

TRANSACTION SERVICE**Developed by-** Jatin Saini2024tm93002@wilp.bits-pilani.ac.inRepository Link: <https://github.com/jatin904/scalable-banking.git>[Video Demo](#)

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:.....	26
Outcome.....	32

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

Component	Technology
Containerization	Docker
Orchestration	Kubernetes (Minikube)
Docs	Swagger / OpenAPI

Responsibilities

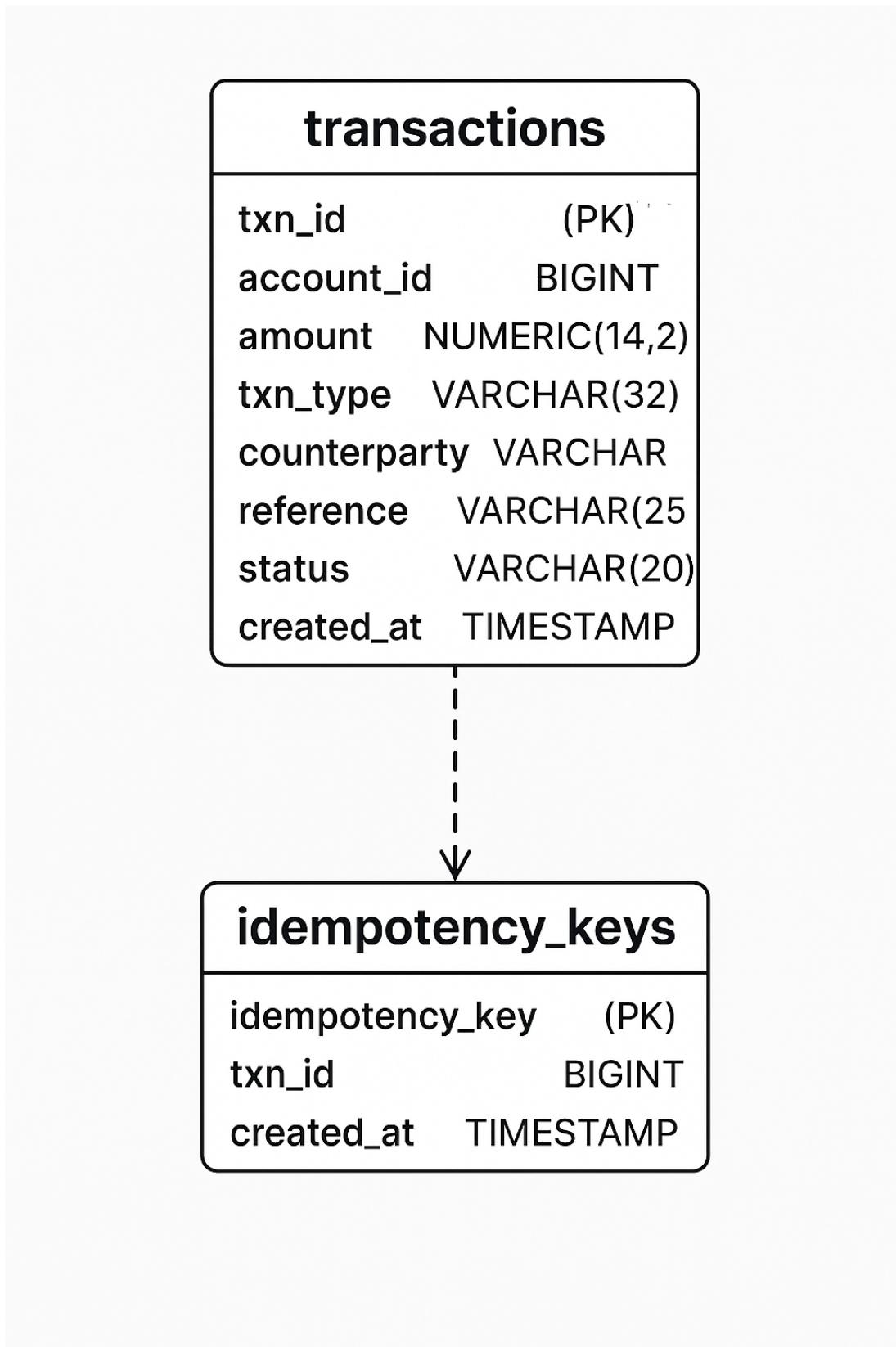
- Record and validate Deposit and Transfer transactions
- Enforce business rules (daily limits, balance, account status)
- Maintain idempotency for safe retries
- Publish transaction events (DEPOSIT_CREATED, TRANSFER_COMPLETED)
- Expose REST APIs + /metrics endpoint for Prometheus
- Log structured JSON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

Transaction Service – ER Diagram

Logical ER structure based on my PostgreSQL schema:

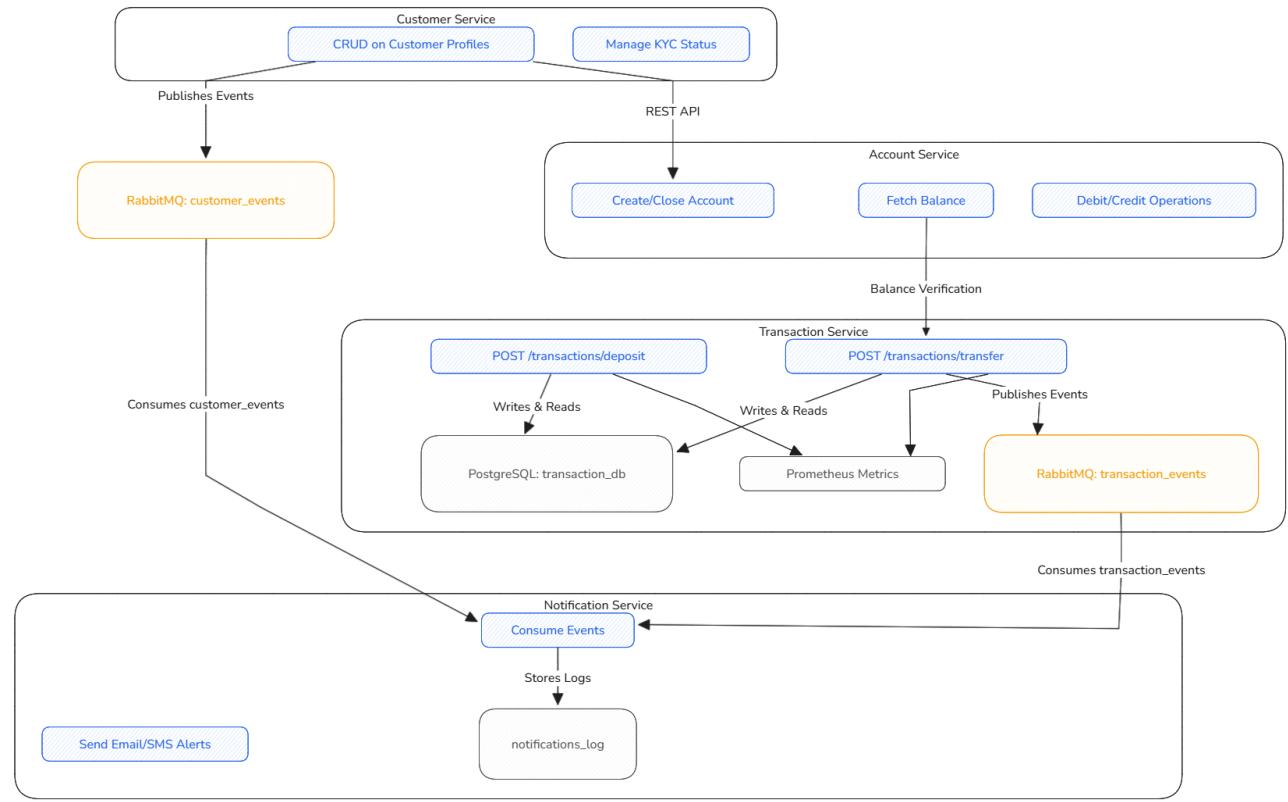
Database: transaction_db contains:

- transactions (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:

```

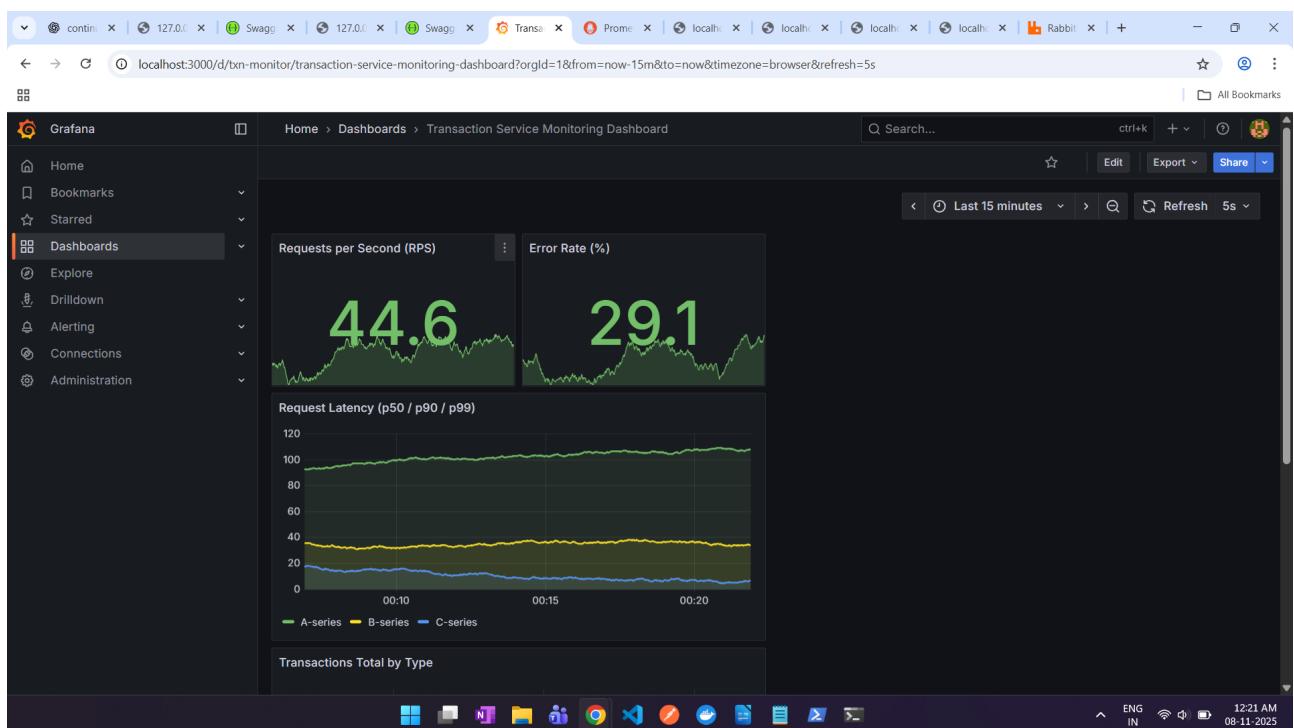
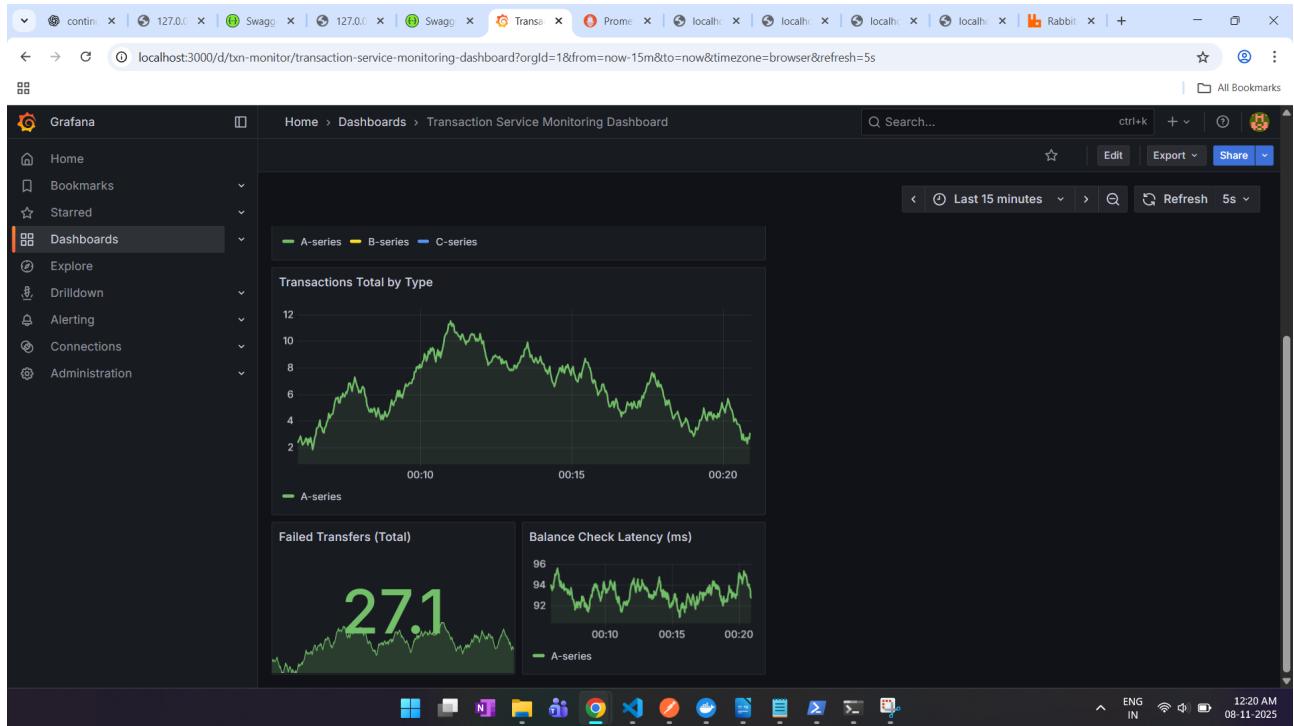
apiVersion: v1
kind: Service
metadata:
  name: transaction-service
  namespace: banking
spec:
  type: NodePort
  selector:
    app: transaction-service
  ports:
    - name: http
      port: 8082
      targetPort: 8082
      nodePort: 30082
  
