	
Database	TimescaleDB, InfluxDB,...		
Search Engine	OpenSearch		Elasticsearch
Dashboard			Grafana/Kibana
Collector			Prometheus/Logstash
Message Queue	SQS	Google PubSub	Kafka
Caching	Redis Cache		
Notification	AWS SNS		

Compare some technologies: Pros and Cons

Kafka (Message Queue)
Pros: Highly scalable, distributed, fault-tolerant, handles high-throughput, enables real-time data streaming, reliable replication.
Cons: Complex to set up and manage, requires careful tuning for high performance, can be overkill for simple projects.

Redis (Cache)
Pros: Fast in-memory data store, supports various data structures, can reduce database load, simple to configure.
Cons: Limited to in-memory storage, not ideal for persistent large-scale data storage, requires memory management.

Time-Series/ClickHouse Database (InfluxDB/TimescaleDB)
Pros: Optimized for time-based data, efficient querying of historical metrics, supports high ingestion rates.
Cons: Complex queries can be slower compared to relational databases, and may require specific knowledge for tuning.

Elasticsearch (Search & Analytics)
Pros: Powerful full-text search capabilities, fast and scalable indexing, great for log and metric analysis.
Cons: Resource-intensive, requires tuning for scaling, complex query language.

Prometheus (Monitoring)
Pros: Real-time monitoring, customizable alerts, lightweight, open-source.
Cons: Not suited for long-term storage, limited support for full-text search and complex queries.

4. Microservice

Reference my system design here https://whimsical.com/metrics-monitoring-and-alerting-system-XbPrrZdDNJuWQmY1JvScPE