

TRANSACTION SERVICE**Developed by-** Jatin Saini2024tm93002@wilp.bits-pilani.ac.inRepository Link: <https://github.com/jatin904/scalable-banking.git>

Table of Contents

TRANSACTION SERVICE.....	1
Description.....	1
Responsibilities.....	2
Transaction Service – ER Diagram.....	2
Database: transaction_db contains:.....	2
Microservice Architecture Diagram.....	4
Patterns Used.....	4
API Endpoints.....	4
Business Rules & Validations.....	4
Containerization with Docker.....	5
Inter-Service Communication.....	5
Dockerfile Highlights.....	5
Docker-Compose Setup.....	5
Verification Steps:.....	6
Health Endpoints.....	6
Kubernetes Deployment (Minikube).....	6
Screenshots:.....	6
Started docker:.....	6
Docker Desktop:.....	7
Grafana dashboard:.....	7
Prometheus:.....	9
API:.....	9
RabbitMQ: transaction_events.....	16
Swagger.....	18
Business validation:.....	26
Minikube:.....	27
Outcome.....	34

Description

The **Transaction** Service handles all money-movement operations within the Scalable Banking system. It records deposits, withdrawals, and transfers, applies business validations, and ensures reliability via idempotency and RabbitMQ-based events for downstream services (e.g., notifications).

Component	Technology
Language	Node.js (Express.js ES Modules)
Database	PostgreSQL
Messaging	RabbitMQ
Monitoring	Prometheus + Grafana
Containerization	Docker

Component	Technology
Orchestration	Kubernetes (Minikube)
Docs	Swagger / OpenAPI

Responsibilities

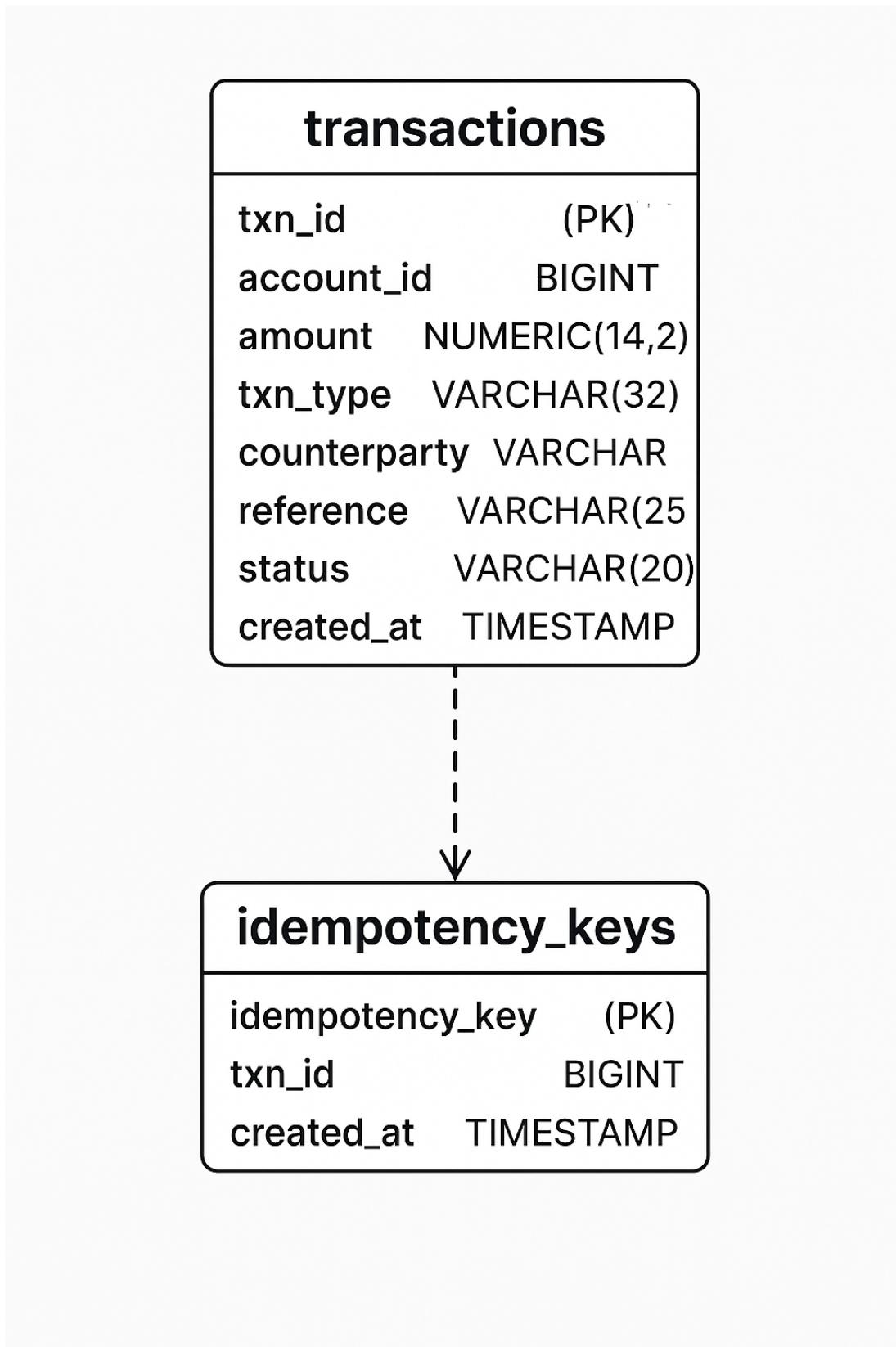
Record and validate Deposit and Transfer transactions
Enforce business rules (daily limits, balance, account status)
Maintain idempotency for safe retries
Publish transaction events (DEPOSIT_CREATED, TRANSFER_COMPLETED)
Expose REST APIs + /metrics endpoint for Prometheus
Log structured JSON with correlation IDs

Transaction Service – ER Diagram

Logical ER structure based on my PostgreSQL schema:

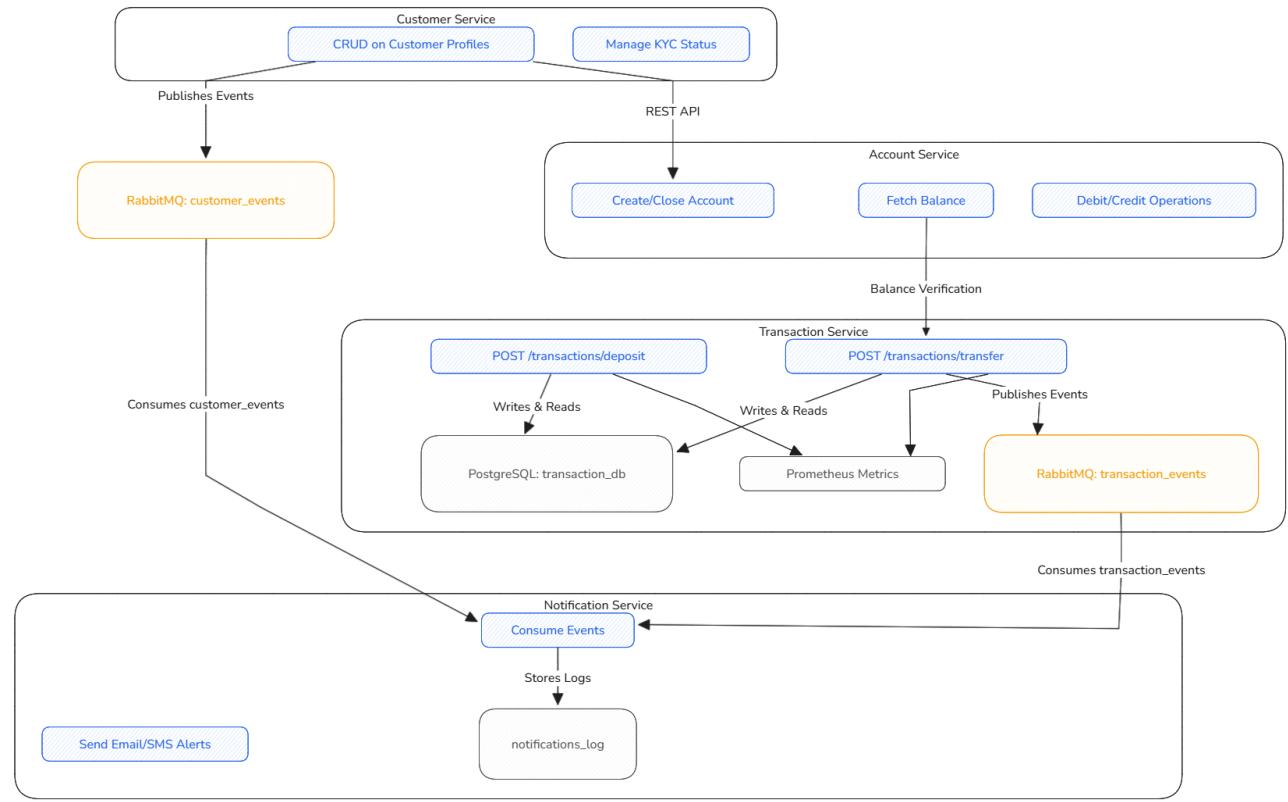
Database: transaction_db contains:

- transactions (txnid, account_id, amount, type, ref, created_at)
- idempotency_keys (key, txnid)



- Each transaction is uniquely identified by **txn_id**.
- **idempotency_keys** ensures API safety — no duplicate transactions.
- Logical **account_id** linkage to Account Service (no actual foreign key since each service owns its DB).

Microservice Architecture Diagram



Patterns Used

- Database per service:** No shared DBs.
- Asynchronous communication:** RabbitMQ → for DEPOSIT_CREATED and TRANSFER_COMPLETED events.
- Synchronous REST calls:** Transaction → Account (/balance, /debit, /credit).
- Data replication pattern:** Transaction stores only account_id and counterparty info (no cross-DB joins).

API Endpoints

Method	Endpoint	Description
GET	/health	Service health check
GET	/db-check	Database connectivity
POST	/transactions/deposit	Deposit into an account
POST	/transactions/transfer	Transfer funds between accounts
GET	/transactions	List all transactions
GET	/metrics	Prometheus metrics endpoint

Business Rules & Validations

Rule	Description	Example
Idempotency	No duplicate processing if same key reused Idempotency-Key: same-123	

Rule	Description	Example
Positive Amount	Must be > 0	Reject 0 or negative
Daily Transfer Limit	₹200,000 per account per day	Reject if exceeded
Account Status	Must be ACTIVE	Reject frozen/inactive
Sufficient Balance	No overdraft	Reject if balance < amount
Self-Transfer Prevention	from ≠ to account	Reject same IDs

Containerization with Docker

Containerize each microservice and its dependent components to ensure consistent, isolated, and reproducible runtime environments. Validate connectivity among services through docker-compose.

