Explicação Técnica - Como o Matcher Funciona

Documento: Guia de Entendimento do Produto para Pricing **Objetivo:** Explicar o fluxo de operações e o racional de sizing da infraestrutura **Data:** Janeiro 2026

---

1. Visão Geral: O que o Matcher faz?

O **Matcher** é uma engine de reconciliação financeira. Ele recebe transações de múltiplas fontes (banco, gateway de pagamento, ledger interno) e automaticamente encontra quais transações "casam" entre si.

Exemplo Prático

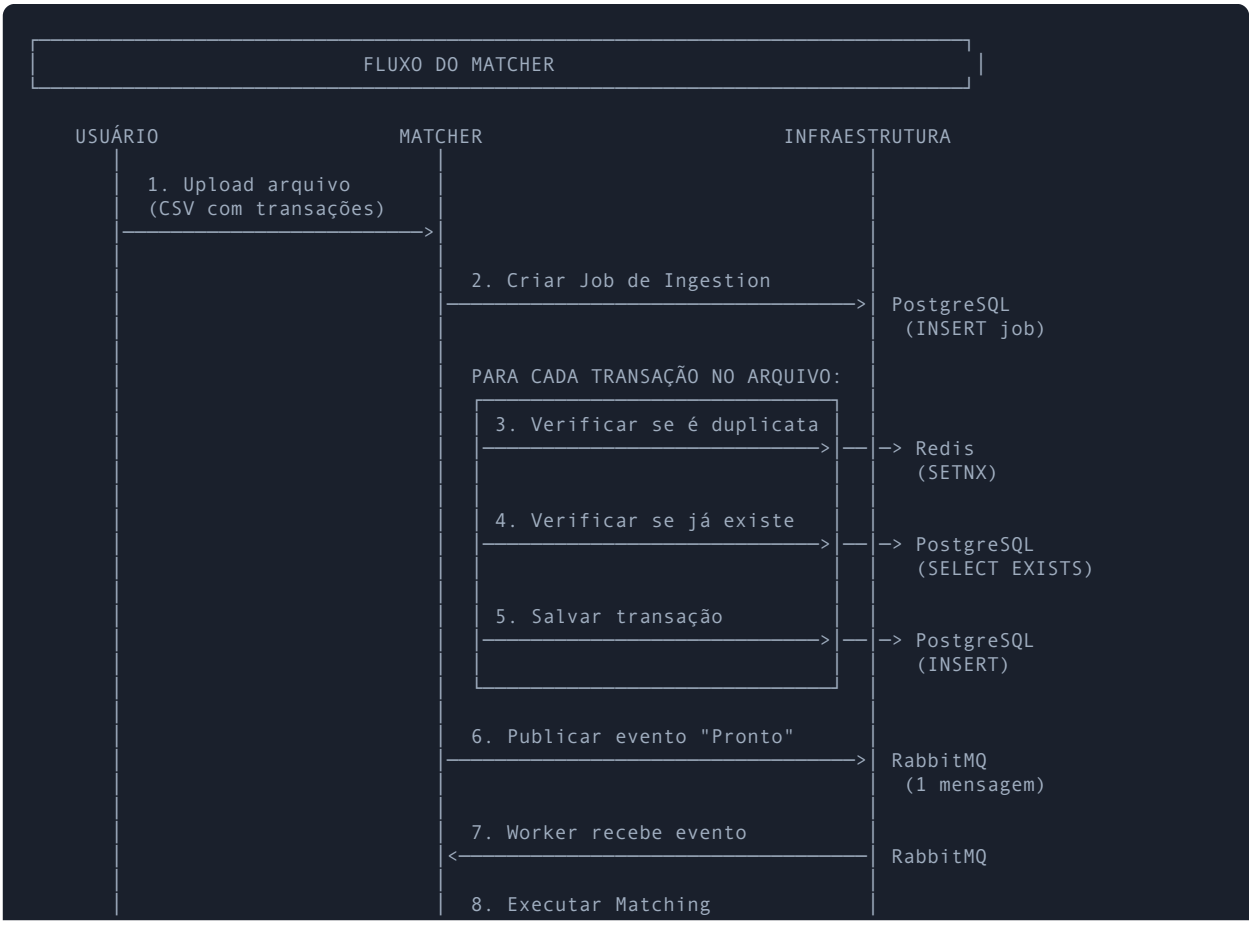
Imagine que você tem: - **Fonte A (Seu sistema - Midaz):** "Venda #123 de R$ 100,00 em 10/01" - **Fonte B (Banco):** "Crédito de R$ 97,00 em 12/01" (com taxa de 3%)

