

# Table of Contents

Description.....	1
Responsibilities.....	1
Transaction Service – ER Diagram.....	2
Database: transaction_db contains:.....	2
Microservice Architecture Diagram.....	4
Patterns Used.....	4
API Endpoints.....	4
Business Rules & Validations.....	5
Containerization with Docker.....	5
Inter-Service Communication.....	5
Dockerfile Highlights.....	5
Docker-Compose Setup.....	6
Verification Steps:.....	6
Health Endpoints.....	6
Kubernetes Deployment (Minikube).....	6
Screenshots:.....	7
Started docker:.....	7
Docker Desktop:.....	7
Grafana dashboard:.....	8
Prometheus:.....	9
API:.....	10
RabbitMQ: transaction_events.....	16
Swagger.....	19
Business validation:.....	28
Minikube:.....	29
Outcome.....	30

## 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
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

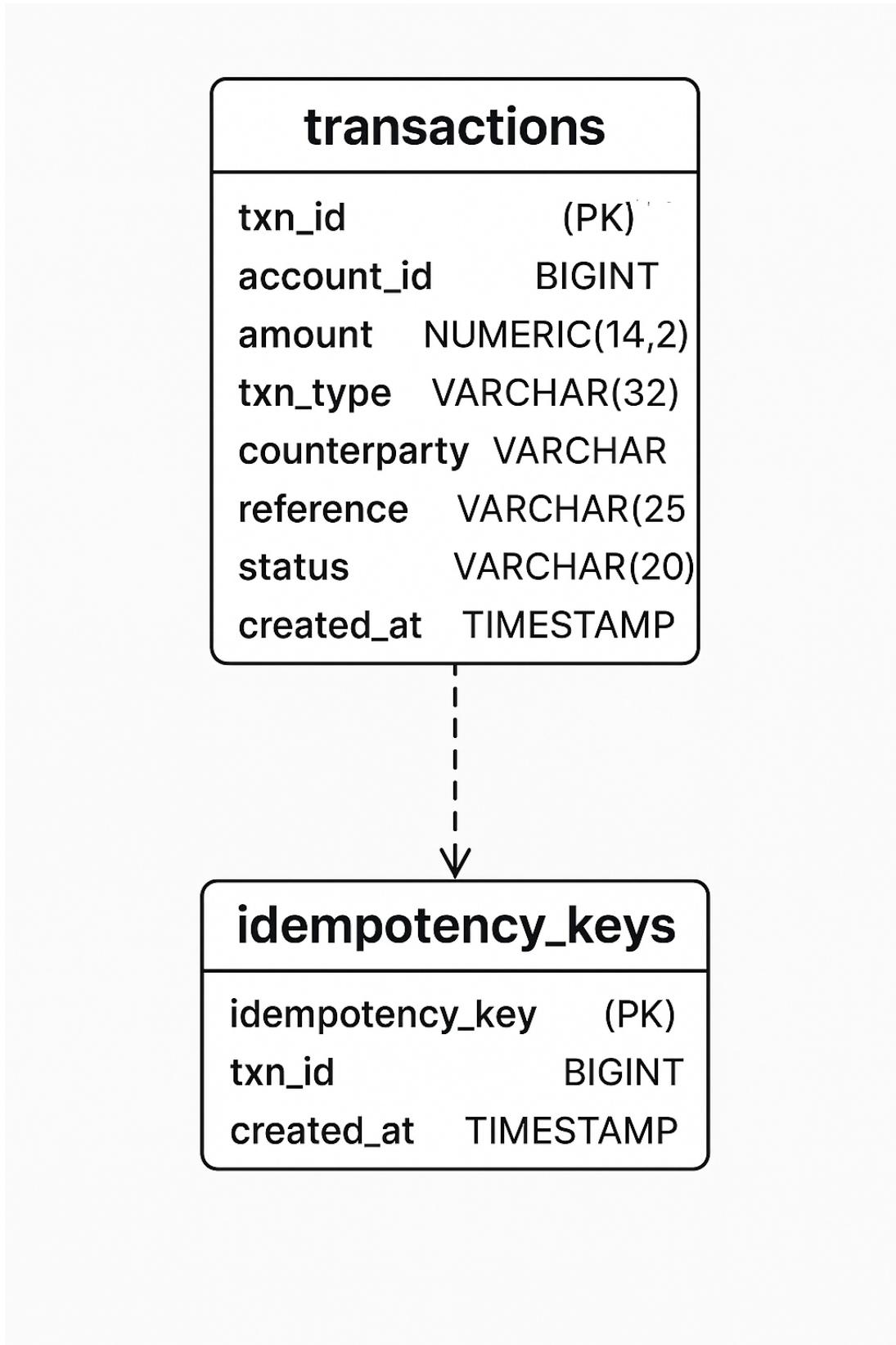
Log structured JSON with correlation IDs

### **Transaction Service – ER Diagram**

Logical ER structure based on my PostgreSQL schema:

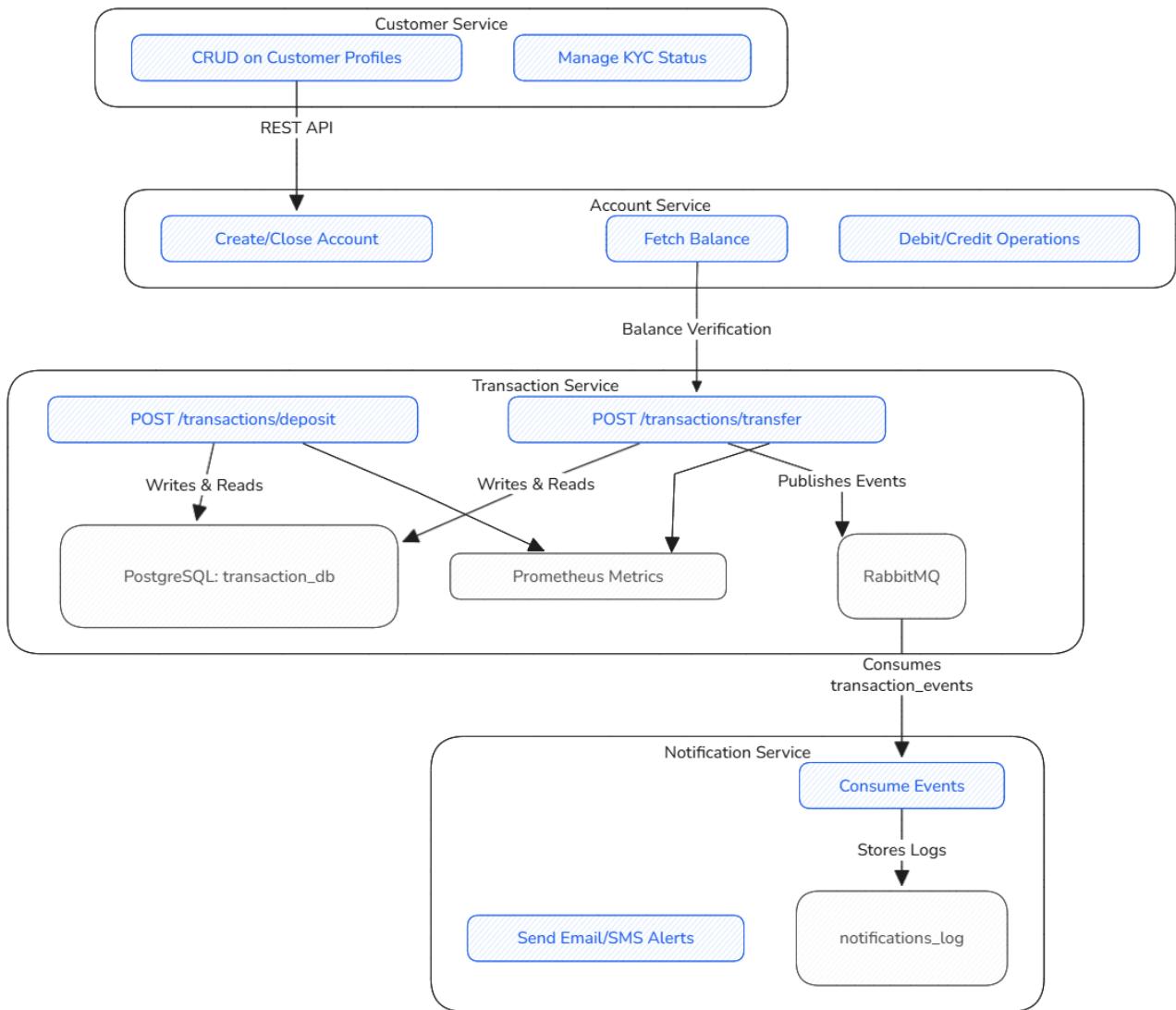
#### **Database: transaction\_db contains:**

- transactions (txn\_id, account\_id, amount, type, ref, created\_at)
- idempotency\_keys (key, txn\_id)



- 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

Method	Endpoint	Description
GET	/metrics	Prometheus metrics endpoint

## Business Rules & Validations

Rule	Description	Example
<b>Idempotency</b>	No duplicate processing if same key reused	Idempotency-Key: same-123
<b>Positive Amount</b>	Must be > 0	Reject 0 or negative
<b>Daily Transfer Limit</b>	₹200,000 per account per day	Reject if exceeded
<b>Account Status</b>	Must be ACTIVE	Reject frozen/inactive
<b>Sufficient Balance</b>	No overdraft	Reject if balance < amount
<b>Self-Transfer Prevention</b>	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

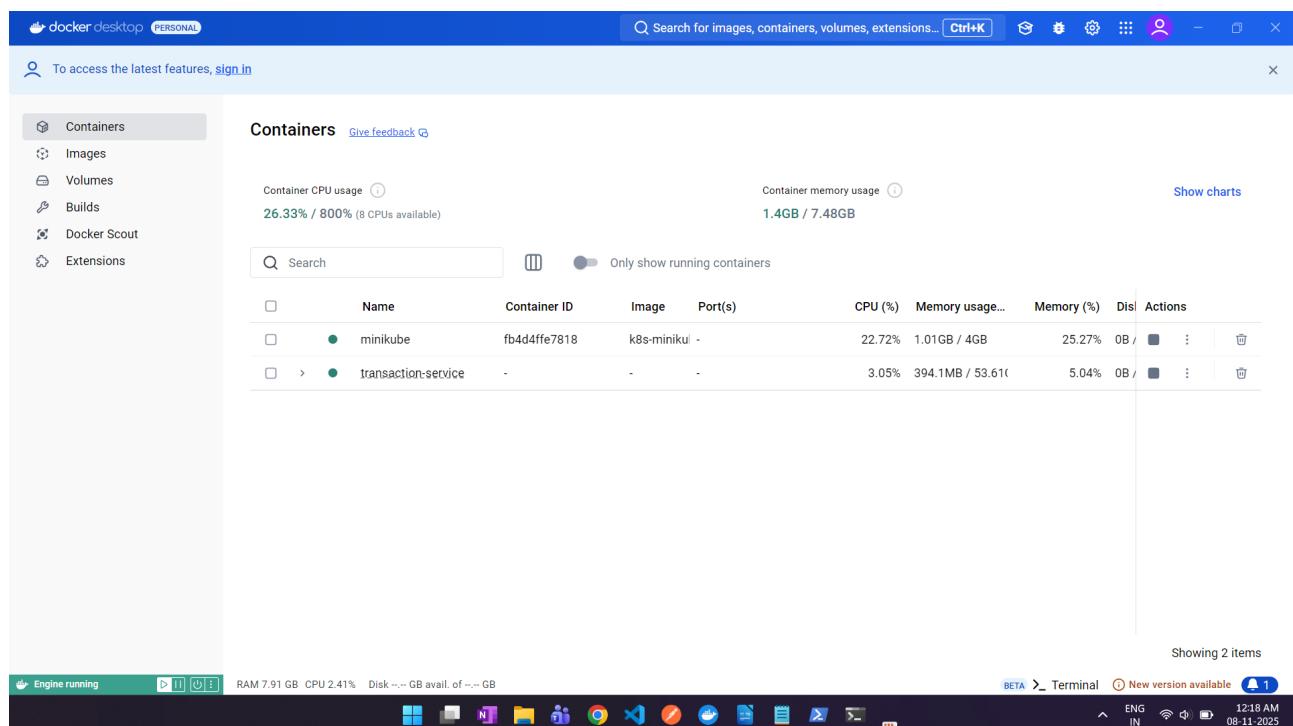
minikube service transaction-service -n banking –url