Inter-Service Communication

- **Synchronous REST** → Account Service
 - /balance (check balance)
 - /debit / /credit (update funds)
- **Asynchronous Events** → Notification Service via RabbitMQ
 - Queue: transaction_events
 - Event types:
 - DEPOSIT_CREATED
 - TRANSFER_COMPLETED

Dockerfile Highlights

Each Node.js microservice contains a lightweight Dockerfile:

```
FROM node:20-alpine
WORKDIR /app
COPY package*.json .
RUN npm install --production
COPY ..
EXPOSE 8082
CMD ["npm","start"]
```

This ensures a minimal footprint and quick build times.

Docker-Compose Setup

Transaction service is orchestrated via a single docker-compose.yml:

- **Networking:** Shared default network for inter-service REST/RabbitMQ communication.
- **Persistent Storage:** Postgres volume (txn_data) maintains data across container restarts.

- **Environment Variables:** Injected securely for DB credentials, ports, and RabbitMQ URLs.
- **Dependencies:** depends_on ensures DB and RabbitMQ are initialized before app startup.

Verification Steps:

Build & Start docker compose up –build

Check Containers docker ps

Health Endpoints

- `http://localhost:8082/health` → “Transaction Service running”
- `http://localhost:8083/health` → “Account Service running”
- `http://localhost:15672` → RabbitMQ management console (guest/guest)
- **Database Connectivity**
- `http://localhost:8082/db-check` → confirms Postgres connection.
- **Metrics**
- `http://localhost:8082/metrics` → Prometheus-formatted metrics exported successfully.

Kubernetes Deployment (Minikube)

kubectl create namespace banking

kubectl apply -f k8s/transaction-service/ -n banking

kubectl get pods -n banking

kubectl get svc -n banking

minikube service transaction-service -n banking –url

Screenshots:

Started docker:

The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows the project structure under "SCALABLE-BANKING".
 - k8s:** Contains "transaction-service" which includes "service.yaml" (shown in the code editor), "configmap.yaml", "openapi-configmap.yaml", "deployment.yaml", and "service.yaml".
 - notification-service:** Contains "init.sql" and "transactions.csv".
 - transaction-service:** Contains "init.ts", "routes", "transactionRoutes.js", "env", "accountClient.js", "db.js", "logger.js", "messageQueue.js", and "server.js".
 - src:** Contains "middleware" with "correlationid.js".
- CODE EDITOR:** Displays the content of "service.yaml".
- TERMINAL:** Shows the command "PS C:\Users\Jatin\scalable-banking\transaction-service> docker compose up --build" being run, followed by logs from RabbitMQ and Prometheus.
- STATUS BAR:** Shows "In 14 Col 22 Spaces: 2 UTF-8 ⚡ YAML ENG IN 12:17 AM 08-11-2025".

Docker Desktop:

The screenshot shows the Docker Desktop interface. On the left, a sidebar lists 'Containers', 'Images', 'Volumes', 'Builds', 'Docker Scout', and 'Extensions'. The main area is titled 'Containers' with a search bar and a filter 'Only show running containers'. It displays two containers: 'minikube' (Container ID: fb4d4ffe7818, Image: k8s-minikube) and 'transaction-service' (Container ID: -, Image: -). Container stats are shown at the top: CPU usage (26.33% / 800%) and Memory usage (1.4GB / 7.48GB). A 'Show charts' button is also present.

Name	Container ID	Image	Port(s)	CPU (%)	Memory usage...	Memory (%)	Disl	Actions
minikube	fb4d4ffe7818	k8s-minikube	-	22.72%	1.01GB / 4GB	25.27%	0B /	⋮ trash
transaction-service	-	-	-	3.05%	394.1MB / 53.61G	5.04%	0B /	⋮ trash

At the bottom, a taskbar shows 'Engine running', system status (RAM 7.91 GB, CPU 2.41%), and a notification for 'New version available' (version 1).

All inner service are running fine inside transaction-service container

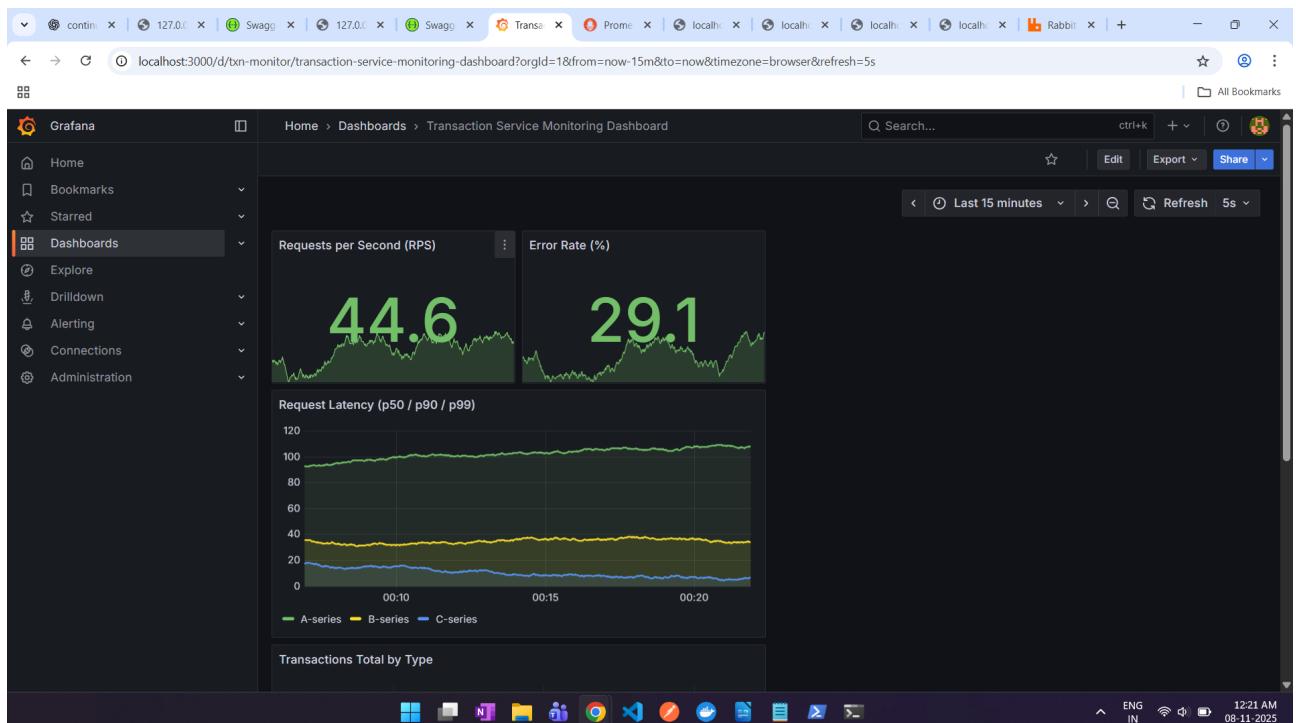
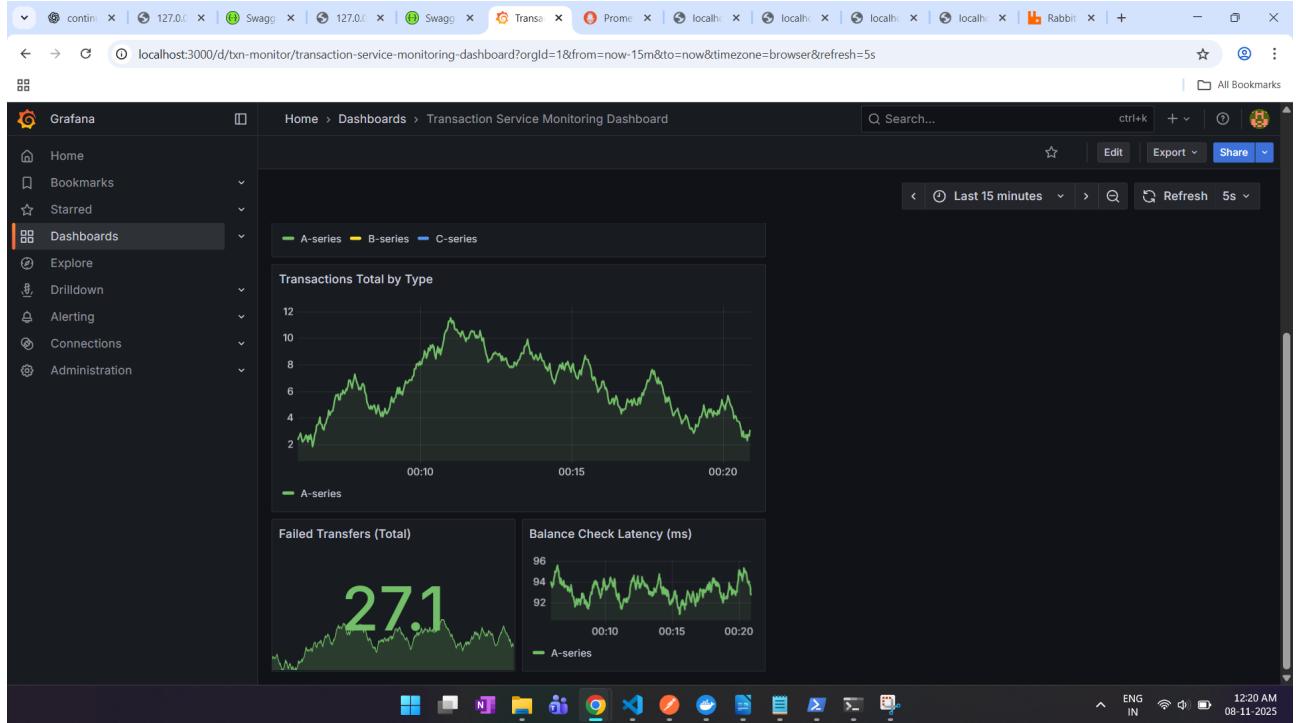
This screenshot shows the Docker Desktop interface with more containers listed in the table. The sidebar and top navigation are identical to the first screenshot. The table now includes nine items:

Name	Container ID	Image	Port(s)	CPU (%)	Memory usage...	Memory (%)	Disl	Actions
minikube	fb4d4ffe7818	k8s-minikube	-	18.96%	1.01GB / 4GB	25.29%	0B /	⋮ trash
transaction-service	-	-	-	3.37%	378.96MB / 53.6G	4.84%	0B /	⋮ trash
rabbitmq-1	047511e2d6b8	rabbitmq:3-15672:15672	:15672	1.39%	141.8MB / 7.66G	1.81%	0B /	⋮ trash
transaction-db-1	0f8b9328e0e1	postgres:1t-5434:5432	:5432	0%	28.27MB / 7.66G	0.36%	0B /	⋮ trash
account-service-1	814292d9be07	transaction:8083:8083	:8083	0%	29.18MB / 7.66G	0.37%	0B /	⋮ trash
notification-service-1	cc3c1c01f136	transaction:8084:8084	:8084	0.01%	22.39MB / 7.66G	0.29%	0B /	⋮ trash
transaction-service-1	bfc9a3bf876a	transaction:8082:8082	:8082	1.3%	42.46MB / 7.66G	0.54%	0B /	⋮ trash
prometheus	0ab823bcc20e	prom/promr:9090:9090	:9090	0.13%	25.62MB / 7.66G	0.33%	0B /	⋮ trash
grafana	24ad10b42108	grafana/grafana:3000:3000	:3000	0.54%	89.24MB / 7.66G	1.14%	0B /	⋮ trash

At the bottom, a taskbar shows 'Engine running', system status (RAM 7.90 GB, CPU 5.64%), and a notification for 'New version available' (version 1).