```

Docker Desktop:

Name	Container ID	Image	Port(s)	CPU (%)	Memory usage...	Memory (%)	Actions
minikube	fb4d4ffe7818	k8s-minikube	-	18.96%	1.01GB / 4GB	25.29%	⋮
transaction-service	-	-	-	3.37%	378.96MB / 53.6Gi	4.84%	⋮
rabbitmq-1	047511e2d6b8	rabbitmq:3-15672:15672	-	1.39%	141.8MB / 7.66Gi	1.81%	⋮
transaction-db-1	0f8b9328e0e1	postgres:11-5434:5432	-	0%	28.27MB / 7.66Gi	0.36%	⋮
account-service-1	814292d9be07	transaction:8083:8083	-	0%	29.18MB / 7.66Gi	0.37%	⋮
notification-service-1	cc3c1c01f136	transaction:8084:8084	-	0.01%	22.39MB / 7.66Gi	0.29%	⋮
transaction-service-1	bfc9a3bf876a	transaction:8082:8082	-	1.3%	42.46MB / 7.66Gi	0.54%	⋮
prometheus	0ab823bcc20e	prom/prom:9090:9090	-	0.13%	25.62MB / 7.66Gi	0.33%	⋮
grafana	24ad10b42108	grafana/grafana:80:3000	-	0.54%	89.24MB / 7.66Gi	1.14%	⋮

Grafana dashboard:

for visual metrics of transaction service



Prometheus:

The screenshot shows the Prometheus web interface at localhost:9090/targets. The top navigation bar includes links for Query, Alerts, Status, and Target health. The main content area displays a table for the 'transaction-service' scrape pool. It lists one endpoint: <http://transaction-service:8082/metrics>, which has labels `instance="transaction-service:8082"` and `job="transaction-service"`. The last scrape was 4.22s ago, and the state is UP.

**Metric in Prometheus:**

The screenshot shows the Prometheus web interface at localhost:9090/tsdb-status. The top navigation bar includes links for Query, Alerts, Status, and TSDB status. The main content area displays two tables. The first table shows TSDB statistics: kind (351) and type (152). The second table, titled 'Top 10 series count by label value pairs', lists metrics and their counts, such as `instance=transaction-service:8082` (Count 148), `job=transaction-service` (Count 148), and `method=GET` (Count 45).