## Screenshots:

### Started docker:

The screenshot shows the VS Code interface with the Scalable-Banking project open. The Explorer sidebar shows files like `penapi.yaml`, `transaction-service`, `k8s`, `notification-service`, and `routes`. The `service.yaml` file is selected and displays the Kubernetes service configuration. The Terminal tab shows the command `PS C:\Users\Jatin\scalable-banking\transaction-service> docker compose up --build` being run, with logs for the `notification-service-1` container. The logs include messages about RabbitMQ and Prometheus metrics. The status bar at the bottom indicates the terminal has 14 columns, 2 spaces, and is in UTF-8 mode.

### Docker Desktop:



All inner service are running fine inside transaction-service container

Docker Desktop interface showing the 'Containers' section. It displays various services running within a Kubernetes cluster (minikube). Key metrics shown include Container CPU usage (22.46% / 800% available) and Container memory usage (1.38GB / 7.48GB). A search bar and a filter for 'Only show running containers' are visible. A 'Show charts' button is present in the top right.

Name	Container ID	Image	Port(s)	CPU (%)	Memory usage...	Memory (%)	Disl	Actions
minikube	fb4d4ffe7818	k8s-minikui	-	18.96%	1.01GB / 4GB	25.29%	OB /	⋮
transaction-service	-	-	-	3.37%	378.96MB / 53.6	4.84%	OB /	⋮
rabbitmq-1	047511e2d6b8	rabbitmq:3	15672:15672	1.39%	141.8MB / 7.66Gi	1.81%	OB /	⋮
transaction-db-1	0f8b9328e0e1	postgres:1t	5434:5432	0%	28.27MB / 7.66Gi	0.36%	OB /	⋮
account-service-1	814292d9be07	transaction	8083:8083	0%	29.18MB / 7.66Gi	0.37%	OB /	⋮
notification-service-1	cc3c1c01ff136	transaction	8084:8084	0.01%	22.39MB / 7.66Gi	0.29%	OB /	⋮
transaction-service-1	bfc9a3bf876a	transaction	8082:8082	1.3%	42.46MB / 7.66Gi	0.54%	OB /	⋮
prometheus	0bb823bcc20e	prom/prom	9090:9090	0.13%	25.62MB / 7.66Gi	0.33%	OB /	⋮
grafana	24ad10b42108	grafana/graf	3000:3000	0.54%	89.24MB / 7.66Gi	1.14%	OB /	⋮

Showing 9 items

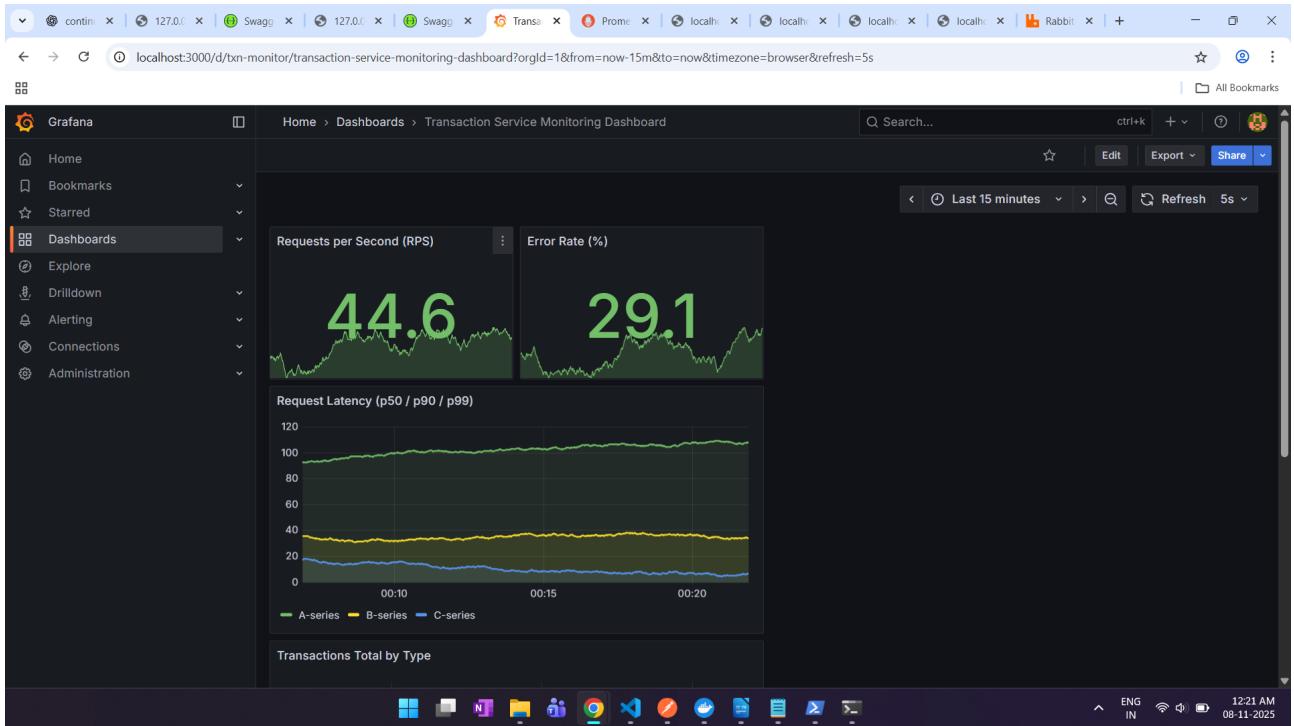
Engine running RAM 7.90 GB CPU 5.64% Disk -- GB avail. of -- GB