Grafana dashboard:

for visual metrics of transaction service



Prometheus:

The screenshot shows the Prometheus 'Targets' page. At the top, there are dropdown menus for 'Select scrape pool' and 'Filter by target health'. Below is a table with columns: Endpoint, Labels, Last scrape, and State. One entry is shown:

Endpoint	Labels	Last scrape	State
http://transaction-service:8082/metrics	instance="transaction-service:8082", job="transaction-service"	4.22s ago	UP

**Metric in Prometheus:**

The screenshot shows the Prometheus 'TSDB status' page. At the top, there is a table with two rows:

kind	351
type	152

Below is a section titled 'Top 10 series count by label value pairs' with a table:

Name	Count
instance=transaction-service:8082	148
job=transaction-service	148
method=GET	45
name=http_request_duration_seconds_bucket	32
status_code=200	22
name=nodejs_gc_duration_seconds_bucket	21
route=/favicon.ico	12
route=/db-check	11
route=/metrics	11
route=/health	11



API:POST <http://localhost:8082/transactions/deposit>

POST <http://localhost:8082/transactions/deposit>

Key	Value	Description
Content-Type	application/json	
Idempotency-Key	test-deposit-002	

```
{
  "success": true,
  "transaction": {
    "tx_id": "314",
    "account_id": "101",
    "amount": "100.00",
    "tx_type": "DEPOSIT",
    "counterparty": "SYSTEM:Deposit",
    "reference": "REF-1762524148184",
    "status": "SUCCESS",
    "created_at": "2025-11-07T14:02:28.184Z"
  }
}
```

POST <http://localhost:8082/transactions/deposit>

Body
{"account_id": 101, "amount": 100}

```
{
  "success": true,
  "transaction": {
    "tx_id": "314",
    "account_id": "101",
    "amount": "100.00",
    "tx_type": "DEPOSIT",
    "counterparty": "SYSTEM:Deposit",
    "reference": "REF-1762524148184",
    "status": "SUCCESS",
    "created_at": "2025-11-07T14:02:28.184Z"
  }
}
```

{ "success": true,

"transaction": {

"tx_id": "317",

"account_id": "101",

```

    "amount": "101.00",
    "txn_type": "DEPOSIT",
    "counterparty": "SYSTEM:Deposit",
    "reference": "REF-1762542603692",
    "status": "SUCCESS",
    "created_at": "2025-11-07T19:10:03.692Z"
}

}

```

If same request sent again getting reused:true

The screenshot shows the Postman interface with a collection named 'Scalable Banking - Transaction Microservi...'. A POST request is selected with the URL `http://localhost:8082/transactions/deposit`. The 'Body' tab is active, showing the following JSON payload:

```

1 {
2   "account_id": 101,
3   "amount": 100
4 }

```

The response status is 200 OK, with a timestamp of 12:27 AM 08-11-2025.

POST <http://localhost:8082/transactions/transfer>

The screenshot shows the Postman interface with a successful API call. The URL is <http://localhost:8082/transactions/transfer>. The response code is 201 Created, and the response body is:

```

1 {
2   "success": true,
3   "sender_transaction": {
4     "txnid": "315",
5     "account_id": "10",
6     "amount": "2802.00",
7     "txntype": "TRANSFER_OUT",
8     "counterparty": "TO:12",
9     "reference": "REF-OUT-1762541913324",
10    "status": "SUCCESS",
11    "created_at": "2025-11-07T18:58:33.324Z"
12  },
13  "receiver_transaction": {
14    "txnid": "316",
15    "account_id": "12"
}

```

The screenshot shows the Postman interface with a successful API call. The URL is <http://localhost:8082/transactions/transfer>. The response code is 201 Created, and the response body is:

```

1 {
2   "from_account_id": 10,
3   "to_account_id": 12,
4   "amount": 2802
5 }

```

```

"success": true,
"sender_transaction": {
  "txnid": "315",
  "account_id": "10",
}

```

```

    "amount": "2002.00",
    "txn_type": "TRANSFER_OUT",
    "counterparty": "TO:12",
    "reference": "REF-OUT-1762541913324",
    "status": "SUCCESS",
    "created_at": "2025-11-07T18:58:33.324Z"
},
"receiver_transaction": {
    "txn_id": "316",
    "account_id": "12",
    "amount": "2002.00",
    "txn_type": "TRANSFER_IN",
    "counterparty": "FROM:10",
    "reference": "REF-IN-1762541913332",
    "status": "SUCCESS",
    "created_at": "2025-11-07T18:58:33.324Z"
}
}

```

If same request is send again, getting reused:true

The screenshot shows the Postman interface with a successful API call to `http://localhost:8082/transactions/transfer`. The response status is `200 OK` with a response time of `30 ms` and a size of `615 B`. The response body is a JSON object:

```

1  {
2     "success": true,
3     "transaction": {
4         "txn_id": "316",
5         "account_id": "10",
6         "amount": "2002.00",
7         "txn_type": "TRANSFER_OUT",
8         "counterparty": "TO:12",
9         "reference": "REF-OUT-1762541913324",
10        "status": "SUCCESS",
11        "created_at": "2025-11-07T18:58:33.324Z"
12    },
13    "reused": true
14 }

```

Get <http://localhost:8082/transactions/16>

The screenshot shows the Postman interface with a successful API call. The URL is <http://localhost:8082/transactions/16>. The response code is 200 OK, and the response body is a JSON object:

```

1  {
2     "success": true,
3     "transaction": {
4         "txn_id": "16",
5         "account_id": "21",
6         "amount": "133.53",
7         "txn_type": "DEPOSIT",
8         "counterparty": "IMPS:External",
9         "reference": "REF20250827-JOQW5D",
10        "status": "SUCCESS",
11        "created_at": "2024-11-21T10:07:47.000Z"
12    }
13 }

```

{
 "success": true,

 "transaction": {
 "txn_id": "16",
 "account_id": "21",
 "amount": "133.53",
 "txn_type": "DEPOSIT",
 "counterparty": "IMPS:External",
 "reference": "REF20250827-JOQW5D",
 "status": "SUCCESS",
 "created_at": "2024-11-21T10:07:47.000Z"
 }
 }

Get <http://localhost:8082/metrics>

Scalable Banking - Transaction Microservice / Metrics

GET http://localhost:8082/metrics

Body Cookies Headers (9) Test Results

```

286 # TYPE balance_check_latency_ms histogram
287 balance_check_latency_ms_bucket{le="5"} 0
288 balance_check_latency_ms_bucket{le="10"} 0
289 balance_check_latency_ms_bucket{le="20"} 2
290 balance_check_latency_ms_bucket{le="50"} 3
291 balance_check_latency_ms_bucket{le="100"} 3
292 balance_check_latency_ms_bucket{le="200"} 3
293 balance_check_latency_ms_bucket{le="500"} 3
294 balance_check_latency_ms_bucket{le="1000"} 3
295 balance_check_latency_ms_sum 58.5078869999852
296 balance_check_latency_ms_count 3
297
  
```

200 OK 12 ms 18.38 KB

Get http://localhost:8083/accounts/10/balance

Scalable Banking - Transaction Microservice (Full Test Suite) / Account Balance - Account 10

GET http://localhost:8083/accounts/10/balance

Body Cookies Headers (8) Test Results

```

1 {
2   "success": true,
3   "account_id": 10,
4   "balance": 25000,
5   "status": "ACTIVE"
6 }
  
```

200 OK 56 ms 357 B

RabbitMQ: transaction_events

The screenshot shows the RabbitMQ Management Interface with the 'Queues' tab selected. The queue 'transaction_events' is listed with the following details:

Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/	transaction_events	classic	D	running	0	0	0	0.00/s	0.00/s	0.00/s

Below the table, there is a link to 'Add a new queue'.

At the bottom of the interface, there is a navigation bar with links to 'HTTP API', 'Documentation', 'Tutorials', 'New releases', 'Commercial edition', 'Commercial support', 'Discussions', 'Discord', 'Plugins', and 'GitHub'.

The system status bar at the bottom right shows the date and time as 08-11-2025 12:37 AM.

The screenshot shows the RabbitMQ Management Interface with the 'Queue transaction_events' page selected. The page displays various metrics and configuration details for the 'transaction_events' queue.

Queued messages (last minute):

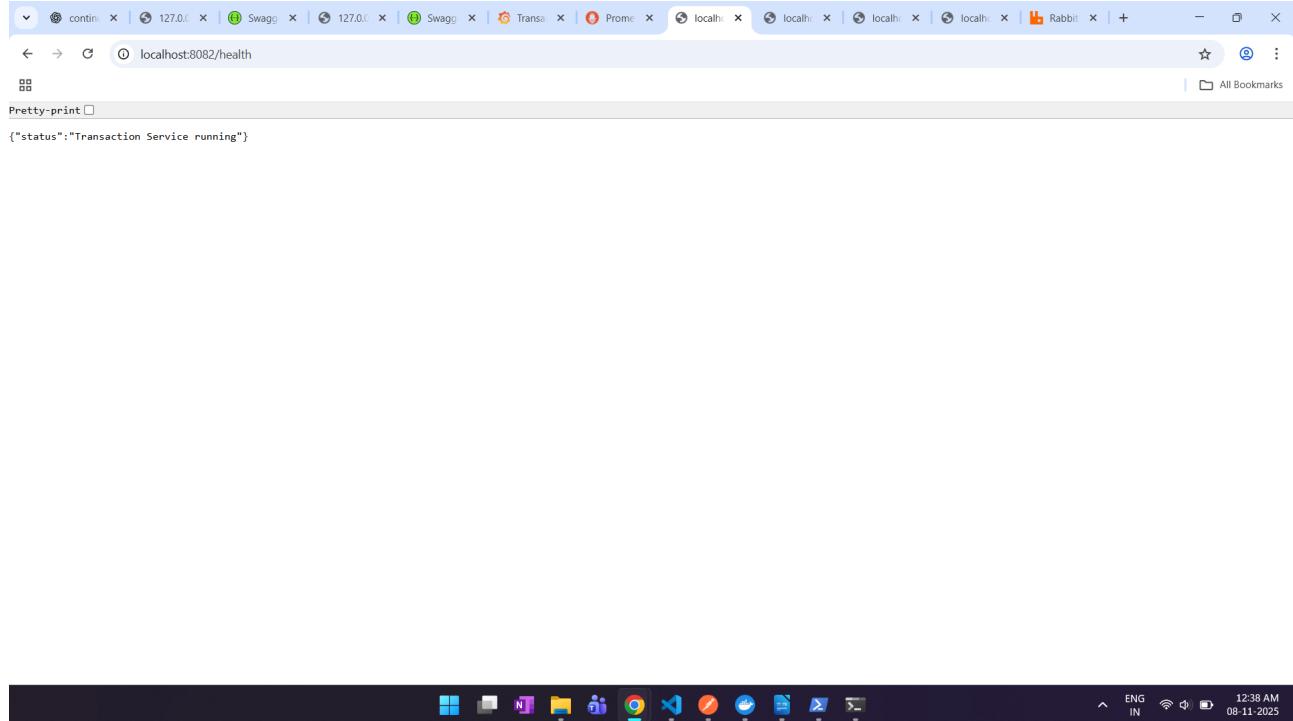
Message rates (last minute):