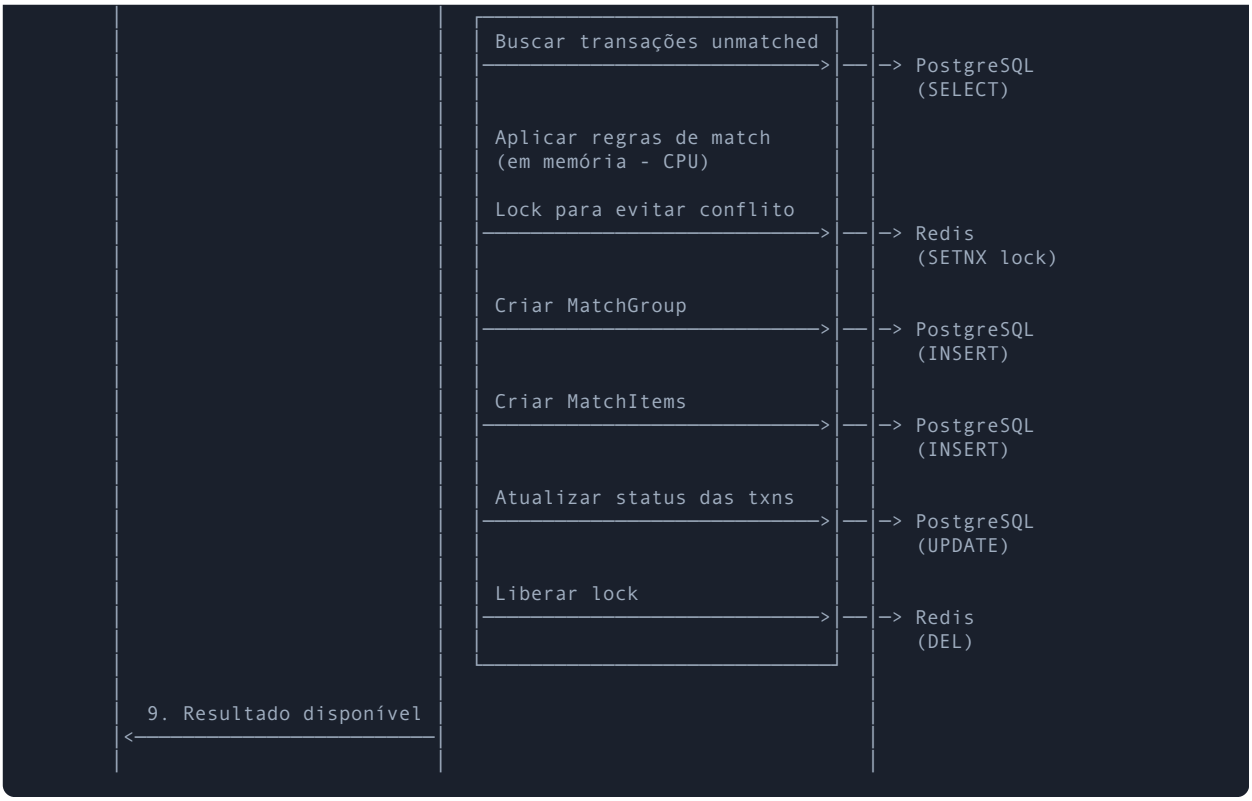
O Matcher identifica que essas duas transações são a mesma operação, considerando: - A diferença de valor (taxa do banco) - A diferença de data (tempo de compensação) - Regras configuradas pelo usuário

---

2. Fluxo Completo de Operações

2.1 Diagrama do Fluxo





---

3. Detalhamento: O que acontece em cada etapa?

3.1 INGESTION (Ingestão de Dados)

Quando você faz upload de um arquivo CSV com 1.000 transações, o que acontece:

Passo 1: Criar Job

```
PostgreSQL: INSERT INTO ingestion_jobs (id, context_id, status, ...)
            VALUES ('uuid', 'ctx-uuid', 'PROCESSING', ...)
```

Operações: 1 INSERT no PostgreSQL

Passo 2: Para CADA transação no arquivo (1.000x):

a) Verificar duplicata no Redis:

```
Redis: SETNX matcher:dedupe:ctx-uuid:hash-da-transacao "1"
```

- Se retornar `1` (setou) → transação é nova
- Se retornar `0` (já existia) → é duplicata, pula

Operações: 1 operação Redis por transação

b) Verificar se já existe no banco:

```
PostgreSQL: SELECT EXISTS (
              SELECT 1 FROM transactions
              WHERE source_id = 'src-uuid' AND external_id = 'TXN-123'
            )
```

Operações: 1 SELECT no PostgreSQL por transação

c) Salvar a transação:

```
PostgreSQL: INSERT INTO transactions (id, amount, currency, date, ...)
            VALUES ('uuid', 100.00, 'BRL', '2026-01-10', ...)
```

Operações: 1 INSERT no PostgreSQL por transação

Passo 3: Publicar evento de conclusão

```
RabbitMQ: PUBLISH to exchange "matcher.events"
          routing_key "ingestion.completed"
          payload { job_id: 'uuid', transaction_count: 1000, ... }
```

Operações: 1 mensagem no RabbitMQ por JOB (não por transação!)