BETA Terminal New version available ENG IN 12:18 AM 08-11-2025

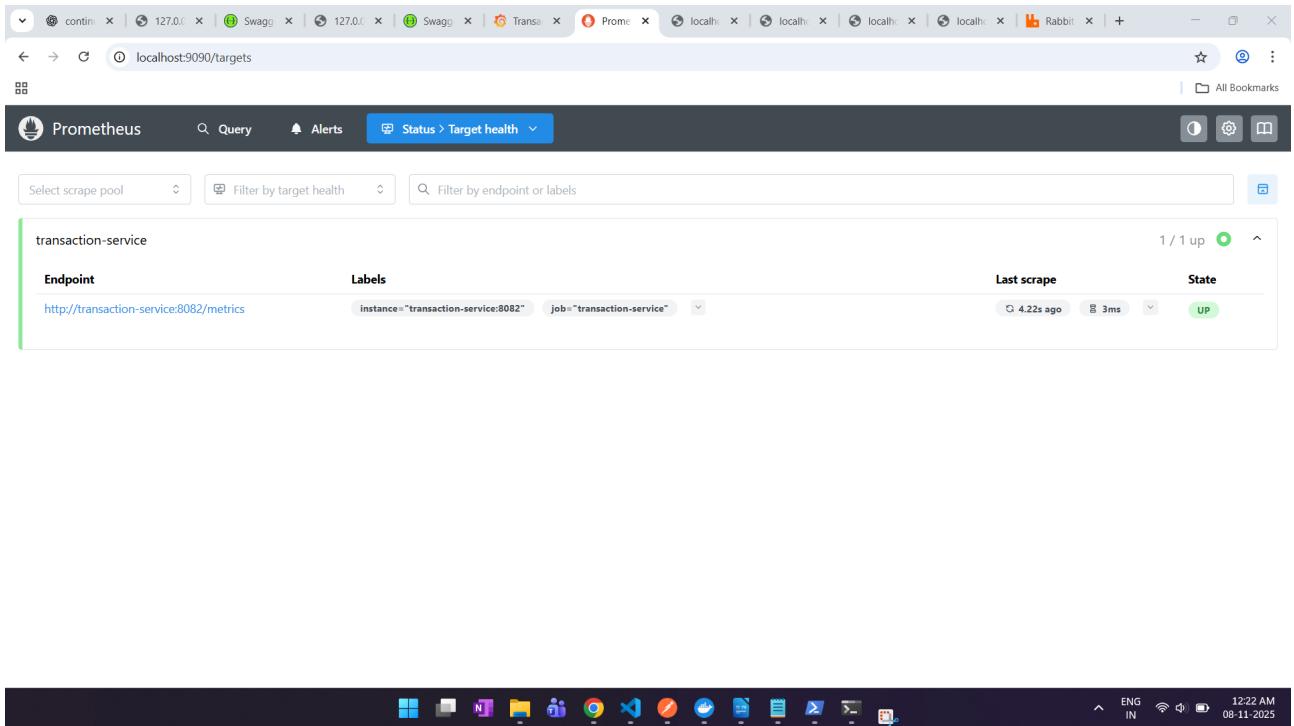
## Grafana dashboard:

for visual metrics of transaction service

Grafana dashboard titled 'Transaction Service Monitoring Dashboard'. The left sidebar shows navigation options like Home, Dashboards, Explore, and Alerting. The main area contains three panels: 1) 'Transactions Total by Type' showing a line chart for the 'A-series' (green line) over a 15-minute period, with values fluctuating between 2 and 12. 2) 'Failed Transfers (Total)' showing a large value of 27.1 in red text on a green background. 3) 'Balance Check Latency (ms)' showing a line chart for the 'A-series' (green line) over a 20-minute period, with values fluctuating between 92 and 96 ms. The bottom status bar shows system information: Engine running, RAM 7.90 GB, CPU 5.64%, Disk -- GB avail. of -- GB, BETA, Terminal, New version available, ENG IN, 12:20 AM, 08-11-2025.



## Prometheus:



Metric in Prometheus:

The screenshot shows the Prometheus web interface. At the top, there are tabs for 'localhost:9090/tsdb-status' and other monitoring services like Grafana, Metrics, and RabbitMQ. The main area displays the 'Status > TSDB status' section, which includes a table with three rows: 'space' (Count: 450), 'kind' (Count: 351), and 'type' (Count: 152). Below this is a section titled 'Top 10 series count by label value pairs', which lists ten entries with their respective counts:

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>

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'jatin's Workspace' containing collections, environments, flows, and history. The main area shows a request for 'http://localhost:8082/transactions/deposit' using the 'POST' method. The 'Headers' tab is selected, showing two headers: 'Content-Type: application/json' and 'Idempotency-Key: test-deposit-002'. The 'Body' tab shows a JSON response with the following content:

```

1  {
2      "success": true,
3      "transaction": {
4          "tx_id": "314",
5          "account_id": "101",
6          "amount": "100.00",
7          "tx_type": "DEPOSIT",
8          "counterparty": "SYSTEM:Deposit",
9          "reference": "REF_1762524148184",
10         "status": "SUCCESS",
11         "created_at": "2025-11-07T14:02:28.184Z"
12     }

```

The status bar at the bottom indicates a '201 Created' response with a duration of 62 ms and a size of 525 B.