Action	Rate
Publish	0.00/s
Deliver (manual ack)	0.00/s
Deliver (auto ack)	0.00/s
Consumer ack	0.00/s
Redelivered	0.00/s
Get (auto ack)	0.00/s
Get (empty)	0.00/s
Get (manual ack)	0.00/s

Details:

Feature	Value	State	Consumers	Capacity	Messages	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
durable	true	idle	1	100%	?	0	0	0	0 B	0 B	0 B
queue storage version	1				Message body bytes	0 B	0 B	0 B	0 B	0 B	0 B
Consumer capacity	?				Process memory	10 KB					

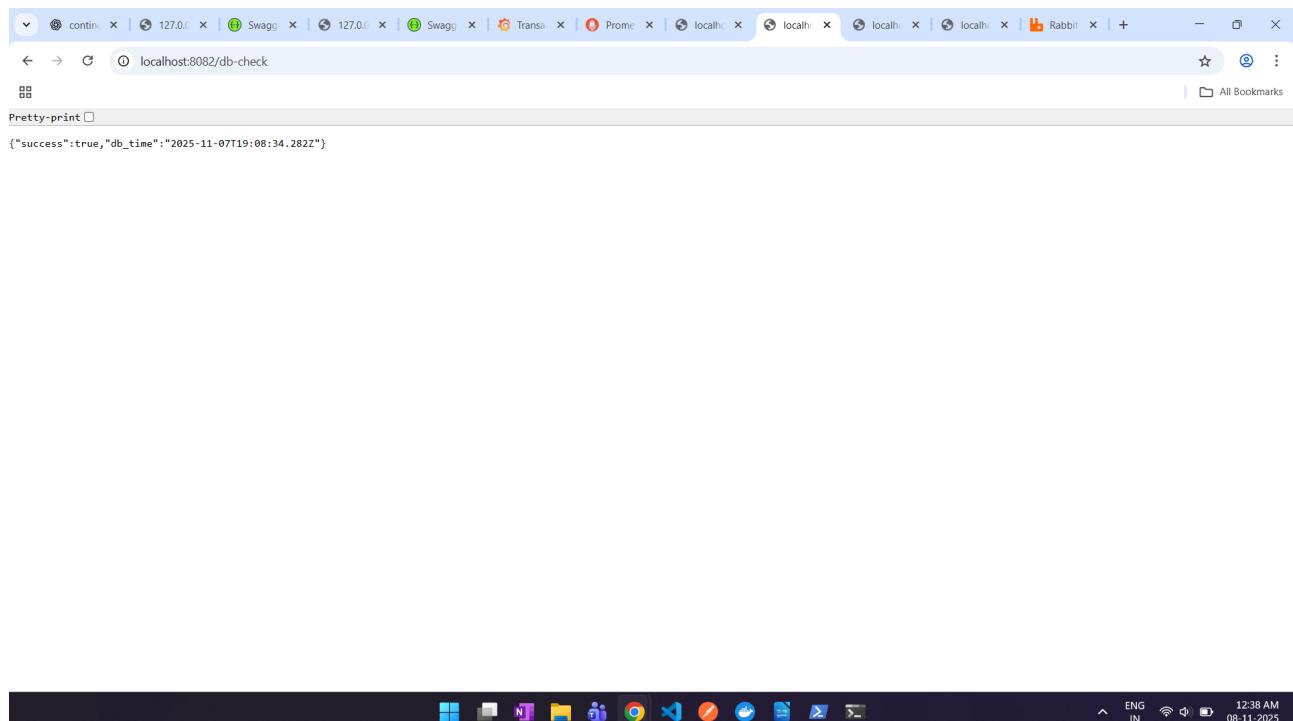
The system status bar at the bottom right shows the date and time as 08-11-2025 12:40 AM.

Transaction-service health check:

A screenshot of a Microsoft Edge browser window. The address bar shows the URL `localhost:8082/health`. The main content area displays a JSON response:

```
{"status": "Transaction Service running"}
```

. The browser has multiple tabs open, including ones for Swagger, Prometheus, and RabbitMQ. The taskbar at the bottom shows various pinned application icons.

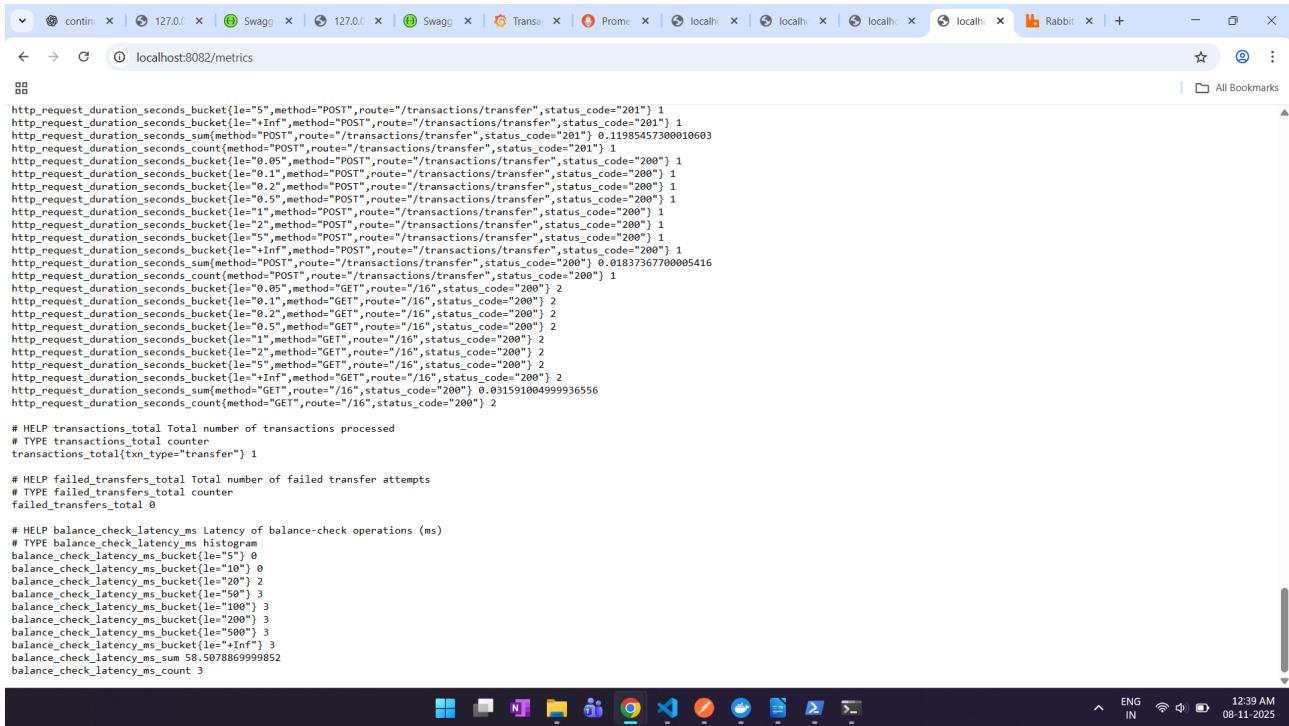
Transaction database health check:

A screenshot of a Microsoft Edge browser window. The address bar shows the URL `localhost:8082/db-check`. The main content area displays a JSON response:

```
{"success": true, "db_time": "2025-11-07T19:08:34.282Z"}
```

. The browser has multiple tabs open, including ones for Swagger, Prometheus, and RabbitMQ. The taskbar at the bottom shows various pinned application icons.

Transaction-service metrics logging:



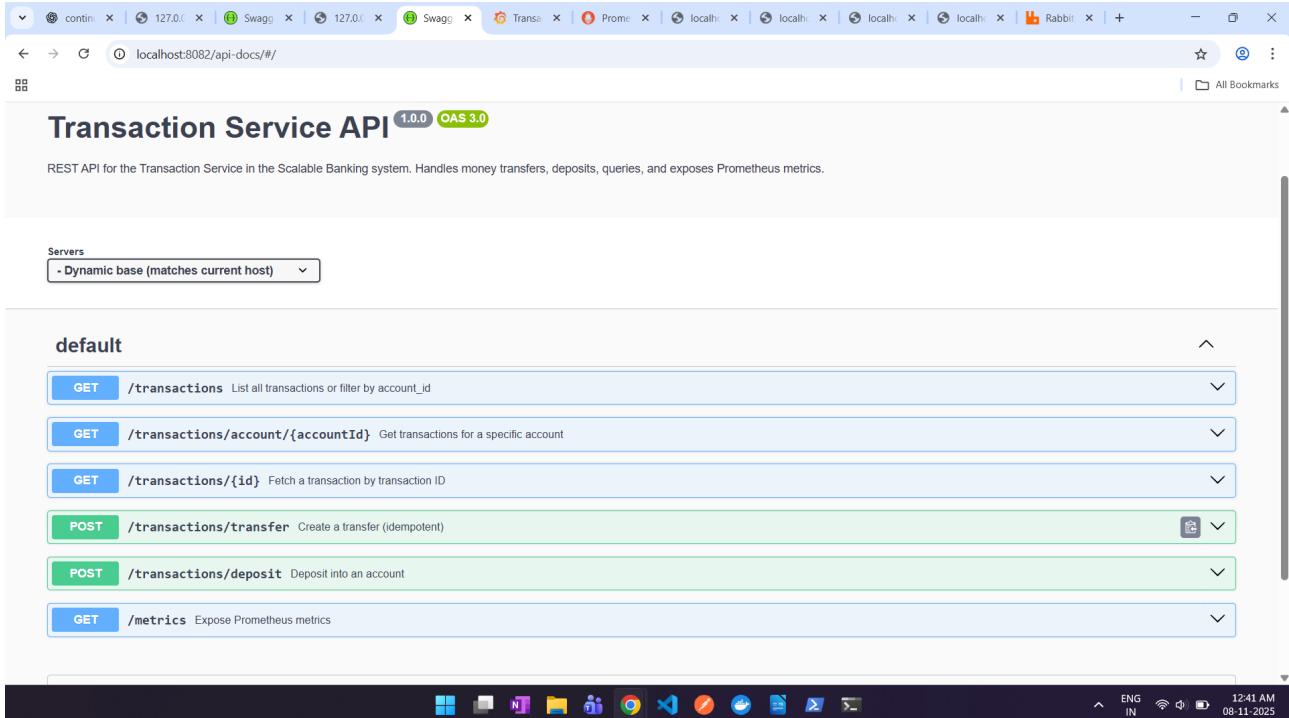
```

http_request_duration_seconds_bucket{le="5",method="POST",route="/transactions/transfer",status_code="201"} 1
http_request_duration_seconds_bucket{le="10",method="POST",route="/transactions/transfer",status_code="201"} 1
http_request_duration_seconds_bucket{le="15",method="POST",route="/transactions/transfer",status_code="201"} 1
http_request_duration_seconds_bucket{le="20",method="POST",route="/transactions/transfer",status_code="201"} 0.01985457300010603
http_request_duration_seconds_count{method="POST",route="/transactions/transfer",status_code="201"} 1
http_request_duration_seconds_bucket{le="0.05",method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_bucket{le="0.1",method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_bucket{le="0.2",method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_bucket{le="0.5",method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_bucket{le="1",method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_bucket{le="2",method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_bucket{le="5",method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_bucket{le="10",method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_bucket{le="15",method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_bucket{le="20",method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_bucket{le="0.05",method="GET",route="/16",status_code="200"} 1
http_request_duration_seconds_bucket{le="0.1",method="GET",route="/16",status_code="200"} 2
http_request_duration_seconds_bucket{le="0.2",method="GET",route="/16",status_code="200"} 2
http_request_duration_seconds_bucket{le="0.5",method="GET",route="/16",status_code="200"} 2
http_request_duration_seconds_bucket{le="1",method="GET",route="/16",status_code="200"} 2
http_request_duration_seconds_bucket{le="2",method="GET",route="/16",status_code="200"} 2
http_request_duration_seconds_bucket{le="5",method="GET",route="/16",status_code="200"} 2
http_request_duration_seconds_bucket{le="10",method="GET",route="/16",status_code="200"} 2
http_request_duration_seconds_bucket{le="15",method="GET",route="/16",status_code="200"} 2
http_request_duration_seconds_bucket{le="20",method="GET",route="/16",status_code="200"} 2
http_request_duration_seconds_bucket{le="0.05",method="GET",route="/16",status_code="200"} 0.01837367700005416
http_request_duration_seconds_count{method="POST",route="/transactions/transfer",status_code="200"} 2
# HELP transactions_total Total number of transactions processed
# TYPE transactions_total counter
transactions_total{tx_type="transfer"} 1
# HELP failed_transfers_total Total number of failed transfer attempts
# TYPE failed_transfers_total counter
failed_transfers_total 0
# HELP balance_check_latency_ms Latency of balance-check operations (ms)
# TYPE balance_check_latency_ms histogram
balance_check_latency_ms_bucket{le="5"} 0
balance_check_latency_ms_bucket{le="10"} 0
balance_check_latency_ms_bucket{le="20"} 2
balance_check_latency_ms_bucket{le="50"} 3
balance_check_latency_ms_bucket{le="100"} 3
balance_check_latency_ms_bucket{le="200"} 3
balance_check_latency_ms_bucket{le="500"} 3
balance_check_latency_ms_bucket{le="Inf"} 3
balance_check_latency_ms_sum 58.5078869999852
balance_check_latency_ms_count 3