3.2 MATCHING (Reconciliação)

Quando o Worker recebe o evento de "ingestion completed", ele executa o matching:

Passo 1: Buscar transações não reconciliadas

```
PostgreSQL: SELECT * FROM transactions
            WHERE context_id = 'ctx-uuid'
            AND status = 'UNMATCHED'
            AND extraction_status = 'COMPLETE'
```

Operações: 1 SELECT (retorna muitas transações de uma vez)

Passo 2: Aplicar regras de matching (em memória)

O algoritmo compara as transações em memória:

```
// Pseudocódigo do matching
for _, txnA := range transacoesDoMidaz {
    for _, txnB := range transacoesDoBanco {
        if matchExato(txnA, txnB) || matchComTolerancia(txnA, txnB) {
            candidatos = append(candidatos, Match{txnA, txnB, confianca})
        }
    }
}
```

Operações: CPU apenas, sem I/O de banco

Passo 3: Para cada MATCH encontrado:

a) Adquirir lock distribuído (evitar conflito):

```
Redis: SETNX matcher:lock:txn-uuid-A "worker-1" EX 30
Redis: SETNX matcher:lock:txn-uuid-B "worker-1" EX 30
```

Operações: 2 operações Redis por match

b) Criar o MatchGroup:

```
PostgreSQL: INSERT INTO match_groups (id, context_id, confidence, status, ...)
            VALUES ('mg-uuid', 'ctx-uuid', 95, 'CONFIRMED', ...)
```

Operações: 1 INSERT no PostgreSQL por match

c) Criar os MatchItems (linkando as transações):

```
PostgreSQL: INSERT INTO match_items (id, match_group_id, transaction_id, allocated_amount, ...)
            VALUES ('mi-uuid-1', 'mg-uuid', 'txn-uuid-A', 100.00, ...),
                   ('mi-uuid-2', 'mg-uuid', 'txn-uuid-B', 97.00, ...)
```

Operações: 1-2 INSERTs no PostgreSQL por match (depende de quantas txns no match)

d) Atualizar status das transações:

```
PostgreSQL: UPDATE transactions SET status = 'MATCHED' WHERE id IN ('txn-uuid-A', 'txn-uuid-B')
```

Operações: 1 UPDATE no PostgreSQL por match

e) Liberar locks:

```
Redis: DEL matcher:lock:txn-uuid-A
Redis: DEL matcher:lock:txn-uuid-B
```

Operações: 2 operações Redis por match

---

4. Resumo: Operações por Transação vs por Match

4.1 Tabela de Operações

Fase	Componente	Operação	Frequência	Quantidade
INGESTION	Redis	SETNX (dedupe)	Por transação	1
INGESTION	PostgreSQL	SELECT EXISTS	Por transação	1
INGESTION	PostgreSQL	INSERT transaction	Por transação	1
INGESTION	RabbitMQ	PUBLISH evento	Por JOB	1
MATCHING	PostgreSQL	SELECT unmatched	Por JOB	1
MATCHING	Redis	SETNX + DEL (locks)	Por MATCH	4
MATCHING	PostgreSQL	INSERT match_group	Por MATCH	1
MATCHING	PostgreSQL	INSERT match_items	Por MATCH	2
MATCHING	PostgreSQL	UPDATE status	Por MATCH	1

4.2 Exemplo Numérico

Cenário: Upload de arquivo com 1.000 transações, taxa de match de 80%

Etapa	Cálculo	Total Operações
Ingestion - Redis	1.000 txns × 1 op	1.000 ops Redis
Ingestion - PostgreSQL	1.000 txns × 2 ops	2.000 ops PostgreSQL
Ingestion - RabbitMQ	1 job × 1 msg	1 msg RabbitMQ
Matching - Redis	800 matches × 4 ops	3.200 ops Redis

Etapa	Cálculo	Total Operações
Matching - PostgreSQL	800 matches × 4 ops	3.200 ops PostgreSQL
TOTAL		**4.200 Redis, 5.200 PostgreSQL, 1 RabbitMQ**

Por transação processada: ~4,2 ops Redis + ~5,2 ops PostgreSQL

---

5. Racional de Sizing: Por que escolhi cada instância?

5.1 PostgreSQL (Amazon RDS)

O que o PostgreSQL faz no Matcher:

- Armazena todas as transações
- Armazena configurações (contextos, regras, field maps)
- Armazena matches e exceptions
- Armazena audit log (compliance SOX)

Como dimensionar:

Variáveis principais: 1. **Queries por segundo (QPS)** - Capacidade de processamento 2. **Armazenamento (GB)** - Tamanho dos dados 3. **Memória (RAM)** - Cache de queries frequentes

Cálculo para cada tier:

Tier	Transações/mês	Transações/dia	QPS estimado*	RAM necessária
Starter	500K	~17K	~5 QPS	4GB suficiente
Growth	2M	~67K	~20 QPS	8GB recomendado
Scale	10M	~333K	~100 QPS	16GB necessário

*Assumindo processamento distribuído ao longo do dia

Escolha de instâncias:

STARTER: db.t3.medium (2 vCPU, 4GB RAM)

Critério	Análise
CPU	2 vCPU com burst suficiente para 5-10 QPS
RAM	4GB permite cache de ~1GB para queries
Storage	100GB gp3 = ~600K transações*
Preço	$80/mês - menor custo possível com performance aceitável

*Estimativa: ~150 bytes/transação + indexes

Por que NÃO db.t3.micro? - Apenas 1GB RAM - insuficiente para PostgreSQL com GORM - CPU burst limitado - pode throttle em picos - Risco de OOM (Out of Memory) com queries complexas