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

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

Key	Value	Description
Content-Type	application/json	

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

```
{
  "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

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8082/transactions/deposit`. The request method is POST. The response status is 200 OK, with a response time of 86 ms and a response size of 615 B. The response body is a JSON object:

```

{
  "transaction": {
    "tx_id": "314",
    "account_id": "101",
    "amount": "100.00",
    "txn_type": "DEPOSIT",
    "counterparty": "SYSTEM:Deposit",
    "reference": "REF-1762542148184",
    "status": "SUCCESS",
    "created_at": "2025-11-07T14:02:28.184Z",
    "reused": true
  }
}

```

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

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

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

```
{
  "success": true,
  "sender_transaction": {
    "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"
  },
  "receiver_transaction": {
    "tx_id": "316",
    "account_id": "12"
  }
}
```

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

Body	Cookies	Headers (10)	Test Results
{ "from_account_id": 10, "to_account_id": 12, "amount": 2002 }			201 Created • 130 ms • 835 B •

```
{
  "success": true,
  "sender_transaction": {
    "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"
  },
  "receiver_transaction": {
    "tx_id": "316",
    "account_id": "12"
  }
}
```

```
{
  "success": true,
  "sender_transaction": {
    "tx_id": "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 body is a JSON object:

```

{
  "success": true,
  "transaction": {
    "txn_id": "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"
  },
  "reused": true
}

```

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 is a JSON object:

```

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

```

{

```

"success": true,
"transaction": {
  "txnid": "16",
  "account_id": "21",
  "amount": "133.53",
  "txntype": "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

Queues

All queues (1)

Overview		Messages			Message rates					
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

Add a new queue

HTTP API Documentation Tutorials New releases Commercial edition Commercial support Discussions Discord Plugins GitHub



Queue transaction_events

Overview

Queued messages last minute ?

Message rates last minute ?

Feature	Value
Features	durable: true queue storage version: 1
Policy	Consumer capacity ? 100%
Operator policy	Effective policy definition

Details

Feature	Value	State	Messages	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Consumers	1	idle	?	0	0	0	0	0	0
Message body bytes	?	0 B	0 B	0 B	0 B	0 B	0 B	0 B	0 B
Process memory	?	10 KiB							

ENGLISH IN 12:37 AM 08-11-2025

Queued messages last minute:

Category	Value
Ready	0
Unacked	0
Total	0

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

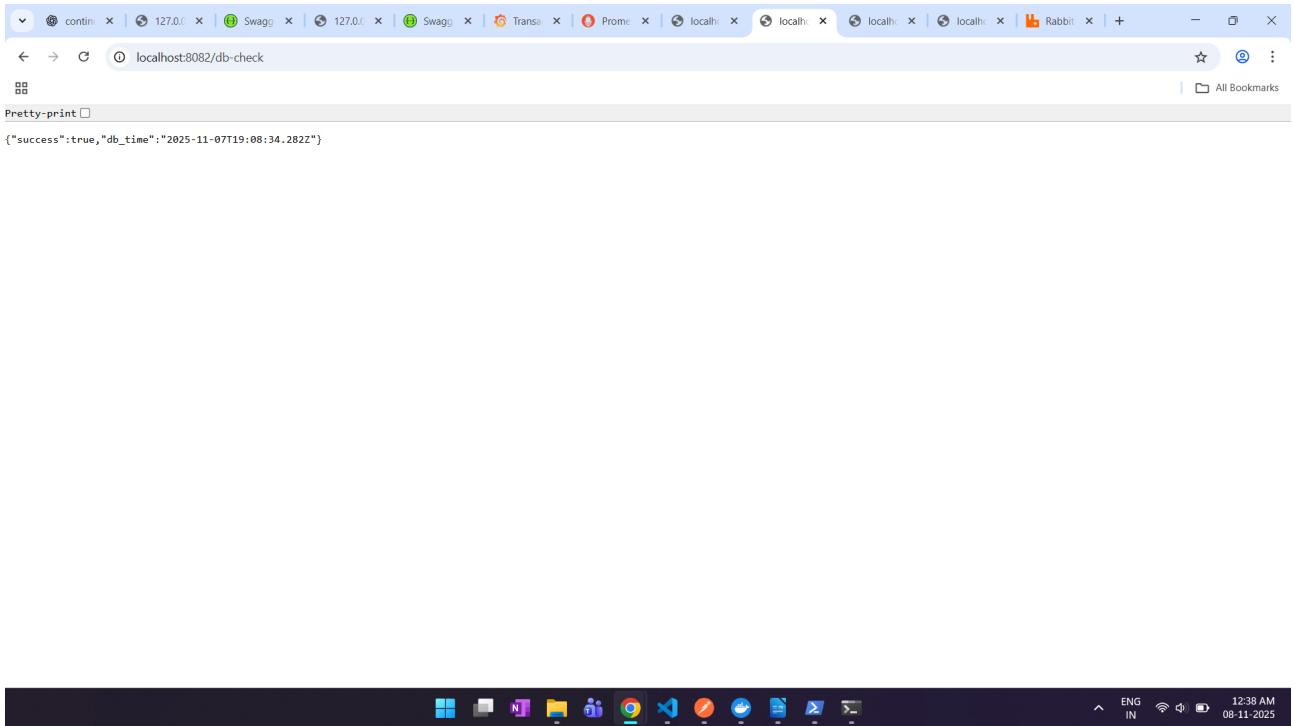
Details:

Feature	Value	State	Consumers	Messages	Total	Ready	Unacked	In memory	Persistent	Transient	Paged Out
Features	durable: true queue storage version: 1	idle	1	0 B	0 B	0 B	0 B	0 B	0 B	0 B	0 B
Policy				Message body bytes	0 B	0 B	0 B	0 B	0 B	0 B	0 B
Operator policy				Process memory	10 KiB						
Effective policy definition											

Transaction-service health check:

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

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="Inf",method="POST",route="/transactions/transfer",status_code="201"} 1
http_request_duration_seconds_sum{method="POST",route="/transactions/transfer",status_code="201"} 0.11985457300010603
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="Inf",method="POST",route="/transactions/transfer",status_code="200"} 1
http_request_duration_seconds_sum{method="POST",route="/transactions/transfer",status_code="200"} 0.01837367706005416
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="/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="50",method="GET",route="/16",status_code="200"} 3
http_request_duration_seconds_bucket{le="100",method="GET",route="/16",status_code="200"} 3
http_request_duration_seconds_bucket{le="200",method="GET",route="/16",status_code="200"} 3
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 58.5078869999857
balance_check_latency_ms_count 3
# 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 58.5078869999857
balance_check_latency_ms_count 3

