Principal Site Reliability Engineer - Take-Home Assessment

Overview

Build a production-ready backend service that integrates with a third-party weather API while demonstrating best practices in observability, reliability, and operational excellence.

Programming Language: Any is fine. Go or Python is strongly preferred.

FULL AI usage is strongly encouraged to shorten the time this task takes. You may use any AI tooling you wish. Code that looks like it was written by AI is perfectly acceptable as long as it performs as expected.

We likely will ask you detailed questions about specific choices you made. We may also ask you to work on it further in a live interview.

The bonus items are truly bonus items depending on what skills or experiences you want to show off and talk to later.

The Challenge

Create a **Weather Alert Service** that:

1. Accepts requests for weather information by location
2. Calls the OpenWeatherMap API (or similar free API)
3. Caches responses appropriately
4. Implements comprehensive observability
5. Handles failures gracefully

You are welcome to use any free API, it does not have to be weather focused.

Spend ~2–4 hours on this assignment. Prioritize correctness, clarity, and maintainability over extra features.

Push your solution to GitHub, GitLab, or another Git repository.

API Endpoints

Your service should expose:

GET /weather/{location}

- Returns current weather data for the specified location
- Response should include: temperature, conditions, humidity, wind speed
- If you use a non-weather API, use your best judgement
- Should cache responses

GET /health

- Returns service health status

GET /metrics

- Exposes Prometheus-compatible metrics

Technical Requirements

1. Observability & Monitoring (Critical)

Prometheus Metrics:

- Implement comprehensive observability including at least rates and latencies.
- We expect to see other areas instrumented based on your experience dealing with observability in a production environment.
- Explain each metric in the code and summarize what you're watching it for.

Logging:

- Use structured logging
- Include correlation IDs for minimal request tracing
- Never log sensitive data (API keys)
- Anything else you feel warranted

Alerting:

- Include a sample prometheus.yaml and alertmanager.yaml with your code to explain how you're using these metrics for alerting and possibly paging with a third party service like Pagerduty, Firehydrant. These are for illustrative purposes, you are not required to run a full Prometheus stack locally and test/prove them.

2. Reliability Patterns

Your app should show typical patterns for high reliability and high observability, including

- Retry logic in the right places
- Context management, propagation and timeout management
- Rate limiting
- Cache and all the patterns that come with that. You can use in-memory or Redis/Memcached and observability around it

3. Configuration Management

- Externalized configuration (environment variables or config file)
- No hardcoded credentials
- Document all configuration options

4. Other typical good practice things

- Unit tests
- README.md
- CLAUDE.md, GEMINI.md, .cursorules, etc… for the next person
- A makefile or script that lets the reviewer spin it up and test these criteria

5. Bonuses:

- SLI/SLO definitions, defined as flexible middleware, parameterized and extensible are all pluses
- Implement graceful shutdown
- Sample kubernetes manifests or helm chart
- Implement request prioritization/shed load
- Distributed tracing (OpenTelemetry or similar, depending on the language and framework you choose it may take a lot of time and energy to implement, so this is not required.)