Por que NÃO db.t3.small? - 2GB RAM - margem pequena para crescimento - Diferença de custo para t3.medium é pequena (~$20/mês)

GROWTH: db.t3.large (2 vCPU, 8GB RAM)

Critério	Análise
CPU	Mesmo 2 vCPU, mas com mais burst credits
RAM	8GB permite cache maior, menos disk I/O
Storage	250GB gp3 = ~1.5M transações
Preço	$160/mês

Por que 8GB de RAM? - 2M transações/mês = queries mais pesadas - Joins entre transactions, match_groups, match_items - Buffer pool maior = menos leituras de disco

---

SCALE: db.r6g.large (2 vCPU, 16GB RAM) + Read Replica

Critério	Análise
CPU	Graviton (ARM) - melhor custo/performance
RAM	16GB - cache significativo
Read Replica	Separa leitura de escrita
Storage	500GB gp3 com IOPS provisionado
Preço	$560/mês (primary + replica)

Por que Read Replica? - 10M transações/mês gera ~100+ QPS - Reporting e dashboard não devem impactar writes - O PRD do Matcher menciona: "Database scales with Read Replicas"

5.2 Redis (Amazon ElastiCache)

O que o Redis faz no Matcher:

1. **Deduplicação** - Evita processar a mesma transação duas vezes
2. **Locking distribuído** - Evita que dois workers façam match da mesma transação

Como dimensionar:

Variáveis principais: 1. **Memória para keys** - Cada transação = 1 key de dedupe 2. **Operações por segundo** - SETNX, GET, DEL 3. **TTL das keys** - Por quanto tempo manter

Cálculo de memória:

```
Memória por key de dedupe:
- Key: "matcher:dedupe:ctx-uuid:hash" = ~80 bytes
- Value: "1" = ~8 bytes
- Overhead Redis: ~50 bytes
- Total: ~140 bytes/key

Para 500K transações/mês com TTL de 30 dias:
500K × 140 bytes = 70MB
```

```
Com margem de segurança (2x): 140MB
```

Escolha de instâncias:

STARTER: cache.t3.small (2 vCPU, 1.5GB RAM)

Critério	Análise
Memória	1.5GB >> 140MB necessário
Operações	Suporta ~25K ops/segundo
Preço	$25/mês

Por que NÃO cache.t3.micro? - Apenas 0.5GB RAM - Risco de eviction se TTL das keys for longo - Diferença de $8/mês não justifica o risco

---

GROWTH: cache.t3.medium (2 vCPU, 3GB RAM)

Critério	Análise
Memória	3GB para 2M transações
Operações	~50K ops/segundo
Preço	$50/mês

SCALE: cache.r6g.large (2 vCPU, 13GB RAM)

Critério	Análise
Memória	13GB para 10M+ transações
Operações	~100K ops/segundo
Graviton	Melhor custo/performance
Preço	$150/mês

Por que upgrade significativo no Scale? - Locking distribuído com alto volume precisa de baixa latência - Mais memória = menos evictions = menos falsos positivos de duplicata

---

5.3 RabbitMQ (Amazon MQ)

O que o RabbitMQ faz no Matcher:

1. **Desacoplar ingestion de matching** - Upload não espera matching terminar
2. **Eventos assíncronos** - Notificar quando processos completam

Insight importante: Volume BAIXO de mensagens

```
Análise do código (event_publisher.go):
```

```
- PublishIngestionCompleted() → 1 mensagem por JOB
- PublishIngestionFailed() → 1 mensagem por JOB falho

Se um JOB processa 1.000 transações:
- 500K transações/mês ÷ 1.000 txns/job = 500 jobs/mês
- 500 jobs/mês = ~17 mensagens/dia
```

Conclusão: RabbitMQ tem uso MUITO BAIXO no Matcher!

Escolha de instâncias:

STARTER e GROWTH: mq.t3.micro (2 vCPU, 1GB RAM)

Critério	Análise
Capacidade	Suporta ~1.000 msg/segundo
Uso real	~17-70 msg/dia (irrelevante)
Preço	$55/mês

Por que manter mq.t3.micro mesmo no Growth? - Volume de mensagens não justifica upgrade - Bottleneck NUNCA será o RabbitMQ - Custo fixo de managed service