The screenshot shows the Postman interface with a collection named "Scalable Banking - Transaction Microservice...". A POST request is made to `http://localhost:8082/transactions/deposit`. The request body is a JSON object:

```

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

```

The response is a `201 Created` status with a JSON payload:

```

1  {
2     "success": true,
3     "transaction": {
4         "txn_id": "314",
5         "account_id": "101",
6         "amount": "100.00",
7         "txn_type": "DEPOSIT",
8         "counterparty": "SYSTEM:Deposit",
9         "reference": "REF-1762524148184",
10        "status": "SUCCESS",
11        "created_at": "2025-11-07T14:02:28.184Z"
12    }
13 }

```

{

```

"success": true,
"transaction": {
    "txn_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

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

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

Body Cookies Headers (10) Test Results

```
{
  "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"
  }
}
```

200 OK 86 ms 615 B

## POST http://localhost:8082/transactions/transfer

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

Key	Value	Description	Bulk Edit	Presets
Content-Type	application/json			
Idempotency-Key	test-transfer-002			

Body Cookies Headers (10) Test Results

```
{
  "success": true,
  "sender_transaction": {
    "tx_id": "315",
    "account_id": "10",
    "amount": "200.00",
    "tx_type": "TRANSFER_OUT",
    "counterparty": "TO:12",
    "reference": "REF-OUT-1762541913324",
    "status": "SUCCESS",
    "created_at": "2025-11-07T18:50:33.324Z"
  },
  "receiver_transaction": {
    "tx_id": "316",
    "account_id": "12"
  }
}
```

201 Created 130 ms 835 B

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

```

1 {
2   "from_account_id": 10,
3   "to_account_id": 12,
4   "amount": 2002
5 }

```

The response status is 201 Created, with a response time of 130 ms and a size of 835 B. The response body is displayed as JSON:

```

{
  "success": true,
  "sender_transaction": {
    "txnid": "315",
    "account_id": "10",
    "amount": "2002.00",
    "txntype": "TRANSFER_OUT",
    "counterparty": "TO:12",
    "reference": "REF-OUT-1762541913324",
    "status": "SUCCESS",
    "created_at": "2025-11-07T18:58:33.324Z"
  },
  "receiver_transaction": {
    "txnid": "316",
    "account_id": "12"
  }
}

```

```

{
  "success": true,
  "sender_transaction": {
    "txnid": "315",
    "account_id": "10",
    "amount": "2002.00",
    "txntype": "TRANSFER_OUT",
    "counterparty": "TO:12",
    "reference": "REF-OUT-1762541913324",
    "status": "SUCCESS",
    "created_at": "2025-11-07T18:58:33.324Z"
  },
  "receiver_transaction": {
    "txnid": "316",
    "account_id": "12",
    "amount": "2002.00",
    "txntype": "TRANSFER_IN",
    "counterparty": "FROM:10",
    "reference": "REF-IN-176254191332",
    "status": "SUCCESS",
  }
}

```

```
"created_at": "2025-11-07T18:58:33.324Z"
```

```
}
```

```
}
```

If same request is send again, getting reused:true

The screenshot shows a Postman collection named 'Scalable Banking - Transaction Microservice...' containing several API endpoints. The 'Transfer Transaction' endpoint is selected. A POST request is made to `http://localhost:8082/transactions/transfer`. The Headers tab shows 'Content-Type: application/json' and 'Idempotency-Key: test-transfer-002'. The response status is 200 OK, and the response body is:

```
{
  "success": true,
  "transaction": {
    "tx_id": "315",
    "account_id": "10",
    "amount": "2802.00",
    "tx_type": "TRANSFER_OUT",
    "counterparty": "TO:12",
    "reference": "REF-OUT-1762541913324",
    "status": "SUCCESS",
    "created_at": "2025-11-07T18:58:33.324Z"
  },
  "reused": true
}
```

Get `http://localhost:8082/transactions/16`

The screenshot shows the same Postman collection and environment. A GET request is made to `http://localhost:8082/transactions/16`. The Headers tab shows 'Content-Type: application/json'. The response status is 200 OK, and the response body is:

```
{
  "success": true,
  "transaction": {
    "tx_id": "16",
    "account_id": "21",
    "amount": "133.53",
    "tx_type": "DEPOSIT",
    "counterparty": "IMPS:External",
    "reference": "REF20250827-30QW50",
    "status": "SUCCESS",
    "created_at": "2024-11-21T10:07:47.000Z"
  }
}
```

```
{
  "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>

The screenshot shows the Postman interface with a successful API call to `http://localhost:8082/metrics`. The response body is a Prometheus-style metrics output:

```

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="+Inf"} 3
295 balance_check_latency_ms_sum 58.5078869999852
296 balance_check_latency_ms_count 3
297

```

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

The screenshot shows the Postman interface with a collection named "jatin's Workspace". A specific test case for "Scalable Banking - Transaction Microservice (Full Test Suite) / Account Balance - Account 10" is selected. The request method is GET, and the URL is http://localhost:8083/accounts/10/balance. The response is a 200 OK status with a JSON payload:

```

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

```

## RabbitMQ: transaction\_events

The screenshot shows the RabbitMQ Management UI at the URL localhost:15672/#/queues. The "Queues" tab is selected. There is one queue listed:

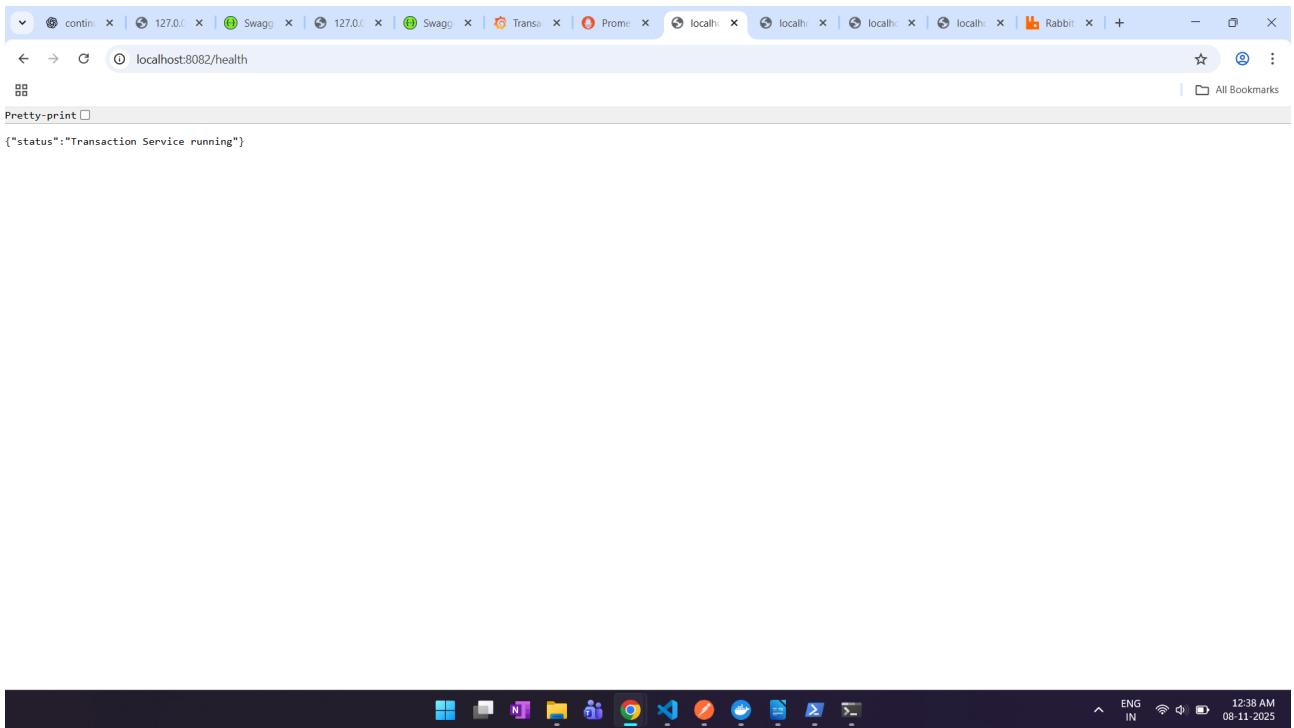
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/	transaction_events	classic	D	running	0	0	0	0.00/s	0.00/s	0.00/s

At the bottom of the page, there are links for "HTTP API", "Documentation", "Tutorials", "New releases", "Commercial edition", "Commercial support", "Discussions", "Discord", "Plugins", and "GitHub".

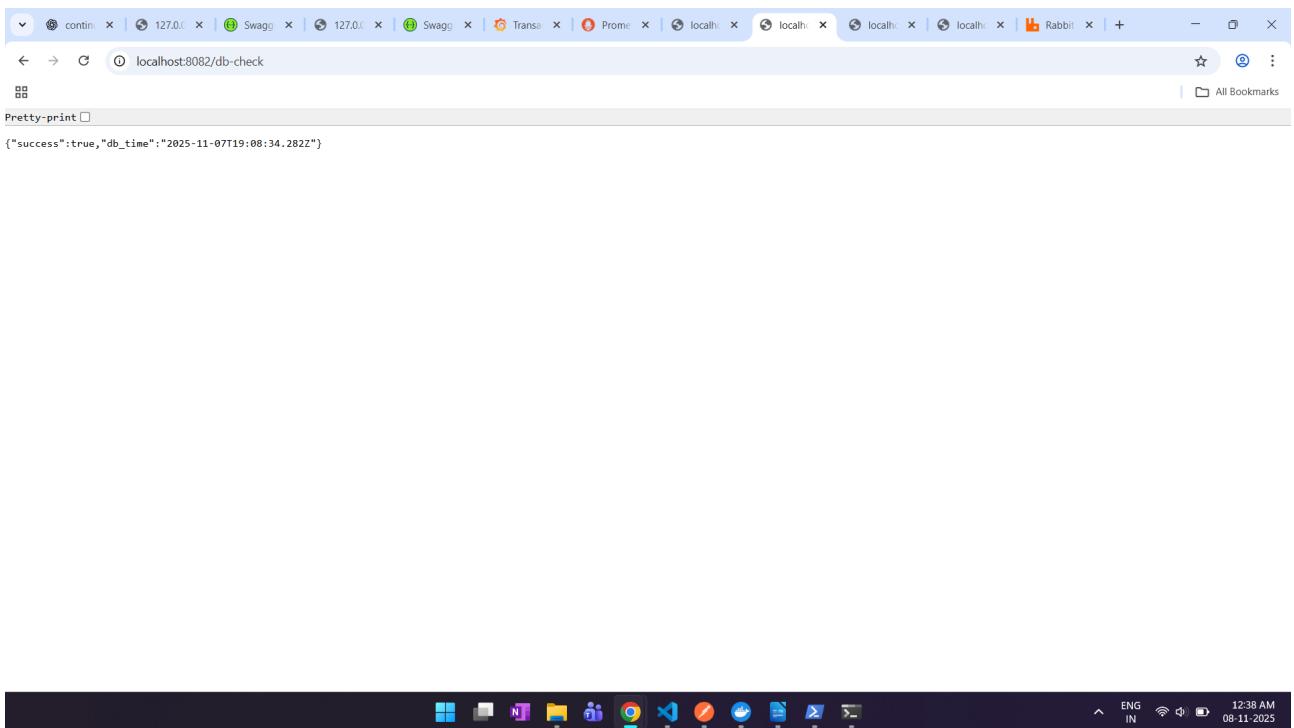
Screenshot of the RabbitMQ Management Console showing the Queue transaction\_events overview. The interface includes a navigation bar with Overview, Connections, Channels, Exchanges, Queues and Streams (selected), and Admin. The main area displays metrics for queued messages and message rates over the last minute. A detailed table provides queue statistics like durable, storage version, state, consumers, and capacity. The bottom status bar shows system information like battery level and date.