```

Swagger

Showing all API of transaction-service



Transaction Service API 1.0.0 OAS 3.0

REST API for the Transaction Service in the Scalable Banking system. Handles money transfers, deposits, queries, and exposes Prometheus metrics.

Servers

- Dynamic base (matches current host) ▾

default

- GET** /transactions List all transactions or filter by account_id
- GET** /transactions/account/{accountId} Get transactions for a specific account
- GET** /transactions/{id} Fetch a transaction by transaction ID
- POST** /transactions/transfer Create a transfer (idempotent)
- POST** /transactions/deposit Deposit into an account
- GET** /metrics Expose Prometheus metrics

The screenshot shows a browser window with multiple tabs open, including several Swagger UI instances. The active tab is for the Prometheus metrics endpoint at localhost:8082/metrics. The interface includes a 'GET /metrics' button, parameters section (empty), and a responses section. It displays curl and request URL examples, and a large block of Prometheus metric output in the response body. The response body starts with:

```
curl -X 'GET' \
'http://localhost:8082/metrics' \
-H 'accept: text/plain'

http_request_duration_seconds_bucket{le="0.01",method="POST",route="/transaction/deposit",status_code="201"} 1
http_request_duration_seconds_bucket{le="0.05",method="POST",route="/transactions/deposit",status_code="201"} 1
http_request_duration_seconds_bucket{le="Inf",method="POST",route="/transactions/deposit",status_code="201"} 1
http_request_duration_seconds_sum{method="POST",route="/transactions/deposit",status_code="201"} 0.028883393000112847
http_request_duration_seconds_count{method="POST",route="/transactions/deposit",status_code="201"} 1

# HELP transactions_total Total number of transactions processed
# TYPE transactions total counter
```

The screenshot shows a browser window with multiple tabs open, including several Swagger UI instances. The active tab is for the listTransactions endpoint at localhost:8082/api-docs/#/default/listTransactions. The interface includes a 'GET /transactions' button, parameters section (with account_id set to 10), and a responses section. It displays curl and request URL examples, and a large block of transaction data in the response body. The response body starts with:

```
curl -X 'GET' \
'http://localhost:8082/transactions?account_id=10' \
-H 'accept: application/json'
```

Curl

```
curl -X 'GET' \
'http://localhost:8082/transactions?account_id=10' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:8082/transactions?account_id=10
```

Server response

Code Details

200 Response body

```
{
  "success": true,
  "count": 15,
  "transactions": [
    {
      "tx_id": "315",
      "account_id": "10",
      "amount": "2002.00",
      "tx_type": "TRANSFER_OUT",
      "counterparty": "TO:12",
      "reference": "REF-OUT-1762541913324",
      "status": "SUCCESS",
      "created_at": "2025-11-07T18:58:33.324Z"
    },
    {
      "tx_id": "312",
      "account_id": "10",
      "amount": "2002.00",
      "tx_type": "TRANSFER_OUT",
      "counterparty": "TO:12",
      "reference": "REF-OUT-1762512250574",
      "status": "SUCCESS",
      "created_at": "2025-11-07T10:44:10.573Z"
    },
    {
      "tx_id": "309",
      "account_id": "10",
      ...
    }
  ]
}
```

Response headers

Download

account_id Filter transactions by account ID

integer (query) 10

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8082/transactions?account_id=10' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:8082/transactions?account_id=10
```

Server response

Code Details

200 Response body

```
{
  "success": true,
  "count": 15,
  "transactions": [
    {
      "tx_id": "315",
      "account_id": "10",
      "amount": "2002.00",
      "tx_type": "TRANSFER_OUT",
      "counterparty": "TO:12",
      "reference": "REF-OUT-1762541913324",
      ...
    }
  ]
}
```

The screenshot shows a Swagger UI interface for a REST API. The URL in the browser is `localhost:8082/api-docs/#/default/getTransactionsByAccountId`. The request method is `GET` to the endpoint `/transactions/account/{accountId}`. The description is "Get transactions for a specific account".

Parameters:

Name	Description
<code>accountId</code> * required	Account ID to fetch transactions for
<code>integer</code> (path)	101

Responses:

Code	Description	Links
200	Transactions found for the account	No links

Media type: application/json

Example Value | Schema

```
{
  "success": true,
  "count": 2,
  "data": [
    {
      "id": "7e0dab25-9b32-42de-bcbf-19f3d431b812",
      "type": "transfer",
      "amount": 100.00
    }
  ]
}
```

The screenshot shows the results of the executed GET request. The URL in the browser is `localhost:8082/api-docs/#/default/getTransactionsByAccountId`.

Parameters:

Name	Description
<code>accountId</code> * required	Account ID to fetch transactions for
<code>integer</code> (path)	101

Responses:

Curl:

```
curl -X 'GET' \
'http://localhost:8082/transactions/account/101' \
-H 'accept: application/json'
```

Request URL:

```
http://localhost:8082/transactions/account/101
```

Server response:

Code	Details
200	Response body

```
{
  "success": true,
  "count": 2,
  "data": [
    {
      "tx_id": "101",
      "account_id": "101",
      "amount": "101.00",
      "tx_type": "DEPOSIT",
      "counterparty": "SYSTEM:Deposit",
      "reference": "REF-1762542603692"
    }
  ]
}
```

localhost:8082/api-docs/#/default/getTransactionsByAccountId

```
curl -X 'GET' \
  'http://localhost:8082/transactions/account/101' \
  -H 'accept: application/json'
```

Request URL
<http://localhost:8082/transactions/account/101>

Server response

Code	Details
200	Response body <pre>{ "success": true, "count": 3, "data": [{ "tx_id": "317", "account_id": "101", "amount": "101.00", "tx_type": "DEPOSIT", "counterparty": "SYSTEM:Deposit", "reference": "REF-1762542603692", "status": "SUCCESS", "created_at": "2025-11-07T19:10:03.692Z" }, { "tx_id": "314", "account_id": "101", "amount": "100.00", "tx_type": "DEPOSIT", "counterparty": "SYSTEM:Deposit", "reference": "REF-1762524148184", "status": "SUCCESS", "created_at": "2025-11-07T14:02:28.184Z" }, { "tx_id": "311", "account_id": "101", "amount": "100.00", "tx_type": "DEPOSIT", "counterparty": "SYSTEM:Deposit", "reference": "REF-1762524148184", "status": "SUCCESS", "created_at": "2025-11-07T14:02:28.184Z" }] }</pre> <p>Download</p>

Response headers

12:46 AM 08-11-2025

localhost:8082/api-docs/#/default/getTransactionById

GET /transactions/{id} Fetch a transaction by transaction ID

Parameters

Name	Description
id * required	string (path)
10	

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8082/transactions/10' \
  -H 'accept: application/json'
```

Request URL
<http://localhost:8082/transactions/10>

Server response

Code	Details
200	Response body <pre>{ "tx_id": "311", "account_id": "101", "amount": "100.00", "tx_type": "DEPOSIT", "counterparty": "SYSTEM:Deposit", "reference": "REF-1762524148184", "status": "SUCCESS", "created_at": "2025-11-07T14:02:28.184Z" }</pre>

12:47 AM 08-11-2025

Curl

```
curl -X 'GET' \
'http://localhost:8082/transactions/10' \
-H 'accept: application/json'
```

Request URL

<http://localhost:8082/transactions/10>

Server response

Code	Details
200	Response body <pre>{ "success": true, "transaction": { "tx_id": "10", "account_id": "47", "amount": "100.00", "tx_type": "DEPOSIT", "counterparty": "NEFT:External", "reference": "REF20250827-TSTTIP", "status": "SUCCESS", "created_at": "2023-03-24T15:28:45.000Z" } }</pre> <p>Copy Download</p> Response headers <pre>access-control-allow-origin: * date: Fri, 07 Nov 2025 19:16:54 GMT content-length: 226 content-type: application/json; charset=utf-8 etag: W/"e2-Y3G154xnbgEyK6yq+7Kh2M8Tk8" keep-alive: timeout=5 x-correlation-id: d998e59a-02fe-4ee3-a93e-b266f5f790c x-powered-by: Express</pre>

Windows taskbar icons: Continuum, 127.0.0.1, Swagger UI, 127.0.0.1, Swagger UI, Transaction API, Prometheus, localhost:8082, localhost:8082, localhost:8082, RabbitMQ.