---

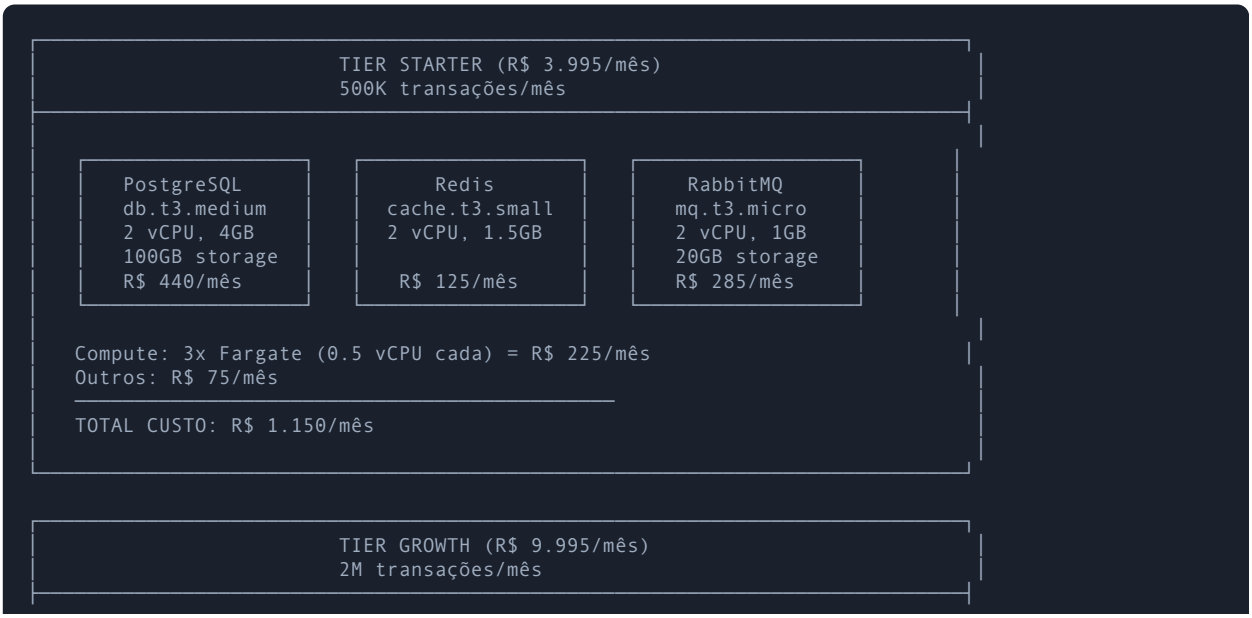
SCALE: mq.m5.large (2 vCPU, 8GB RAM)

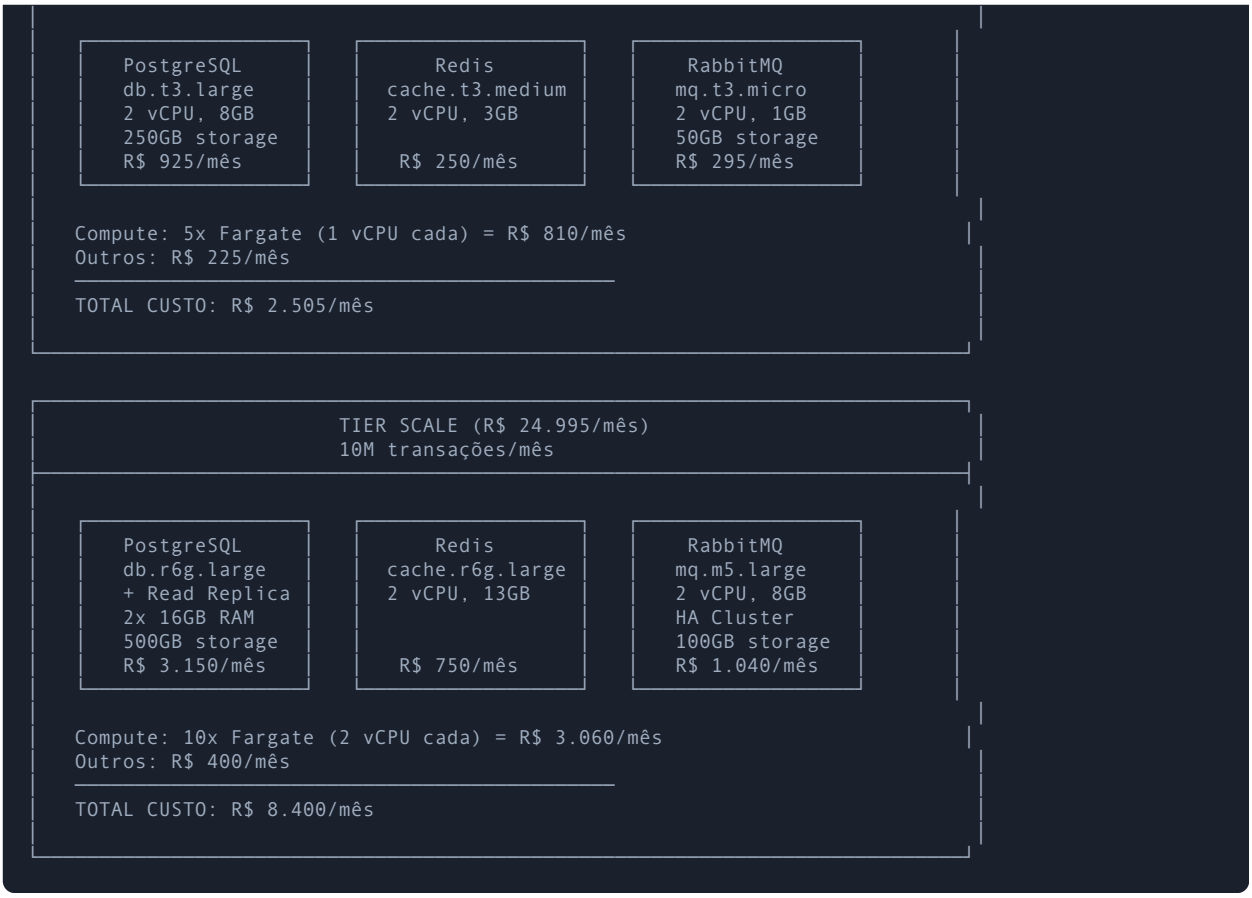
Critério	Análise
Capacidade	Suporta clustering
HA	Alta disponibilidade para enterprise
Preço	$200/mês

