ZUZU Take-Home Test: Review System Microservice

Objective

Build a Review System Microservice that retrieves Agoda.com / Booking.com / Expedia reviews from an AWS S3 bucket, processes the data, and stores it in a relational database.

Business Context

Our company sources reviews from third-party providers. One such provider (Agoda.com / Booking.com / Expedia) uploads its daily review data in JSON Lines (.jl) format to a designated AWS S3 bucket. The goal is to design and implement a robust and scalable microservice that performs the following:

- Periodically pulls review files from the S3 bucket
- Parses and processes the data (including validation and transformation)
- Stores the processed review data in a relational database (e.g., PostgreSQL / MYSQL)

✏️ Problem Statement

Design and implement a microservice (in a language/framework of your choice — we prefer Go,PHP, or Java) that does the following:

Connects to AWS S3

- Retrieves `.jl` files uploaded daily to a specified path.
- Handles pagination/multiple files, only processing new files (idempotent processing).

Parses JSONL reviews

- Each line in the file is a well-formed JSON document representing a single review.

[Sample .jl]

Example:

Unset

```
{"hotelId": 10984, "platform": "Agoda", "hotelName":
"Oscar Saigon Hotel", "comment":
{"isShowReviewResponse": false, "hotelReviewId":
948353737, "providerId": 332, "rating": 6.4,
"checkInDateMonthAndYear": "April 2025",
"encryptedReviewData": "cZwJ6a6ZoFX2W5WwVXaJkA==",
"formattedRating": "6.4", "formattedReviewDate":
"April 10, 2025", "ratingText": "Good",
"responderName": "Oscar Saigon Hotel",
"responseDateText": "", "responseTranslateSource":
"en", "reviewComments": "Hotel room is basic and very
small. not much like pictures. few areas were getting
repaired. but since location is so accessible from all
main areas in district-1, i would prefer to stay here
again. Staff was good.", "reviewNegatives": "",
"reviewPositives": "", "reviewProviderLogo": "",
"reviewProviderText": "Agoda", "reviewTitle": "Perfect
location and safe but hotel under renovation ",
"translateSource": "en", "translateTarget": "en",
"reviewDate": "2025-04-10T05:37:00+07:00",
"reviewerInfo": {"countryName": "India",
"displayMemberName": "********", "flagName": "in",
"reviewGroupName": "Solo traveler", "roomTypeName":
"Premium Deluxe Double Room", "countryId": 35,
"lengthOfStay": 2, "reviewGroupId": 3, "roomTypeId":
0, "reviewerReviewedCount": 0, "isExpertReviewer":
false, "isShowGlobalIcon": false,
"isShowReviewedCount": false}, "originalTitle": "",
"originalComment": "", "formattedResponseDate": ""},
"overallByProviders": [{"providerId": 332, "provider":
"Agoda", "overallScore": 7.9, "reviewCount": 7070,
"grades": {"Cleanliness": 7.7, "Facilities": 7.2,
"Location": 9.1, "Room comfort and quality": 7.5,
"Service": 7.8, "Value for money": 7.8}}]}
```

Validates input data:

- Ensure required fields are present.
- Discard or log malformed entries.

Stores the data:

- Write reviews to a relational database (PostgreSQL or MySQL preferred).

The schema should support future expansion for:

- Reviews from multiple providers
- Multilingual support
- Extended rating systems

🔧 Requirements

The solution should include:

✅ Must Have

- Command-line tool, web service, or cron-job-compatible design
- Robust error handling: AWS failures, data issues, DB write failures
- Logging (To console or a file)
- Dockerized setup
- README with setup and run instructions

🌟 Nice to Have

- Unit tests or integration tests
- Support for concurrent file processing
- Modular codebase (using Clean Architecture/Layered Structure)
- Use ORM (e.g., SQLAlchemy, TypeORM)

💡 Evaluation Criteria

Category	Details
Code Quality	Clean, readable, maintainable, idiomatic code
Architecture	Appropriate design patterns, modularity, separation of concerns
Correctness	Meets all business and functional requirements
Robustness	Proper error handling, logging, retry mechanisms
Testing	Test coverage, structure, clarity

Infrastructure	Use of Docker, deployment setup, configuration management
Documentation	Clarity of README, ability to get started quickly
Scalability and Flexibility	Easy to expand to support multiple providers or real-time ingestion

---

📁 Submission Guidelines

Push code to a public GitHub repository (or shared private repo)

Include:

- The source code
- Docker setup
- README with:
  - Setup instructions
  - Design decisions
  - Assumptions
  - Instructions to run the ingestion flow

Target completion time: 3 - 7 days

📝 Bonus (Optional)

If you'd like to go beyond, consider adding:

- Support for multiple third-party providers
- Architecture Diagram
- Real-time streaming (e.g., AWS Lambda + SQS/Kinesis integration)
- REST API to fetch stored reviews
- CI/CD pipeline for testing and deployment

If you have any questions or clarifications, feel free to email us.
(ghanu@zuzuhs.com, ana@zuzuhs.com)

Good luck! 🍀 We're excited to see what you build.