POST /transactions/transfer Create a transfer (idempotent)

[Cancel](#)

Parameters

Name	Description
Idempotency-Key <small>* required</small>	Unique key to ensure idempotent request handling. test-transfer-003 <small>(header)</small>
X-Correlation-ID	Correlation ID for traceability. X-Correlation-ID <small>(header)</small>

Request body required

application/json

Edit Value Schema

```
{
  "from_account_id": 101,
  "to_account_id": 202,
  "amount": 250.5
}
```

Windows taskbar icons: Continuum, 127.0.0.1, Swagger UI, 127.0.0.1, Swagger UI, Transaction API, Prometheus, localhost:8082, localhost:8082, localhost:8082, RabbitMQ.

The screenshot shows a browser window with multiple tabs open. The active tab displays a curl command for creating a transfer:

```
curl -X 'POST' \
  'http://localhost:8082/transactions/transfer' \
  -H 'accept: application/json' \
  -H 'Idempotency-Key: test-transfer-003' \
  -H 'Content-Type: application/json' \
  -d '{
    "from_account_id": 101,
    "to_account_id": 102,
    "amount": 250.5
}'
```

Below the curl command, the Request URL is listed as <http://localhost:8082/transactions/transfer>. The Server response section shows a 503 Error: Service Unavailable. The Response body contains the JSON object:

```
{
  "success": false,
  "message": "Account Service temporarily unavailable (circuit open)"
}
```

The Response headers section shows the following headers:

```
access-control-allow-origin: *
access-control-expose-headers: X-Correlation-ID
connection: keep-alive
content-length: 86
content-type: application/json; charset=utf-8
date: Fri, 07 Nov 2025 19:18:26 GMT
etag: W/"54-Http4fKuyXizpXjbz41/ng4AB24"
keep-alive: timeout=5
x-correlation-id: d729a8f0-8fe1-486e-8ac3-e0cd8646ac0
x-powered-by: Express
```

The bottom of the browser window shows the Windows taskbar with various pinned icons.

The screenshot shows a browser window with multiple tabs open. The active tab displays the configuration for the POST /transactions/transfer endpoint:

POST /transactions/transfer Create a transfer (idempotent)

Parameters

Name	Description
Idempotency-Key * required	Unique key to ensure idempotent request handling. test-transfer-003 <small>string (header)</small>
X-Correlation-ID	Correlation ID for traceability. X-Correlation-ID <small>string (header)</small>

Request body required

application/json

Edit Value | Schema

```
{
  "from_account_id": 10,
  "to_account_id": 12,
  "amount": 250.5
}
```

The bottom of the browser window shows the Windows taskbar with various pinned icons.

Curl

```
curl -X 'POST' \
  'http://localhost:8082/transactions/transfer' \
  -H 'accept: application/json' \
  -H 'Idempotency-Key: test-transfer-003' \
  -H 'Content-Type: application/json' \
  -d '{
    "from_account_id": 10,
    "to_account_id": 12,
    "amount": 250.5
}'
```

Request URL

`http://localhost:8082/transactions/transfer`

Server response

Code	Details
201	Response body

```
{
  "success": true,
  "sender_transaction": {
    "tx_id": "318",
    "account_id": "10",
    "amount": "250.50",
    "txn_type": "TRANSFER_OUT",
    "counterparty": "TO:12",
    "reference": "REF-OUT-1762543281437",
    "status": "SUCCESS",
    "created_at": "2025-11-07T19:21:21.437Z"
  },
  "receiver_transaction": {
    "tx_id": "319",
    "account_id": "12",
    "amount": "250.50",
    "txn_type": "TRANSFER_IN",
    "counterparty": "FROM:10",
    "reference": "REF-IN-1762543281438",
    "status": "SUCCESS",
    "created_at": "2025-11-07T19:21:21.437Z"
  }
}
```

[Download](#)

Curl

```
curl -X 'POST' \
  'http://localhost:8082/transactions/transfer' \
  -H 'accept: application/json' \
  -H 'Idempotency-Key: test-transfer-003' \
  -H 'Content-Type: application/json' \
  -d '{
    "from_account_id": 10,
    "to_account_id": 12,
    "amount": 2500000000
}'
```

Request URL

`http://localhost:8082/transactions/transfer`

Responses

Execute	Clear
---------	-------

Curl

```
curl -X 'POST' \
  'http://localhost:8082/transactions/transfer' \
  -H 'accept: application/json' \
  -H 'Idempotency-Key: test-transfer-003' \
  -H 'Content-Type: application/json' \
  -d '{
    "from_account_id": 10,
    "to_account_id": 12,
    "amount": 2500000000
}'
```

The screenshot shows a browser window with multiple tabs open, focusing on the API documentation for creating a transfer. The main content area displays the following:

- Response headers:**

```
access-control-allow-origin: *
access-control-expose-headers: X-Correlation-ID
connection: keep-alive
content-length: 241
content-type: application/json; charset=utf-8
date: Fri, 07 Nov 2025 19:22:54 GMT
etag: W/\"f1-2c9b5vratm1n0ufr3y3ldrZQ"
keep-alive: true
x-correlation-id: acd7a2fe-6d4a-4147-b846-e50905d4bcfe
x-powered-by: Express
```
- Responses:**

Code	Description	Links
201	Transfer created successfully	No links
400	Bad request or business rule violation	No links
409	Duplicate request (idempotency conflict)	No links
- Example Value:**

```
{
  "id": "7e0ddab25-ab32-42de-bcbd-19f3d431b812",
  "type": "transfer",
  "from_account_id": 101,
  "to_account_id": 202,
  "amount": 250,
  "status": "completed",
  "created_at": "2025-11-06T10:00:00Z"
}
```

Metrics getting logged: no. of transaction failed, success and response time

The screenshot shows a browser window displaying metrics logs. The logs include the following:

- http_request_duration_seconds bucket** (multiple entries):


```
http_request_duration_seconds.bucket{le="+Inf",method="GET",route="/10",status_code="200"} 1
http_request_duration_seconds.sum{method="GET",route="/10",status_code="200"} 0.01708155500004068
http_request_duration_seconds.count{method="GET",route="/10",status_code="200"} 1
...
http_request_duration_seconds.bucket{le="0.05",method="POST",route="/transactions/transfer",status_code="503"} 1
http_request_duration_seconds.bucket{le="0.1",method="POST",route="/transactions/transfer",status_code="503"} 1
http_request_duration_seconds.bucket{le="0.2",method="POST",route="/transactions/transfer",status_code="503"} 1
...
http_request_duration_seconds.bucket{le="0.5",method="POST",route="/transactions/transfer",status_code="503"} 1
http_request_duration_seconds.bucket{le="1",method="POST",route="/transactions/transfer",status_code="503"} 1
http_request_duration_seconds.bucket{le="2",method="POST",route="/transactions/transfer",status_code="503"} 1
http_request_duration_seconds.bucket{le="5",method="POST",route="/transactions/transfer",status_code="503"} 1
http_request_duration_seconds.bucket{le="10",method="POST",route="/transactions/transfer",status_code="503"} 1
http_request_duration_seconds.sum{method="POST",route="/transactions/transfer",status_code="503"} 0.030672572000185028
http_request_duration_seconds.count{method="POST",route="/transactions/transfer",status_code="503"} 1
...

```
- # HELP transactions_total** Total number of transactions processed


```
# HELP transactions_total Total number of transactions processed
# TYPE transactions_total counter
transactions_total{tx_type="transfer"} 2
transactions_total{tx_type="deposit"} 1
```
- # HELP failed_transfers_total** Total number of failed transfer attempts


```
# HELP failed_transfers_total Total number of failed transfer attempts
# TYPE failed_transfers_total counter
failed_transfers_total 0
```
- # HELP balance_check_latency_ms** Latency of balance-check operations (ms)


```
# TYPE balance_check_latency_ms histogram
balance_check_latency_ms.bucket{le="5"} 0
balance_check_latency_ms.bucket{le="10"} 1
balance_check_latency_ms.bucket{le="20"} 6
balance_check_latency_ms.bucket{le="50"} 8
balance_check_latency_ms.bucket{le="100"} 8
balance_check_latency_ms.bucket{le="200"} 8
balance_check_latency_ms.bucket{le="500"} 8
balance_check_latency_ms.bucket{le="Inf"} 8
balance_check_latency_ms.sum 132.74823499983177
balance_check_latency_ms.count 8
```

Same check be checked from docker as well in logs are in JSON format:

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure under "SCALABLE-BANKING". Key files include deployment.yaml, service.yaml, deployment.yaml, transactionRoutes.js, db.js, and init.sql.
- Terminal View:** Displays the command "PS C:\Users\Jatin\scalable-banking\transaction-service> docker compose up --build" and its output, which includes logs from transaction-service-1 indicating a successful POST request for a deposit.
- Bottom Status Bar:** Shows the current user (jatin), date (08-11-2025), and time (12:57 AM).

Business validation:

The screenshot shows the Postman application interface with the following details:

- Sidebar:** Shows "jatin's Workspace" with collections like "Scalable Banking - Transaction Microservice..." containing various API endpoints such as Health Check, DB Check, Deposit Transaction, Transfer Transaction, Fetch Transactions for Account 10, Metrics (Prometheus), Account Service Health, and Account Balance - Account 10.
- Request Panel:** A POST request to "http://localhost:8082/transactions/transfer" is being tested. The "Body" tab is selected, showing a JSON payload:


```
{
        "from_account_id": 10,
        "to_account_id": 12,
        "amount": 1000000000
      }
```
- Response Panel:** The response shows a 400 Bad Request error with the message:


```
{"success": false, "message": "Insufficient balance"}
```
- Bottom Status Bar:** Shows the current user (Postman), date (08-11-2025), and time (12:59 AM).