```

Swagger

Showing all API of transaction-service

The screenshot shows the Transaction Service API documentation page. At the top, it says "Transaction Service API 1.0.0 OAS 3.0". Below that, a brief description states: "REST API for the Transaction Service in the Scalable Banking system. Handles money transfers, deposits, queries, and exposes Prometheus metrics." A dropdown menu "Servers" is set to "Dynamic base (matches current host)". The main content area is titled "default" and lists several API endpoints:

- 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 status bar at the bottom shows the date and time as 12:41 AM 08-11-2025.

This screenshot shows the details for the "/metrics" endpoint under the "default" section. It includes a "Parameters" section (empty), an "Execute" button, and a "Clear" button. The "Responses" section contains a "Curl" command and a "Request URL" field. Under "Server response", there are tabs for "Code" and "Details". The "Code" tab shows a response body with Prometheus metrics data:

```

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

http://localhost:8082/metrics

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

The status bar at the bottom shows the date and time as 12:43 AM 08-11-2025.

The screenshot shows a Swagger UI interface for a REST API. The URL in the browser is `localhost:8082/api-docs/#/default/listTransactions`. The request method is `GET` and the endpoint is `/transactions`. The description is "List all transactions or filter by account_id".
Parameters section:
Name: account_id, Description: Filter transactions by account ID, Type: integer (query), Value: 10.
Buttons: Execute, Clear, Cancel.
Responses section:
Curl command:

```
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: 200, Details: Response body.

The screenshot shows a Swagger UI interface for a REST API. The URL in the browser is `localhost:8082/api-docs/#/default/listTransactions`. The request method is `GET` and the endpoint is `/transactions`. The description is "List all transactions or filter by account_id".
Parameters section:
Name: account_id, Description: Filter transactions by account ID, Type: integer (query), Value: 10.
Buttons: Execute, Clear.
Responses section:
Curl command:

```
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: 200, Details: Response body.
The response body is a JSON object:

```
{
  "success": true,
  "count": 15,
  "transactions": [
    {
      "txn_id": "315",
      "account_id": 10,
      "amount": "2692.00",
      "txn_type": "TRANSFER_OUT",
      "counterparty": "TO:12",
      "reference": "REF-OUT-1762541913324",
      ...
    }
  ]
}
```

System status bar at the bottom right: ENG IN 12:44 AM 08-11-2025

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
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": 3,
  "data": [
    {
      "id": "7e0dab25-ab32-42de-bcbf-19f3d431b812",
      "type": "transfer",
      ...
    }
  ]
}
```

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 `curl -X 'GET' \n'http://localhost:8082/transactions/account/101' \n-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", ... }] }</pre>

Response headers:

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

Parameters

Name	Description
id * required	string (path)

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

POST /transactions/transfer Create a transfer (idempotent)

Parameters

Name	Description
Idempotency-Key * required	Unique key to ensure idempotent request handling. string (header)
X-Correlation-ID	Correlation ID for traceability. string (header)

Request body required

application/json

Edit Value Schema

```
{
  "from_account_id": 101,
  "to_account_id": 202,
  "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": 101,
    "to_account_id": 202,
    "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-Http4fKuyXizpXjz241/hgAAB24"
keep-alive: timeout=5
x-correlation-id: d729a8f0-8fe1-486e-8ac3-e0cd8646ac0
x-powered-by: Express
```

Responses



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": 10,
    "to_account_id": 12,
    "amount": 250.5
}'
```

Request URL

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

Server response

Code	Details
201	Response body

Response body

```
{
  "success": true,
  "sender_transaction": {
    "tx_id": "318",
    "account_id": "10",
    "amount": "250.50",
    "tx_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",
    "tx_type": "TRANSFER_IN",
    "counterparty": "FROM:10",
    "reference": "REF-IN-1762543281438",
    "status": "SUCCESS",
    "created_at": "2025-11-07T19:21:21.437Z"
  }
}
```



ENG IN 12:51 AM 08-11-2025

The screenshot shows the Swagger UI for a transaction creation endpoint. The URL in the address bar is `localhost:8082/api-docs/#/default/createTransfer`. The request body is defined as:

```
{
  "from_account_id": 10,
  "to_account_id": 12,
  "amount": 250000000
}
```

Below the body, there is a 'Responses' section. It contains a 'Curl' command and a 'Request URL'.

```
curl -X 'POST' \
  '/api/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": 250000000
  }'
```

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

The screenshot shows the detailed response for the transaction creation endpoint. The URL in the address bar is `localhost:8082/api-docs/#/default/createTransfer`. The response body is a JSON object:

```
{
  "created_at": "2025-11-08T10:00:00Z",
  "id": "7e0db25-ab32-42de-bcbd-19f3d431b812",
  "reused": true
}
```

Below the response body, there is a 'Response headers' section. It lists several 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: Mon, 08 Nov 2025 19:22:54 GMT
etag: W/"f1c29bcb4a5100ff3yJldrIQ"
keep-alive: timeout=5
x-correlation-id: acd7a2fe-6d4a-4147-b846-e50905d4bcfe
x-powered-by: Express
```

Below the response headers, there is a 'Responses' table. It has columns for 'Code', 'Description', and 'Links'.