Screenshot of the RabbitMQ Management Console showing the Queue transaction\_events overview. The interface includes a navigation bar with Overview, Connections, Channels, Exchanges, Queues and Streams (selected), and Admin. The main area displays metrics for queued messages and message rates over the last minute. A detailed table provides queue statistics like durable, storage version, state, consumers, and capacity. The bottom status bar shows system information like battery level and date.

Transaction-service health check:



#### Transaction database health check:



#### 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="20",method="POST",route="/transactions/transfer",status_code="201"} 0.1985457300010603
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="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="20",method="GET",route="/16",status_code="200"} 2
http_request_duration_seconds_count{method="POST",route="/transactions/transfer",status_code="200"} 0.01837367700005416
http_request_duration_seconds_count{method="POST",route="/transactions/transfer",status_code="200"} 1
# 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 '/metrics' endpoint. The interface includes sections for 'Parameters' (none listed), 'Responses' (including a 'Curl' command and a 'Request URL'), and a large 'Response body' section containing Prometheus metrics output. 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. The interface includes sections for 'Parameters' (with a 'account\_id' parameter set to '10'), 'Responses' (including a 'Curl' command and a 'Request URL'), and a large 'Response body' section containing transaction data. 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	<p>Response body</p> <pre>{   "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",       "amount": "2002.00",       "tx_type": "TRANSFER_OUT",       "counterparty": "TO:12",       "reference": "REF-OUT-1762512250574",       "status": "SUCCESS",       "created_at": "2025-11-07T10:44:10.573Z"     }   ] }</pre> <p>Download</p>

Response headers

account\_id Filter transactions by account ID

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	<p>Response body</p> <pre>{   "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",       "amount": "2002.00",       "tx_type": "TRANSFER_OUT",       "counterparty": "TO:12",       "reference": "REF-OUT-1762512250574",       "status": "SUCCESS",       "created_at": "2025-11-07T10:44:10.573Z"     }   ] }</pre>

**GET /transactions/account/{accountId}** Get transactions for a specific account

**Parameters**

Name	Description
accountId * required	Account ID to fetch transactions for
integer (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-ab32-42de-bcbf-19f3d431b812",
      "type": "transfer",
      "amount": 100.00,
      "counterparty": "SYSTEM:Deposit",
      "reference": "REF-1762542603692"
    }
  ]
}
```

**accountID \* required** Account ID to fetch transactions for

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"
    }
  ]
}
```

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	<b>Response body</b> <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": "PENDING",       "created_at": "2025-11-07T14:02:28.184Z"     }   ] }</pre> <p><b>Download</b></p>

Response headers

12:46 AM 08-11-2025

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

Parameters

Name	Description
<b>id</b> * required	string (path)
10	

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	<b>Response body</b> <pre>{   "tx_id": "311",   "account_id": "101",   "amount": "100.00",   "tx_type": "DEPOSIT",   "counterparty": "SYSTEM:Deposit",   "reference": "REF-1762524148184",   "status": "PENDING",   "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	<p>Response body</p> <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>Response headers</p> <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-Y3G154xnbgEyK6y+7Kh2MBTk8" keep-alive: timeout=5 x-correlation-id: d998e59a-02fe-4ee3-a93e-b266f5f790c x-powered-by: Express</pre>

Download

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.

System tray: ENG IN, 12:47 AM, 08-11-2025

POST /transactions/transfer Create a transfer (idempotent)

Parameters

Name	Description
<b>Idempotency-Key</b> <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.

System tray: ENG IN, 12:48 AM, 08-11-2025

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": 101,
    "to_account_id": 102,
    "amount": 250.5
}'
```

Request URL

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

Server response

Code	Details
503 Undocumented	Error: Service Unavailable

Response body

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

Response 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
```

Responses

POST /transactions/transfer Create a transfer (idempotent)

Cancel Reset

Parameters

Name	Description
Idempotency-Key <small>required</small>	Unique key to ensure idempotent request handling. <input type="text" value="test-transfer-003"/>
X-Correlation-ID	Correlation ID for traceability. <input type="text" value="X-Correlation-ID"/>

Request body required

application/json

Edit Value Schema

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

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	<p>Response body</p> <pre>{   "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"   } }</pre> <p><a href="#">Copy</a> <a href="#">Download</a></p>



ENG IN 12:51 AM 08-11-2025

Curl

```
{
  "from_account_id": 10,
  "to_account_id": 12,
  "amount": 2500000000
}
```

Execute Clear

Responses

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



ENG IN 12:53 AM 08-11-2025

Github:<https://github.com/jatin904/scalable-banking.git>

The screenshot shows a browser window with multiple tabs open, including 'contin...', '127.0.0.1 Swagg...', '127.0.0.1 Swagg...', 'Transa...', 'Prom...', 'localhost...', 'localhost...', 'localhost...', 'localhost...', 'Rabbit...', and '+'. The main content area is focused on the 'localhost:8082/api-docs/#/default/createTransfer' endpoint.