The screenshot shows the Postman interface with a collection named "jatin's Workspace". A specific POST request titled "Transfer Transaction" is selected. The request URL is `http://localhost:8082/transactions/transfer`. The request body is set to "raw" JSON:

```

1 {
2   "from_account_id": 11,
3   "to_account_id": 12,
4   "amount": 199999
5 }

```

The response status is 403 Forbidden, with a timestamp of 15 ms and a size of 435 B. The response body is:

```

1 {
2   "success": false,
3   "message": "Account frozen or inactive"
4 }

```

Minikube:

The screenshot shows the VS Code interface with several open files in the Explorer sidebar, including `prometheus-deployment.yaml`, `grafana-deployment.yaml`, `grafana-provisioning-configmap.yaml`, and `grafana-dashboard-configmap.yaml`. The main editor area displays a portion of the `grafana-dashboard-configmap.yaml` file, specifically the `transaction-red.json` section:

```

1
2   "data":
3     "transaction-red.json": [
4       {
5         "panels": [
6           ...
7         ],
8         "templating": {
9           "list": []
10        },
11        "time": {
12          "from": "now-15m",
13          "to": "now"
14        },
15        "timepicker": {
16          "refresh_intervals": ["5s", "10s", "30s", "1m", "5m"]
17        }
18      }
19    ]
20  }
21}

```

The bottom terminal window shows the output of Kubernetes commands:

```

PS C:\Users\Jatin\scalable-banking> kubectl apply -f k8s/monitoring/grafana-dashboard-configmap.yaml -n banking
PS C:\Users\Jatin\scalable-banking> kubectl rollout restart deployment/grafana -n banking
deployment.apps/grafana restarted
PS C:\Users\Jatin\scalable-banking> kubectl get pods -n banking --show-labels
>>
NAME          READY   STATUS    RESTARTS   AGE   LABELS
grafana-5dd577c77-7c44r   1/1    Running   0          115m app=grafana,pod-template-hash=5dd577c77
prometheus-76cdcd6cb-g9hrl   1/1    Running   0          146m app=prometheus,pod-template-hash=76cdcd6cb
rabbitmq-cc4587c68-6f9z1   1/1    Running   0          5h24m app=rabbitmq,pod-template-hash=c4587c68
transaction-db-7bb8c6775-mq9bl   1/1    Running   0          6h22m app=transaction-db,pod-template-hash=7bb8c6775
transaction-service-85f8c88598-plgcl   1/1    Running   0          4h40m app=transaction-service,pod-template-hash=85f8c88598

```

```

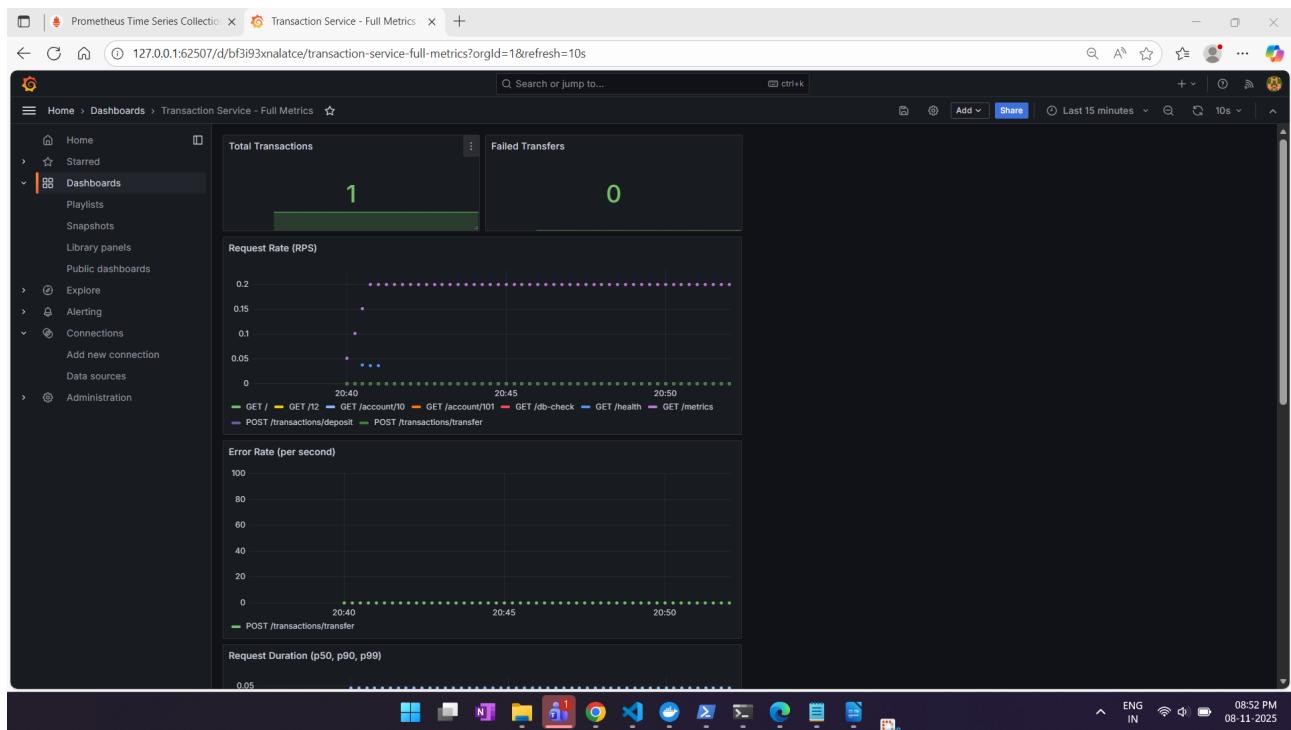
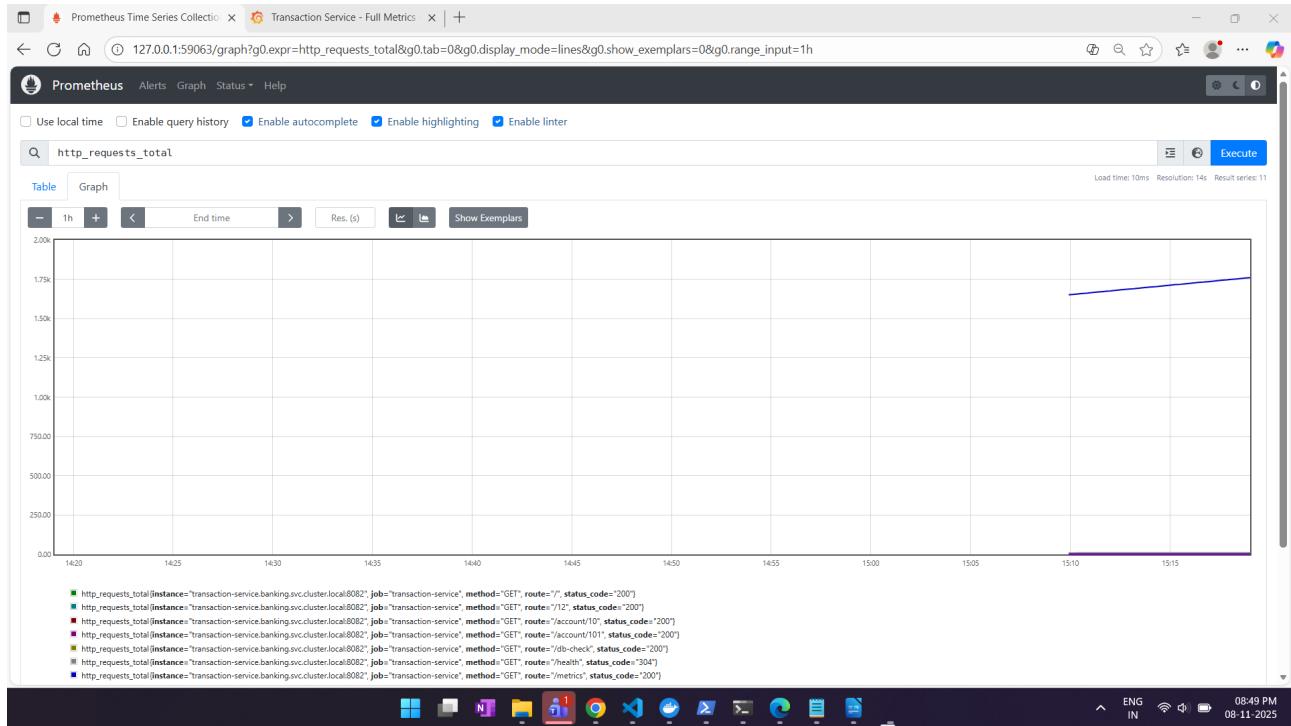
PS C:\Users\Jatin> minikube service rabbitmq -n banking
* Starting tunnel for service rabbitmq./
| NAMESPACE | NAME | TARGET PORT | URL |
| banking   | rabbitmq |             | http://127.0.0.1:49676
|           |         |             | http://127.0.0.1:49677 |
* Starting tunnel for service rabbitmq.
[banking rabbitmq http://127.0.0.1:49676
http://127.0.0.1:49677]
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
↳ Stopping tunnel for service rabbitmq.
PS C:\Users\Jatin> minikube service rabbitmq -n banking
| NAMESPACE | NAME | TARGET PORT | URL |
| banking   | rabbitmq | amqp/5672    | http://192.168.49.2:30672
|           |         | management/15672 | http://192.168.49.2:31672 |
* Starting tunnel for service rabbitmq./
| NAMESPACE | NAME | TARGET PORT | URL |
| banking   | rabbitmq |             | http://127.0.0.1:63719
|           |         |             | http://127.0.0.1:63720 |
* Starting tunnel for service rabbitmq.
[banking rabbitmq http://127.0.0.1:63719
http://127.0.0.1:63720]
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
|
```

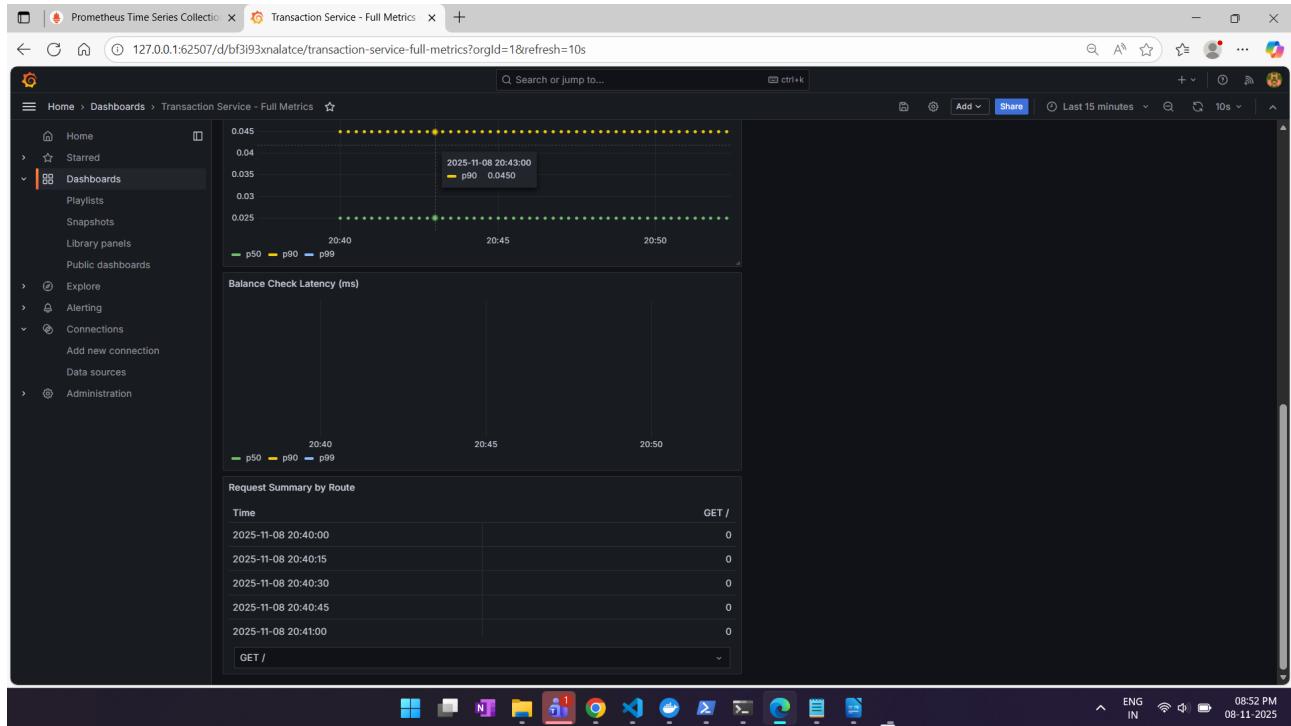
```

PS C:\Users\Jatin> minikube service grafana -n banking
| NAMESPACE | NAME | TARGET PORT | URL |
| banking   | grafana | http/3000   | http://192.168.49.2:30300 |
* Starting tunnel for service grafana./
| NAMESPACE | NAME | TARGET PORT | URL |
| banking   | grafana |             | http://127.0.0.1:55780 |
* Starting tunnel for service grafana.
🔗 Opening service banking/grafana in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
↳ Stopping tunnel for service grafana.
PS C:\Users\Jatin> minikube service grafana -n banking
| NAMESPACE | NAME | TARGET PORT | URL |
| banking   | grafana | http/3000   | http://192.168.49.2:30300 |
* Starting tunnel for service grafana./
| NAMESPACE | NAME | TARGET PORT | URL |
| banking   | grafana |             | http://127.0.0.1:62507 |
* Starting tunnel for service grafana.
🔗 Opening service banking/grafana in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
|
```

```
PS C:\Users\Jatin> minikube service prometheus -n banking
  NAMEPORT URL
banking prometheus web/9090 http://192.168.49.2:30900
* Starting tunnel for service prometheus./
  NAMEPORT URL
banking prometheus http://127.0.0.1:56200
* Starting tunnel for service prometheus.
* Opening service banking/prometheus in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
PS C:\Users\Jatin> minikube service prometheus -n banking
  NAMEPORT URL
banking prometheus web/9090 http://192.168.49.2:30900
* Starting tunnel for service prometheus./
  NAMEPORT URL
banking prometheus http://127.0.0.1:59063
* Starting tunnel for service prometheus.
* Opening service banking/prometheus in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://transaction-service.banking.svc.cluster.local:8082/metrics	UP	instance: "transaction-service.banking.svc.cluster.local:8082", job: "transaction-service"	1.497s ago	3.668ms	