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

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

```

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"} 0.1708155080004068
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="Inf",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
http_request_duration_seconds_bucket{le="0.05",method="POST",route="/transactions/deposit",status_code="400"} 1
http_request_duration_seconds_bucket{le="0.1",method="POST",route="/transactions/deposit",status_code="400"} 1
http_request_duration_seconds_bucket{le="0.2",method="POST",route="/transactions/deposit",status_code="400"} 1
http_request_duration_seconds_bucket{le="0.5",method="POST",route="/transactions/deposit",status_code="400"} 1
http_request_duration_seconds_bucket{le="1",method="POST",route="/transactions/deposit",status_code="400"} 1
http_request_duration_seconds_bucket{le="2",method="POST",route="/transactions/deposit",status_code="400"} 1
http_request_duration_seconds_bucket{le="5",method="POST",route="/transactions/deposit",status_code="400"} 1
http_request_duration_seconds_bucket{le="Inf",method="POST",route="/transactions/deposit",status_code="400"} 1
http_request_duration_seconds_sum{method="POST",route="/transactions/deposit",status_code="400"} 0.0007274000002071261
http_request_duration_seconds_count{method="POST",route="/transactions/deposit",status_code="400"} 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 132.7482349998317
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. The output shows logs from transaction-service-1 indicating a successful POST request for a deposit.
- Bottom Status Bar:** Shows the date (08-11-2025), time (12:57 AM), and system status (ENG IN).

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 titled "Transfer Transaction" is being tested against the URL http://localhost:8082/transactions/transfer. The request body is JSON with fields: "from_account_id": 10, "to_account_id": 12, and "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 date (08-11-2025), time (12:59 AM), and system status (ENG IN).

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
3
4
5
6
7
8   data:
9     transaction-red.json: |
10       {
11         "panels": [
12           {
13             "grid": {
14               "x": 0,
15               "y": 0,
16               "w": 12,
17               "h": 12
18             }
19           }
20         ],
21         "templating": {
22           "list": []
23         },
24         "time": {
25           "from": "now-15m",
26           "to": "now"
27         },
28         "timepicker": {
29           "refresh_intervals": ["5s", "10s", "30s", "1m", "5m"]
30         }
31       }
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123

```

The Terminal tab at the bottom 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

```

