IG

Site Reliability Engineering Test

Transport for London (TfL) API Integration Challenge

Duration: 3 hours

Difficulty: Mid to Senior SRE level

Language: Java

Overview

Build a resilient service that monitors London Underground line status and demonstrates SRE best practices including reliability patterns and SLO/SLI definition.

Scenario

You're building a critical service that provides London Underground status information to a trading platform. Traders need real-time tube status to plan their commute and trading floor presence. The service must be highly reliable as it impacts business operations.

> ⚠️ **Important:** This test focuses on SRE reliability patterns and thinking, not just functional coding. We're evaluating your approach to building production-ready, resilient systems.

Part 1: Core Functionality (30 minutes)

Using the API documentation on the Transport for London website, write code using **Java**, that can:

Requirement 1: Return Status of a Given Tube Line

Your code must be able to return the **status of a given Tube line** along with **any reason for disruption**.

Expected functionality:

- Accept a tube line identifier as input (e.g., "central", "northern")
- Query the TfL API for that specific line
- Return the current status (e.g., "Good Service", "Minor Delays", "Severe Delays")
- Include any disruption reasons if applicable

Requirement 2: Future Status with Date Range

The code should be capable of providing both the **current status** of the line and the **future status** of the line, **if a date range is supplied**.

Expected functionality:

- Accept optional date range parameters (start date and end date)
- If no date range is provided, return current status
- If date range is provided, return status information for that period
- Include any planned disruptions or engineering works within the date range

Requirement 3: All Unplanned Disruptions

The code should be capable of returning **all Tube lines** that are currently facing **unplanned disruption**.

Expected functionality:

- Query all tube lines in a single operation
- Filter to show only lines with unplanned disruptions
- Exclude planned engineering works and scheduled closures
- Return list of affected lines with their current status and disruption reasons

Sample curl Commands for Testing

```
# Get all tube line statuses
curl "https://api.tfl.gov.uk/Line/Mode/tube/Status"

# Get specific line status (Central line)
curl "https://api.tfl.gov.uk/Line/central/Status"

# Get line status with date range
curl "https://api.tfl.gov.uk/Line/northern/Status/2025-11-20/to/2025-11-22"

# Get line disruptions
curl "https://api.tfl.gov.uk/Line/central/Disruption"
```

```
# Pretty print JSON response
curl -s "https://api.tfl.gov.uk/Line/Mode/tube/Status" | python3 -m json.tool
```

Part 2: SRE Reliability Patterns (60 minutes)

This is the core of the test. Implement these reliability patterns:

1. Circuit Breaker

- Implement circuit breaker for TfL API calls
- Circuit should open after 5 consecutive failures
- Half-open after 30 seconds to test recovery
- When circuit is open, return appropriate error response

2. Retry Logic

- Implement exponential backoff with jitter
- Max 3 retries for failed TfL API calls
- Don't retry on 4xx errors (client errors)
- Retry on 5xx errors and network timeouts

> ✅ **Good Example:** Retry delays: 1s (±0.25s jitter), 2s (±0.5s), 4s (±1s)

> ❌ **Poor Example:** Retry every 1 second exactly 10 times

3. Rate Limiting / Throttling

- Implement rate limiting to protect TfL API and your service

- Limit: 100 requests per minute per client IP
- Return 429 (Too Many Requests) when limit exceeded
- Include Retry-After header in 429 responses

Part 3: SLO/SLI Definition (30 minutes)

Define Service Level Objectives for this tube status service in a separate document:

1. Identify SLIs (Service Level Indicators)

- Add metrics to measure user experience
- Identify SLIs (Service Level Indicators)

2. Define SLOs (Service Level Objectives)

- Set specific, measurable targets for each SLI
- Include time windows (e.g., 30-day rolling window)
- Example format: "99.9% of requests complete successfully over 30 days"
- Justify your targets - why these numbers?

3. Alerting Strategy

- What would you alert on and why?
- Define alert thresholds (when to page vs ticket)
- Use SLO burn rate methodology if familiar
- Explain how alerts relate to your SLOs

Provide your answers in SLO_DEFINITION.md:

- List 3-4 SLIs with justification
- Define corresponding SLOs with time windows
- Describe alerting strategy

Deliverables

1. **Source code** in a Git repository
2. **README.md** with:
    - Setup instructions
    - How to run the service
    - Architecture decisions and trade-offs
3. **SLO_DEFINITION.md** document with:
    - Proposed SLIs and justification
    - SLO targets with time windows
    - Alerting strategy
4. **Sample requests** (curl commands showing your API works)

Architecture Decisions to Document

In your README, explain your thinking on:

1. **Circuit Breaker Thresholds:** Why 5 failures? Why 30s recovery?
2. **Retry Policy:** Why exponential backoff? Why max 3 retries?

3. **Trade-offs:** What did you sacrifice for reliability?
4. **SLOs:** How did you determine your SLO targets and why?

What We're Looking For

✅ Strong Signals:

- Demonstrates understanding of circuit breaker pattern
- Implements proper retry logic (not naive retry loops)
- Comprehensive error handling (no bare try-catch)
- Well-reasoned SLO/SLI definitions with justification

❌ Red Flags:

- No error handling or retry logic
- Crashes when TfL API is down
- No timeouts (hangs forever)
- Hardcoded configuration
- No SLO/SLI understanding or arbitrary targets (99.999%)
- Ignoring rate limits (429 responses)

Submission Instructions

Please submit your work by providing:

1. Git repository link (GitHub/GitLab) or zip file
2. Ensure README includes setup and run instructions
3. Include SLO_DEFINITION.md document
4. Provide sample curl commands demonstrating the API works

> **Note:** We value working code over perfect code. Focus on demonstrating SRE principles even if some features are incomplete. Document what you would improve with more time.

Interview Discussion

After submission, be prepared to discuss:

- **Architecture:** Why did you structure it this way?
- **Failure Scenarios:** What happens when TfL API is down for 6 hours?
- **Production Readiness:** What's missing for production deployment?
- **Scaling:** How would this handle 10,000 requests/second?
- **SLO Rationale:** Why did you choose those specific SLO targets?
- **Improvements:** What would you do differently with more time?