Por que upgrade no Scale? - Não é por volume de mensagens - É por **SLA e alta disponibilidade** - Clientes Scale esperam 99.9%+ uptime - mq.t3.micro não suporta cluster mode

---

6. Resumo Visual: Sizing por Tier





---

7. Entendendo o "Custo por Match"

7.1 O que é um Match?

Um **Match** é quando o sistema encontra que duas (ou mais) transações representam a mesma operação financeira.

```
Exemplo de Match 1:1 (mais comum):

Transação A (Midaz):     Transação B (Banco):
├─ ID: TXN-001           ├─ ID: BANK-99887
├─ Valor: R$ 100,00      ├─ Valor: R$ 97,00
├─ Data: 10/01/2026      ├─ Data: 12/01/2026
└─ Ref: "Venda #123"     └─ Ref: "CRED TXN-001"

                    ↓ MATCH! ↓

MatchGroup:
├─ ID: MG-001
├─ Confiança: 95%
├─ Status: CONFIRMED
└─ MatchItems:
   ├─ Item 1: TXN-001, allocated: R$ 100,00
   └─ Item 2: BANK-99887, allocated: R$ 97,00
```

7.2 Operações por Match

Operação	Componente	Por quê?
SETNX lock txn A	Redis	Evitar que outro worker pegue a mesma transação
SETNX lock txn B	Redis	Idem

Operação	Componente	Por quê?
INSERT match_group	PostgreSQL	Registrar o match
INSERT match_item A	PostgreSQL	Linkar transação A ao match
INSERT match_item B	PostgreSQL	Linkar transação B ao match
UPDATE txn A status	PostgreSQL	Marcar como MATCHED
UPDATE txn B status	PostgreSQL	Marcar como MATCHED
DEL lock txn A	Redis	Liberar lock
DEL lock txn B	Redis	Liberar lock
TOTAL		**4 Redis + 5 PostgreSQL = 9 operações**

7.3 Custo por Match (estimativa)

Usando ElastiCache provisionado (não Serverless): - Redis: Custo fixo por hora, não por operação - PostgreSQL: Custo fixo por hora + IOPS se usar io1

Se usássemos Serverless (para referência):

```
ElastiCache Serverless: $0.0034/milhão ECPUs
Aurora Serverless: $0.20/milhão I/Os

Custo por match:
- Redis: 4 ops × $0.0000000034 = $0.0000000136
- PostgreSQL: 5 ops × $0.0000002 = $0.000001

Custo por match = ~$0.000001 = R$ 0,000005 por match
```

Conclusão: O custo VARIÁVEL por match é desprezível. O que importa é o custo FIXO da infraestrutura.

---

8. Por que a Metodologia "Standalone-First"?

8.1 O Problema

Se você cobra R$ 0,000005 por match, precisaria de **200 milhões de matches** para cobrir R$ 1.000 de custo fixo.

8.2 A Solução

Cobrar um **valor fixo mensal** que: 1. Cobre o custo da infraestrutura dedicada 2. Inclui um volume de transações 3. Tem margem de lucro embutida

8.3 Exemplo

```
Tier Starter:
├─ Custo fixo de infra: R$ 1.150/mês
├─ Volume incluso: 500K transações
├─ Preço: R$ 3.995/mês
├─ Margem: 71%
│
├─ Se cliente usa 500K txns: Margem = 71%
```

```
├─ Se cliente usa 100K txns: Margem = 71% (mesmo custo fixo)
└─ Se cliente usa 600K txns: Cobra overage de R$ 2,50/1K txns extras
```

---

9. TPS → QPS → Infraestrutura: Dimensionamento por Carga

9.1 Entendendo a Relação TPS e QPS

TPS (Transactions Per Second): É a carga que o cliente gera. Se o cliente faz 1.000.000 de transações por mês: - 1.000.000 ÷ 30 dias ÷ 24h ÷ 3600s = **~0.4 TPS médio** - Com fator de pico (3x): **~1.2 TPS de pico**

QPS (Queries Per Second): É a carga real nos componentes de infraestrutura (PostgreSQL, Redis). Cada TPS gera múltiplas queries.