```

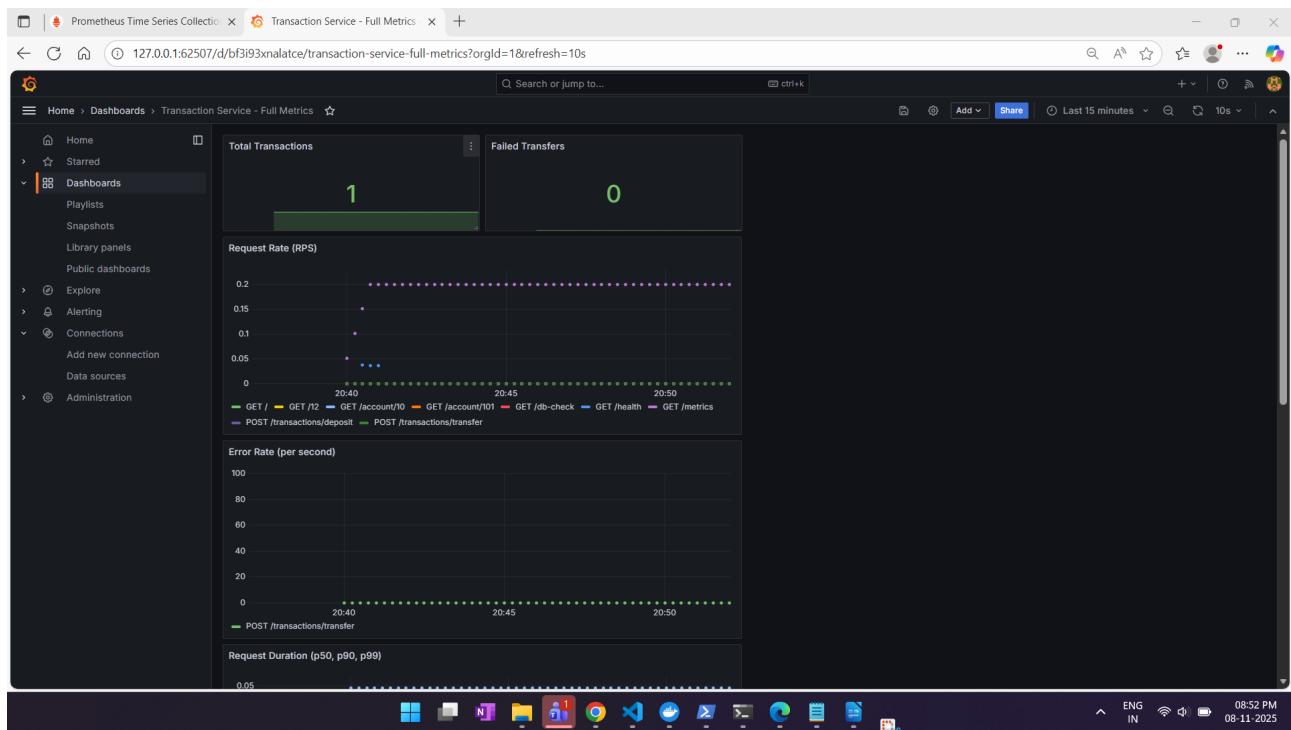
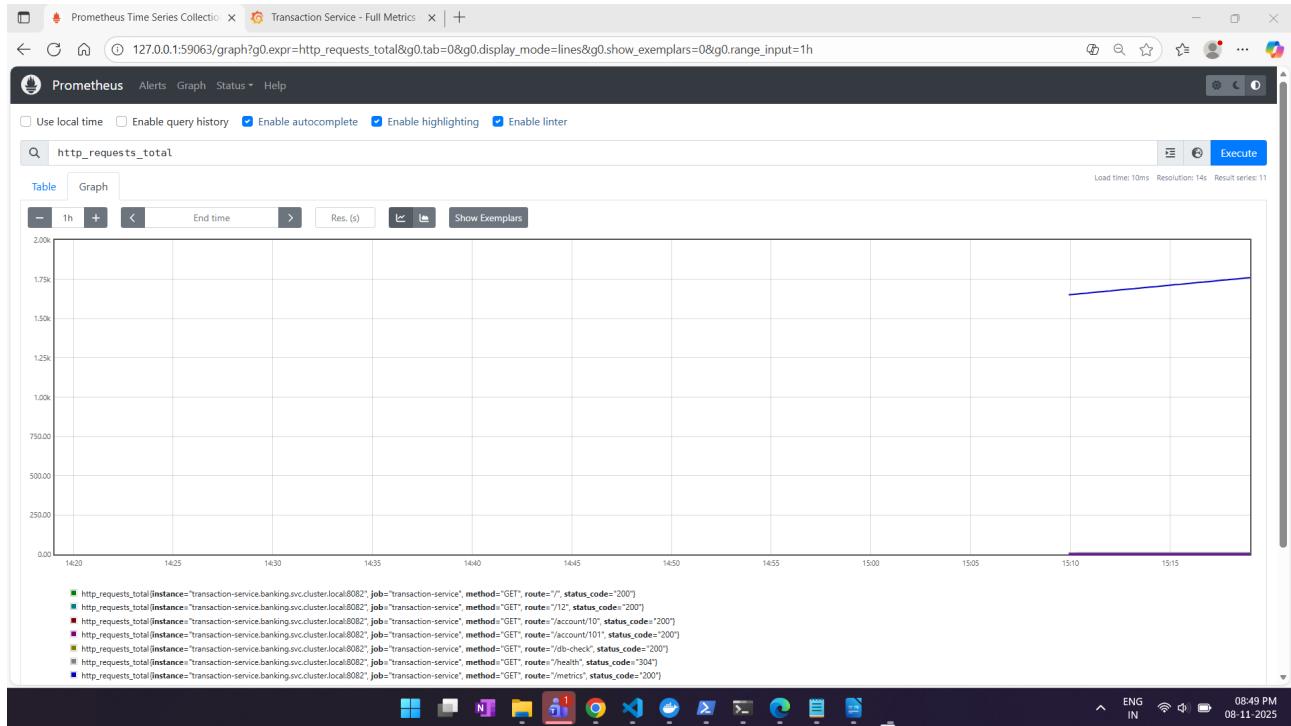
PS C:\Users\Jatin> minikube service rabbitmq -n banking
* Starting tunnel for service rabbitmq./
| NAMESPACE | NAME | TARGET PORT | URL |
| banking   | rabbitmq |             | http://127.0.0.1:49676
|           |         |             | http://127.0.0.1:49677 |
* Starting tunnel for service rabbitmq.
[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.
|
```

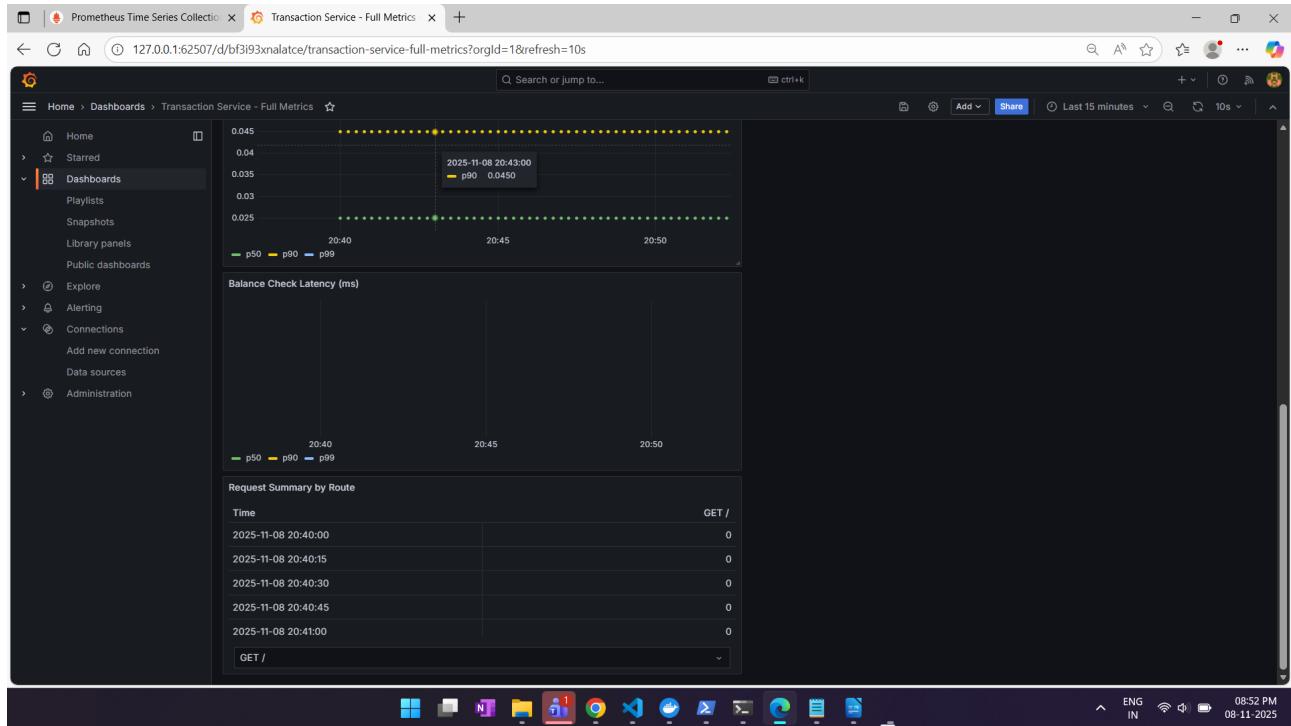
```