**Code Examples:**

```
        "created_at": "2025-11-07T10:00:00Z",
    },
    "reused": true
}

Response headers:
```

access-control-allow-origin: \*  
access-control-expose-headers: X-Correlation-ID  
connection: keep-alive  
content-length: 113  
content-type: application/json; charset=utf-8  
date: Fri, 07 Nov 2025 19:22:54 GMT  
etag: W/"f1-t2pbKwvfatmlMuuqTr3y1ldrZQ"  
keep-alive: timeout=5  
x-correlation-id: acdf7afe-6d4a-4147-b846-e50985d4bcfe  
x-powered-by: Express

**Responses**

Code	Description	Links
201	Transfer created successfully	No links
	Media type	
	application/json	
	Controls Accept header	
	Example Value   Schema	
	{ "id": "7e0db25-ab32-42de-bcbd-19f3d431b812", "type": "transfer", "from_account_id": 101, "to_account_id": 202, "amount": 550.5, "status": "completed", "created_at": "2025-11-06T10:00:00Z" }	
400	Bad request or business rule violation	No links
409	Duplicate request (idempotency conflict)	No links

At the bottom of the screen, there is a taskbar with various icons for system functions like search, file operations, and connectivity. The status bar at the bottom right shows the date and time as '08-11-2025 12:53 AM'.

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

```
contin 127.0.0.1 Swagg 127.0.0.1 Swagg Trans Prometheus localhost localhost localhost Rabbit + All Bookmarks  
localhost:8082/metrics  
  
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"} 0.0170815500004068  
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.sum{method="POST",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.bucket{le="0.05",method="PUT",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.bucket{le="0.1",method="PUT",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.sum{method="PUT",route="/transactions/transfer",status_code="503"} 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.sum{method="POST",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.bucket{le="0.05",method="PUT",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.bucket{le="0.1",method="PUT",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.sum{method="PUT",route="/transactions/transfer",status_code="503"} 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.sum{method="POST",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.bucket{le="0.05",method="PUT",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.bucket{le="0.1",method="PUT",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.sum{method="PUT",route="/transactions/transfer",status_code="503"} 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.sum{method="POST",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.bucket{le="0.05",method="PUT",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.bucket{le="0.1",method="PUT",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.sum{method="PUT",route="/transactions/transfer",status_code="503"} 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.sum{method="POST",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.bucket{le="0.05",method="PUT",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.bucket{le="0.1",method="PUT",route="/transactions/transfer",status_code="503"} 1  
http.request.duration.seconds.sum{method="PUT",route="/transactions/transfer",status_code="503"} 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.sum{method="POST",route="/transactions/transfer",status_code="503"} 1  
# 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  
# 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  
  
12:55 AM  
08-Nov-2023
```

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 log entries from the transaction-service container.
- Bottom Status Bar:** Shows the date and time (08-11-2025, 12:57 AM), system status (ENG IN), and file tabs for main, Dockerfile, and docker-compose.yml.

## Business validation:

The screenshot shows the Postman interface with the following details:

- Sidebar:** Shows "jatin's Workspace" with collections like "Scalable Banking - Transaction Microservice..." containing various API endpoints.
- Request Panel:** A POST request to "http://localhost:8082/transactions/transfer" with the following body:

```
1 {
2   "from_account_id": 10,
3   "to_account_id": 12,
4   "amount": 1000000000
5 }
```

- Response Panel:** Shows a 400 Bad Request response with the following JSON body:

```
1 {
2   "success": false,
3   "message": "Insufficient balance"
4 }
```

- Bottom Status Bar:** Shows the date and time (08-11-2025, 12:59 AM), system status (ENG IN), and file tabs for Postman, Dockerfile, and docker-compose.yml.

The screenshot shows the Postman interface. On the left, there's a sidebar with collections, environments, flows, and history. The main area displays a test suite for 'Scalable Banking - Transaction Microservice'. A specific test case for 'Transfer Transaction' is selected, showing a POST request to '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 duration of '15 ms' and a size of '435 B'. The response body is:

```

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

```

At the bottom, there are tabs for Body, Cookies, Headers (10), Test Results, and a timestamp of '08-11-2025 01:00 AM'.

## Minikube:

```

Administrator: Windows PowerShell
#0 building with "default" instance using docker driver
#1 [internal] load build definition from Dockerfile
#1 transferring dockerfile: 575B done
#1 DONE 0.0s
#2 [internal] load metadata for docker.io/library/node:20-alpine
#2 DONE 2.2s
#3 [internal] load .dockerignore
#3 transferring context: 2B done
#3 DONE 0.0s
#4 [1/5] FROM docker.io/library/node:20-alpine@sha256:6178e78b972f79c335df281f4b7674a2d85071aae2af020ffa39f0a770265435
#4 DONE 0.0s
#5 [internal] load build context
#5 transferring context: 26.41MB 1.3s done
#5 DONE 1.3s
#6 [2/5] WORKDIR /app
#6 CACHED
#7 [3/5] COPY package*.json .
#7 CACHED
#8 [4/5] RUN npm install
#8 CACHED
#9 [5/5] COPY .
#9 DONE 0.8s
#10 exporting to image
#10 exporting layers 0.3s done
#10 writing image sha256:63d417da092003b85abf8290477c324ff870ab166408b6a730ccc494ea5b27b2 done
#10 naming to docker.io/library/transaction-service:latest done
#10 DONE 0.3s
PS C:\Users\jatin\scalable-banking\transaction-service> kubectl rollout restart deployment transaction-service -n banking
deployment.apps/transaction-service restarted
PS C:\Users\jatin\scalable-banking\transaction-service> kubectl get pods -n banking
>>
NAME          READY   STATUS    RESTARTS   AGE
rabbitmq-7766b749dd-bljfs   1/1   Running   1 (12m ago)  137m
transaction-db-6fb87977d-j5v8m  1/1   Running   1 (12m ago)  139m
transaction-service-7bd59b4bdc-2vllv 1/1   Running   0           4s
PS C:\Users\jatin\scalable-banking\transaction-service>

```

```
C:\Users\Jatin>minikube service transaction-service -n banking
| NAMESPACE | NAME | TARGET PORT | URL |
| banking   | transaction-service | http/8082 | http://192.168.49.2:30082 |
* Starting tunnel for service transaction-service./
| NAMESPACE | NAME | TARGET PORT | URL |
| banking   | transaction-service |          | http://127.0.0.1:57581 |
* Starting tunnel for service transaction-service.
* Opening service banking/transaction-service in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

## 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