9.2 Fórmula: TPS → QPS

```
┌─────────────────────────────────────────────────────────────────────┐
│                      FÓRMULA TPS → QPS                              │
├─────────────────────────────────────────────────────────────────────┤
│                                                                     │
│   Para SIZING de infraestrutura, assumimos 100% match rate (pior caso)      │
│                                                                     │
│   Por transação processada:                                         │
│   ├─ INGESTION: 1 Redis + 2 PostgreSQL = 3 operações                │
│   └─ MATCHING:  4 Redis + 4 PostgreSQL = 8 operações (assumindo match 1:1) │
│                                                                     │
│   TOTAL: 5 Redis ops + 6 PostgreSQL ops = 11 operações por transação        │
│                                                                     │
│   ┌─────────────────────────────────────────────────────────────┐   │
│   │                                                             │   │
│   │   QPS_PostgreSQL = TPS × 6                                  │   │
│   │   QPS_Redis = TPS × 5                                       │   │
│   │   QPS_Total = TPS × 11                                      │   │
│   │                                                             │   │
│   └─────────────────────────────────────────────────────────────┘   │
│                                                                     │
│   Exemplo: Cliente com 10 TPS de pico                               │
│   ├─ PostgreSQL: 10 × 6 = 60 QPS                                    │
│   └─ Redis: 10 × 5 = 50 QPS                                         │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

9.3 De Transações/Mês para TPS de Pico

```
┌─────────────────────────────────────────────────────────────────────┐
│           CONVERSÃO: TRANSAÇÕES/MÊS → TPS PICO                      │
├─────────────────────────────────────────────────────────────────────┤
│                                                                     │
│   TPS_médio = Transações_mês ÷ (30 × 24 × 3600)                     │
│   TPS_pico = TPS_médio × FATOR_PICO                                 │
│                                                                     │
│   Fator de pico recomendado: 3x (padrão da indústria)               │
│                                                                     │
├──────────────┬─────────────┬──────────────┬─────────────────────────┤
│   Txns/Mês   │ TPS Médio   │ TPS Pico (3x)│ QPS Pico (×11)          │
├──────────────┼─────────────┼──────────────┼─────────────────────────┤
│   500.000    │    0.19     │     0.6      │      7 QPS              │
│   1.000.000  │    0.39     │     1.2      │     13 QPS              │
│   2.000.000  │    0.77     │     2.3      │     25 QPS              │
│   5.000.000  │    1.93     │     5.8      │     64 QPS              │
│  10.000.000  │    3.86     │    11.6      │    128 QPS              │
│  50.000.000  │   19.29     │    57.9      │    637 QPS              │
│ 100.000.000  │   38.58     │   115.7      │  1.273 QPS              │
└──────────────┴─────────────┴──────────────┴─────────────────────────┘
```

9.4 Tabela de Custos por TPS (Formato Midaz)

Esta tabela segue o mesmo formato da planilha de pricing do Midaz, mas para o Matcher:

MATCHER - CUSTOS POR TPS (SaaS)

TPS (pico)	Total/mês (USD)	Total/mês (BRL)	COMPONENTES					
			Fargate	PostgreSQL	Redis	RabbitMQ	Rede	Outros (logs, monitoring, S3)
1	$230.00	R$ 1.150	$45.00	$88.00	$25.00	$57.00	$5.00	$10.00
5	$370.00	R$ 1.850	$90.00	$132.00	$50.00	$57.00	$16.00	$25.00
10	$501.00	R$ 2.505	$162.00	$185.00	$50.00	$59.00	$25.00	$20.00
25	$748.00	R$ 3.740	$270.00	$280.00	$75.00	$59.00	$34.00	$30.00
50	$1.110.00	R$ 5.550	$405.00	$420.00	$100.00	$100.00	$45.00	$40.00
100	$1.680.00	R$ 8.400	$612.00	$630.00	$150.00	$208.00	$50.00	$30.00
200	$2.600.00	R$13.000	$918.00	$945.00	$300.00	$208.00	$79.00	$150.00
500	$5.200.00	R$26.000	$1.530.00	$1.890.00	$600.00	$416.00	$114.00	$650.00
1000	$9.600.00	R$48.000	$3.060.00	$3.360.00	$1.200.00	$624.00	$156.00	$1.200.00

Notas:
- Valores em USD convertidos a R$ 5,00/USD
- Assume 100% match rate para sizing conservador
- Fargate: API + Workers (escala com TPS)
- PostgreSQL: Instância RDS escala com QPS
- Redis: ElastiCache escala com memória e ops/s
- RabbitMQ: Amazon MQ (mínimo de ~$57/mês mesmo com baixo uso)

9.5 Comparação: Matcher vs Midaz

COMPARAÇÃO DE CUSTOS: MIDAZ vs MATCHER

TPS (pico)	MIDAZ Total/mês (USD)	MATCHER Total/mês (USD)	Diferença
50	$2.060,94	$1.110	-46% (Matcher mais barato)
100	$2.813,42	$1.680	
300	$5.090,83	$3.380	
600	$6.918,76	$6.100	
1000	$9.972,52	$9.600	

POR QUE O MATCHER É MAIS BARATO?

1. Sem DocumentDB (Midaz usa, Matcher não)
2. Sem EKS - usa Fargate direto (mais simples)
3. Menos storage - não guarda histórico completo
4. Menos CloudWatch - menos métricas
5. Aplicação mais simples - menos containers

O Midaz é um ledger completo com:
- DocumentDB para flexibilidade de schema
- EKS para orquestração complexa
- Mais réplicas para alta disponibilidade

O Matcher é focado em uma única função (reconciliação)

9.6 Como Usar na Conversa com Cliente

Cenário: Cliente diz "Fazemos 5 milhões de transações por mês"

Passo 1: Converter para TPS

5.000.000 ÷ (30 × 24 × 3600) = 1.93 TPS médio
Com fator de pico (3x): 5.8 TPS de pico

Passo 2: Encontrar o tier adequado

TPS 5.8 → Entre 5 e 10 TPS → Tier Growth ou início de Scale