PS C:\Users\Jatin> minikube service grafana -n banking
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

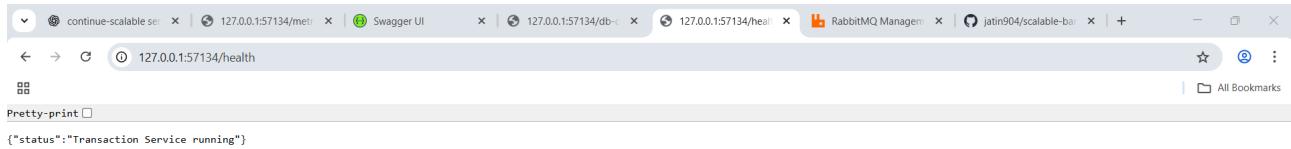
http://127.0.0.1:57134/transactions/account/101

Server response

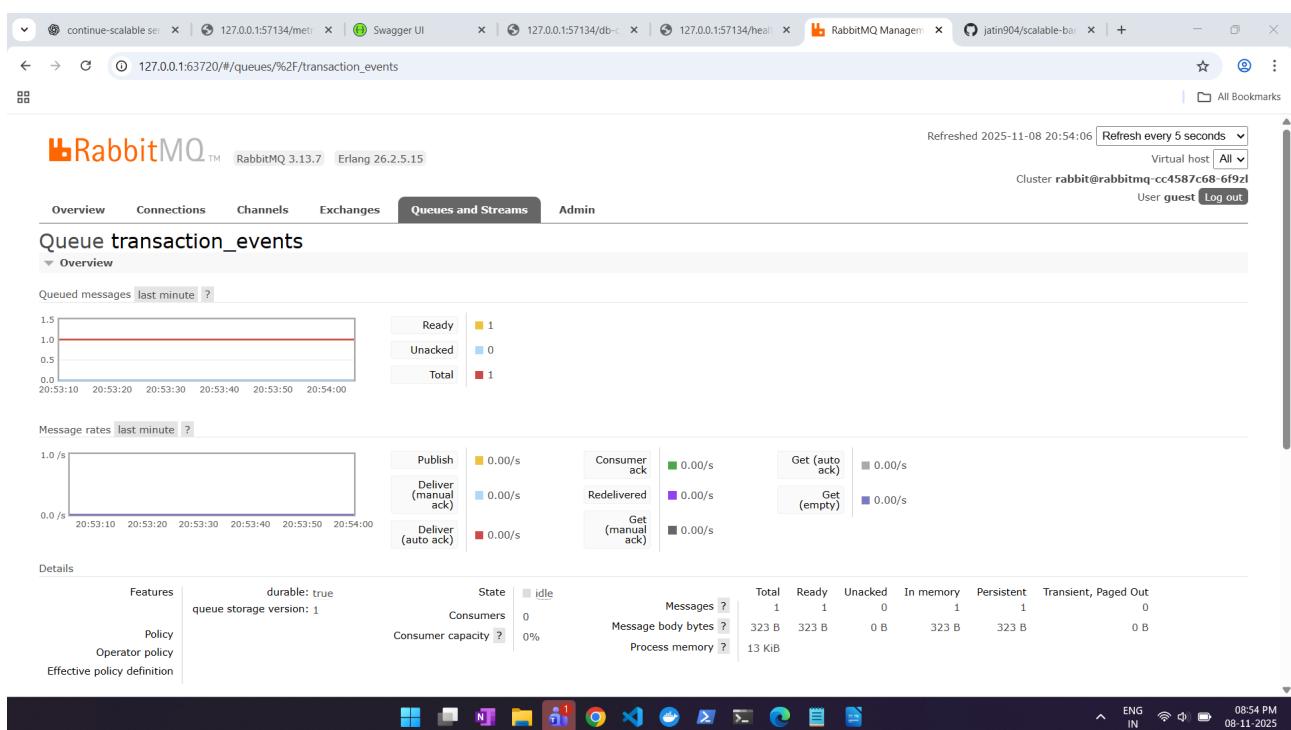
Code	Details
200	<p>Response body</p> <pre>{ "success": true, "count": 1, "data": [{ "tx_id": "382", "account_id": "101", "amount": "500.00", "tx_type": "DEPOSIT", "counterparty": "SYSTEM:Deposit", "reference": "REF-1762592163377", "status": "SUCCESS", "created_at": "2023-11-08T08:56:03.377Z" }] }</pre>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

Pretty-print

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



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

RabbitMQ™ RabbitMQ 3.13.7 Erlang 26.2.5.15

Refreshed 2025-11-08 20:54:06 | Refresh every 5 seconds | Virtual host All | Cluster **rabbit@rabbitmq-cc4587c68-6f9z1** User **guest** Log out

Queue transaction_events

Overview

Queued messages last minute [?]

Ready: 1
Unacked: 0
Total: 1

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	Consumer capacity	Messages	Total	Ready	Unacked	In memory	Persistent	Transient	Paged Out
Features	durable: true queue storage version: 1	idle	0	0%	Messages ?	1	1	0	1	1	0	
Policy					Message body bytes ?	323 B	323 B	0 B	323 B	323 B	0 B	
Operator policy					Process memory ?	13 kB						
Effective policy definition												



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", "updated": "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