```

http_request_duration_seconds_bucket{le="5",method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_bucket{le="10",method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_sum{method="POST",route="/transactions/transfer",status_code="200"} 0.01464601800003196
http_request_duration_seconds_count{method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_bucket{le="0.05",method="GET",route="/12",status_code="200"} 1
http_request_duration_seconds_bucket{le="0.1",method="GET",route="/12",status_code="200"} 1
http_request_duration_seconds_bucket{le="0.2",method="GET",route="/12",status_code="200"} 1
http_request_duration_seconds_bucket{le="0.5",method="GET",route="/12",status_code="200"} 1
http_request_duration_seconds_bucket{le="1",method="GET",route="/12",status_code="200"} 1
http_request_duration_seconds_bucket{le="2",method="GET",route="/12",status_code="200"} 1
http_request_duration_seconds_bucket{le="5",method="GET",route="/12",status_code="200"} 1
http_request_duration_seconds_bucket{le="10",method="GET",route="/12",status_code="200"} 1
http_request_duration_seconds_sum{method="GET",route="/12",status_code="200"} 0.023239715999379755
http_request_duration_seconds_count{method="GET",route="/12",status_code="200"} 1
http_request_duration_seconds_bucket{le="0.05",method="GET",route="/account/10",status_code="200"} 1
http_request_duration_seconds_bucket{le="0.1",method="GET",route="/account/10",status_code="200"} 1
http_request_duration_seconds_bucket{le="0.2",method="GET",route="/account/10",status_code="200"} 1
http_request_duration_seconds_bucket{le="0.5",method="GET",route="/account/10",status_code="200"} 1
http_request_duration_seconds_bucket{le="1",method="GET",route="/account/10",status_code="200"} 1
http_request_duration_seconds_bucket{le="2",method="GET",route="/account/10",status_code="200"} 1
http_request_duration_seconds_bucket{le="5",method="GET",route="/account/10",status_code="200"} 1
http_request_duration_seconds_bucket{le="10",method="GET",route="/account/10",status_code="200"} 1
http_request_duration_seconds_sum{method="GET",route="/account/10",status_code="200"} 0.0241125600039959
http_request_duration_seconds_count{method="GET",route="/account/10",status_code="200"} 1

# HELP transactions_total Total number of transactions processed
# TYPE transactions_total counter
transactions_total{tx_type="deposit"} 1

# HELP failed_transfers_total Total number of failed transfer attempts
# TYPE failed_transfers_total counter
failed_transfers_total 0

# HELP balance_check_latency_ms Latency of balance-check operations (ms)
# TYPE balance_check_latency_ms histogram
balance_check_latency_ms_bucket{le="5"} 0
balance_check_latency_ms_bucket{le="10"} 0
balance_check_latency_ms_bucket{le="20"} 3
balance_check_latency_ms_bucket{le="50"} 3
balance_check_latency_ms_bucket{le="100"} 3
balance_check_latency_ms_bucket{le="200"} 3
balance_check_latency_ms_bucket{le="500"} 3
balance_check_latency_ms_bucket{le="1000"} 4
balance_check_latency_ms_sum 2071.46007599995083
balance_check_latency_ms_count 4

```

Curl

```
curl -X 'GET' \
'http://127.0.0.1:57134/transactions/account/101' \
-H 'accept: application/json'
```

Request URL

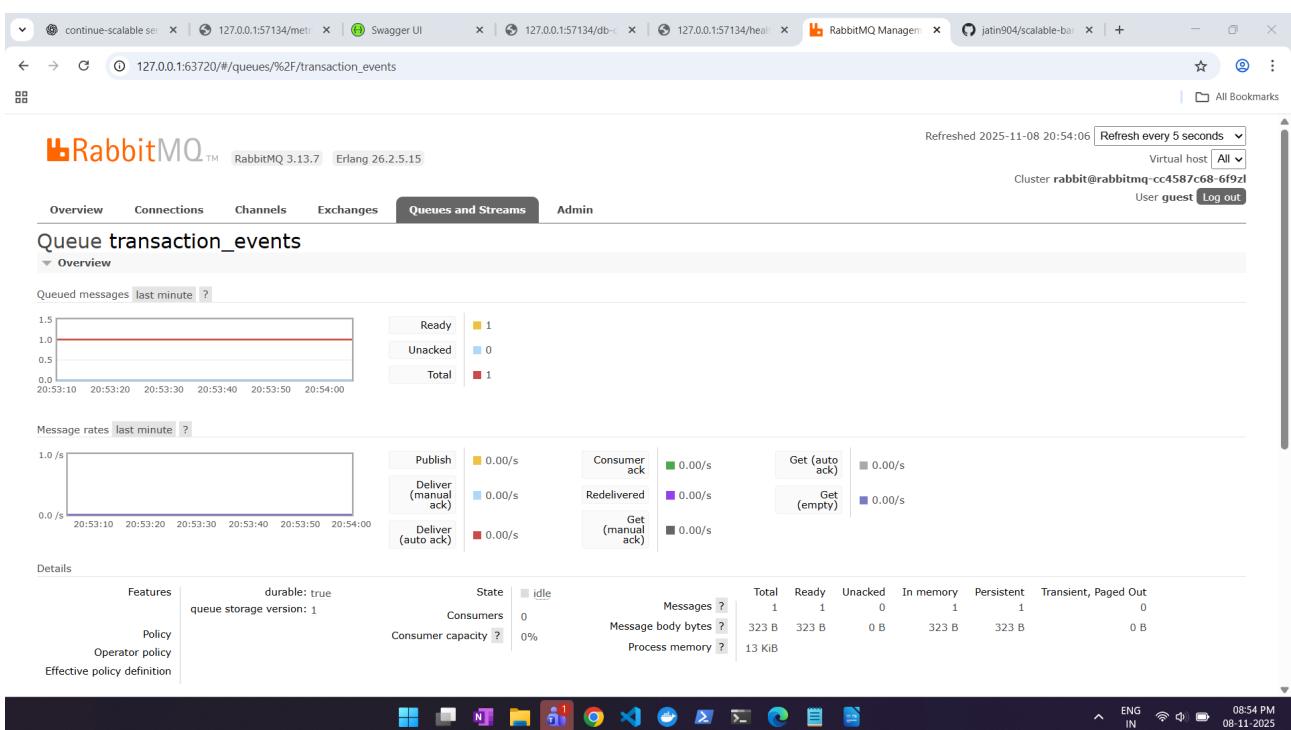
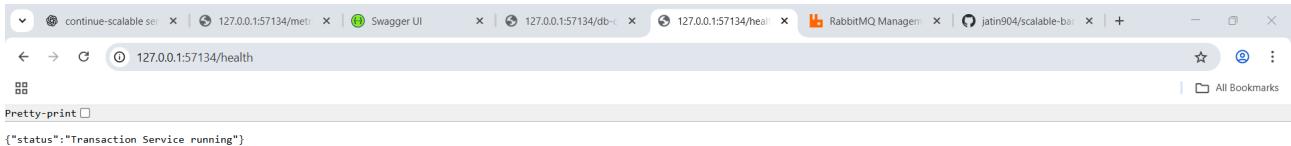
http://127.0.0.1:57134/transactions/account/101

Server response

Code	Details
200	<p>Response body</p> <pre>{ "success": true, "count": 1, "data": [{ "tx_id": "382", "account_id": "101", "amount": "500.00", "tx_type": "DEPOSIT", "counterparty": "SYSTEM:Deposit", "reference": "REF-1762592163377", "status": "SUCCESS", "created_at": "2023-11-08T08:56:03.377Z" }] }</pre> <p>Download</p>

Pretty-print

```
{"success":true,"db_time":"2025-11-08T13:22:17.480Z"}
```



The screenshot shows a browser window with multiple tabs open, all pointing to local host addresses. The central tab displays the RabbitMQ Management interface for a cluster named 'rabbit@rabbitmq-cc4587c68-6f921'. The user is logged in as 'guest'. The main page shows a single message in the 'transaction_events' queue. The message details are as follows:

- Exchange:** (AMQP default)
- Routing Key:** transaction_events
- Redelivered:** 0
- Properties:** delivery_mode: 2, headers: {{"type": "DEPOSIT_CREATED", "transaction": {"txnid": "303", "account_id": "10", "amount": "500.00", "txntype": "DEPOSIT", "counterparty": "SYSTEM:Deposit", "reference": "REF-1762598196443", "status": "SUCCESS", "created": "2025-11-08T08:54:26Z"}}}
- Payload:** 323 bytes, Encoding: string

Below the message details, there are buttons for 'Get Message(s)', 'Move messages', and 'Delete'. The system status bar at the bottom right indicates the time as 08:54 PM on 08-11-2025.

Outcome

Transaction service run consistently in isolated containers, with working inter-service connectivity and event streaming. The container setup satisfies:

- Independent deployment and scaling per service
- Portable environment for local and Minikube deployment
- Verified health, metrics, and logging endpoints