Bazar.com Microservices System Design and Evaluation

Introduction

The architecture, functionality, design trade-offs, and run instructions of the Bazar.com microservices system are all covered in detail in this document. It enhances the performance metrics and source code that were submitted with this repository.

Architecture Overview

There are three main parts to the system:

1. Catalog Service: Product data is managed by a Spark-Java microservice. High availability and load distribution are guaranteed by two replicas (catalog-1, catalog-2).

2. Order Service: A Spark-Java microservice that manages requests for purchases. Two replicas (order-1 and order-2) replicate writes to peers and process POST /purchase/:id.

3. Frontend Proxy: A Spark-Java proxy that exposes /invalidate for back-end services to alert them of cache invalidation, load-balances requests, and offers an in-memory LRU cache for reads.

Every component can be run directly or containerized, and they all communicate via HTTP. Environment variables set proxy endpoints, peer URLs, replica IDs, and ports.

How It Works

GET /item/:id:

1. Proxy checks the Caffeine cache.

2. On a cache miss, it forwards to a round-robin catalog replica.

3. Response is cached and returned to the client.

POST /purchase/:id:

1. Proxy forwards to a round-robin order replica.

2. Order service calls /invalidate on the proxy to remove cached entries.

3. The order is replicated to the peer before confirmation.

Design Trade-offs

-Spark-Java is quick and lightweight, but it doesn't have sophisticated features like dependency injection.

-Caffeine Cache: Provides size and TTL limits, sacrificing strict freshness in favor of speed.

-Round-robin LB: straightforward and stateless, but it ignores load and health.

-Although it adds a small latency (about 6–8 ms), validation-before-write guarantees consistency.

-Simple peer-to-peer replication without failover or consensus.

Running the Program

Prerequisites:

- JDK 17+, Maven 3.9+, PowerShell (or Bash), optional Docker.

Steps:

1. Clone the repo:

```
git clone https://github.com/<you>/bazar-lab.git && cd bazar-lab
```

2. Build services:

```
cd catalog-service && mvn clean package
cd ../order-service && mvn clean package
cd ../frontend-proxy && mvn clean package
```

3. Launch services in separate terminals:

- Catalog-1: `$env:SERVER_ID="catalog-1"; $env:PORT="7000"; java -jar target/catalog-service.jar`

- Catalog-2: `$env:SERVER_ID="catalog-2"; $env:PORT="7001"; java -jar target/catalog-service.jar`

- Order-1: `$env:SERVER_ID="order-1"; $env:PORT="7100"; $env:FRONTEND="http://localhost:8000"; java -jar target/order-service.jar`

- Order-2: `$env:SERVER_ID="order-2"; $env:PORT="7101"; $env:FRONTEND="http://localhost:8000"; java -jar target/order-service.jar`

- Frontend: $env:PORT="8000"; $env:CATALOG_URLS="http://localhost:7000,http://localhost:7001"; $env:ORDER_URLS="http://localhost:7100,http://localhost:7101"; java -jar target/frontend-proxy.jar

Run Tests in PowerShell

Read Test: cache hit

curl.exe -i -s http://localhost:8000/item/123 -o NUL -w "%{http_code} %{time_total}`n"

Write Test: trigger purchase

curl.exe -i -s -X POST http://localhost:8000/purchase/123 -o NUL -w "%{http_code} %{time_total}`n"

Read after invalidation

curl.exe -i -s http://localhost:8000/item/123 -o NUL -w "%{http_code} %{time_total}`n"

Clean Up

Stop each service with Ctrl+C.

Conclusion

The design, functionality, trade-offs, and run instructions for the Bazar.com microservices are described in this document. It exhibits a scalable, reliable, and effective distributed system along with the source code and performance metrics.