```
Passo 3: Verificar custo de infra
──────────────────────────────────

~10 TPS de pico → $501/mês USD → R$ 2.505/mês

Passo 4: Aplicar margem para pricing
──────────────────────────────────

Custo: R$ 2.505/mês
Margem desejada: 70%
Preço sugerido: R$ 2.505 ÷ (1 - 0.70) = R$ 8.350/mês

Ou usar tier fixo: Growth = R$ 9.995/mês
```

9.7 Calculadora Rápida

```
┌──────────────────────────────────────────────────────────────────────┐
│                    CALCULADORA: TRANSAÇÕES → TIER                    │
├──────────────────────────────────────────────────────────────────────┤
│                                                                      │
│   Transações/mês        TPS Pico      Tier Recomendado    Preço Sugerido │
│   ──────────────        ────────      ────────────────    ──────────────  │
│   < 1 milhão            < 1.2         STARTER             R$ 3.995/mês   │
│   1-3 milhões           1.2 - 3.6     GROWTH              R$ 9.995/mês   │
│   3-15 milhões          3.6 - 18      SCALE               R$ 24.995/mês  │
│   > 15 milhões          > 18          ENTERPRISE          Custom         │
│                                                                      │
├──────────────────────────────────────────────────────────────────────┤
│   Fórmulas rápidas:                                                  │
│                                                                      │
│   TPS_médio = Txns_mês ÷ 2.592.000                                   │
│   TPS_pico = TPS_médio × 3                                           │
│   QPS_infra = TPS_pico × 11                                          │
│   Custo_base ≈ R$ 500 + (TPS_pico × R$ 450)                          │
│                                                                      │
└──────────────────────────────────────────────────────────────────────┘
```

9.8 Detalhamento dos Componentes por TPS

```
┌──────────────────────────────────────────────────────────────────────┐
│                    SIZING DE INSTÂNCIAS POR TPS                      │
└──────────────────────────────────────────────────────────────────────┘
```

TPS (pico)	PostgreSQL	Redis	RabbitMQ
1-5	db.t3.medium 2 vCPU, 4GB 100GB gp3 ~30 QPS capacity	cache.t3.small 2 vCPU, 1.5GB ~25K ops/s capacity	mq.t3.micro 2 vCPU, 1GB 20GB ~1K msg/s
5-20	db.t3.large 2 vCPU, 8GB 250GB gp3 ~100 QPS capacity	cache.t3.medium 2 vCPU, 3GB ~50K ops/s capacity	mq.t3.micro 2 vCPU, 1GB 50GB ~1K msg/s
20-50	db.r6g.large 2 vCPU, 16GB 500GB gp3 ~200 QPS capacity	cache.r6g.large 2 vCPU, 13GB ~100K ops/s capacity	mq.m5.large 2 vCPU, 8GB 100GB ~10K msg/s
50-100	db.r6g.xlarge + Read Replica 4 vCPU, 32GB × 2 1TB gp3 ~500 QPS capacity	cache.r6g.xlarge 4 vCPU, 26GB ~200K ops/s capacity	mq.m5.large (HA) Cluster mode ~50K msg/s
>100	db.r6g.2xlarge + Multi-AZ + Replicas 8 vCPU, 64GB 2TB+ io1 ~1000+ QPS capacity	cache.r6g.2xlarge + Cluster mode 8 vCPU, 52GB ~500K ops/s capacity	mq.m5.xlarge (HA) Multi-node cluster ~100K msg/s

10. Resumo Final

O que você precisa saber para avaliar o pricing:

1. **TPS do cliente → QPS na infra:** Multiplicador de ~11x (5 Redis + 6 PostgreSQL)
2. **Conversão rápida:** Txns/mês ÷ 2.592.000 × 3 = TPS de pico
3. **PostgreSQL é o componente mais caro** - É onde estão os dados e as queries mais pesadas
4. **Redis tem uso moderado** - Deduplicação e locking, mas volume não é crítico
5. **RabbitMQ tem uso BAIXO** - 1 mensagem por JOB, não por transação
6. **Custo variável por match é desprezível** - O que importa é o custo fixo
7. **Margem vem do volume** - Quanto mais transações no plano, mais diluído o custo fixo

Validação recomendada:

- [ ] Rodar infra real por 30 dias para validar custos
- [ ] Medir QPS real em ambiente de produção
- [ ] Ajustar sizing se uso for diferente do estimado

Changelog

Versão	Data	Alteração
1.0	Jan 2026	Versão inicial
1.1	Jan 2026	Adicionada seção TPS → QPS → Infraestrutura com tabela comparativa Midaz