“Radar de Saúde & Risco do Comerciante por UF”

• Consolida **saúde (A/B/C)** e **risco (LOW/MED/HIGH)** por **UF**.

• Mantém **Top-K merchants por valor** em cada UF.

• Mapeia **hotspots** (cidades com maior nº de merchants **HIGH**).

• Thresholds **parametrizáveis** via -D (sem recompilar).

**Fluxo (alto nível):**

**1 Job 1 (por merchant)**  
CSV → *(Mapper)* → (merchant\_id, TransactionMiniWritable)  
→ *(Reducer)* agrega por merchant, classifica Health/Risk, identifica UF/cidade predominantes  
→ **Emite 1 linha textual por merchant ancorada na UF predominante**

**2 Job 2 (por UF)**  
Saída textual do Job 1 → *(Mapper)* parseia e reconstrói StateMerchantAggWritable  
→ *(Combiner)* merge associativo local  
→ *(Reducer)* consolida UF: distribuições Health/Risk, Top-K merchants, hotspots

**1) MerchantHealthRisk.java (Driver)**

**Pacote:** routines.advanced.merchanthrisk  
**Papel:** Orquestra os **dois jobs**, define formatos de entrada/saída, classes de map/reduce, número de reducers e **defaults de parâmetros** (que podem ser sobrescritos com -D).

**O que faz**

• Lê argumentos: input\_csv, stage1\_out, final\_out, [num\_reducers], [local].

• (Opcional) habilita **modo local** (standalone) para testes rápidos.

• Define **defaults** (se não vierem por -D), ex.:

◦ health.revenue.med\_cents, health.revenue.high\_cents

◦ risk.error.med, risk.error.high, risk.online.med, risk.online.high, risk.max.high\_cents

◦ top.merchants.k, min.uf.merchants

• Configura e executa **Job 1** e, em seguida, **Job 2**.

**2) MerchantAggMapper.java (Mapper do Job 1)**

**Pacote:** routines.advanced.merchanthrisk  
**Papel:** Ler o **CSV** e **emitir um evento compacto** por transação, chaveado por merchant\_id.

**O que faz**

• Ignora cabeçalho.

• Faz split de CSV **respeitando aspas**.

• Extrai campos relevantes e normaliza:

◦ amount(4) → **centavos** (long)

◦ use\_chip(5) → **canal**: “ONLINE TRANSACTION” = isOnline=true; “SWIPE TRANSACTION” = false

◦ errors(11) → **flag de erro** (qualquer coisa ≠ vazio/None)

◦ merchant\_city(7), merchant\_state(8), mcc(10) → UPPERCASE, “UNKNOWN” se vazio

• Emite:

**◦ Key**: merchant\_id

**◦ Value**: TransactionMiniWritable(amountCents, isOnline, hasError, city, state, mcc)

**Entrada:** linhas CSV.  
**Saída (map output):** (merchant\_id, TransactionMiniWritable).

**3) TransactionMiniWritable.java (Writable de Evento)**

**Pacote:** routines.advanced.merchanthrisk  
**Papel:** Estruturar o **mínimo necessário** de uma transação para permitir métricas ricas no Reduce.

**O que faz**

• Define campos:

◦ amountCents (long)

◦ online (boolean)

◦ hasError (boolean)

◦ city, state, mcc (String)

• Implementa write/readFields (serialização Hadoop).

**Objetivo**

**• Compactar** e **transportar** informações de transações do Mapper para o Reducer do Job 1 de forma eficiente e tipada.

**Entrada:** — (usado internamente pelo MR).  
**Saída:** — (objeto serializado entre fases).

**4) MerchantAggReducer.java (Reducer do Job 1)**

**Pacote:** routines.advanced.merchanthrisk  
**Papel:** Agregar **todas** as transações de um merchant\_id, calcular métricas, **classificar saúde/risco** e **ancorar** o merchant na **UF/cidade predominantes**.

**O que faz**

• Soma: tx, sumCents, maxCents, errors, onlineTx.

• Conta ocorrências por state e por city → acha topState e topCity.

• Calcula métricas:

◦ errorRate = errors/tx

◦ onlineRate = onlineTx/tx

◦ avgCents = sumCents/tx

• Lê thresholds do Configuration:

**◦ Health**: health.revenue.med\_cents, health.revenue.high\_cents, health.avg.med\_cents, health.tx.min\_for\_avg

**◦ Risk**: risk.error.med/high, risk.online.med/high, risk.max.high\_cents

**• Classifica**:

◦ Health A/B/C (receita total e, opcionalmente, avg com volume mínimo)

◦ Risk LOW/MED/HIGH (erros, share online, pico)

**• Emite 1 linha textual por merchant** ancorada na **UF predominante**:

**◦ Key**: topState

**◦ Value**: "1:A:B:C|RL:RM:RH|[CITY=<topCity>=1|]MER=<merchant\_id>:<sumCents>"

**Objetivo**

• Converter um conjunto de transações por merchant em **1 linha de resumo** orientada a **UF**, pronta para ser somada pelo Job 2.

**Entrada (reduce input):** (merchant\_id, TransactionMiniWritable...)  
**Saída (reduce output):** (UF, linha\_textual\_resumida).

**5) StateMerchantAggWritable.java (Writable Agregado por UF – Job 2)**

**Pacote:** routines.advanced.merchanthrisk  
**Papel:** Agregar, por **UF**, as contagens de **Health** e **Risk**, o **mapa de hotspots** (cidades com merchants HIGH) e manter **Top-K merchants** por valor.

**O que faz**

• Mantém contadores:

◦ totalMerchants

◦ healthA/B/C

◦ riskLow/Med/High

• Mantém Map<String,Integer> highRiskCityCounts (cidade → nº merchants HIGH).

• Mantém **Top-K** (ids + vals) com métodos:

◦ addOneMerchant(...) (atualiza contadores/mapa e insere no Top-K)

◦ pushTop(...) (política de manter só os K maiores)

◦ merge(...) (soma contadores, **mescla mapas** e **intercala top-lists** mantendo K maiores)

◦ setK(k) para configurar K.

**Objetivo**

• Ser o **value** do Job 2 (map/combiner/reduce), **associativo** para permitir **combiner** seguro e consolidar tudo por UF.

**Entrada:** alimentado pelo StateAggMapper e mesclado no Combiner/Reducer.  
**Saída:** idem (objeto intermediário até a emissão final em texto).

**6) StateAggMapper.java (Mapper do Job 2)**

**Pacote:** routines.advanced.merchanthrisk  
**Papel:** Ler as **linhas textuais** do Job 1 e reconstruir um StateMerchantAggWritable por UF.

**O que faz**

• Cada linha vem como:  
UF \t 1:A:B:C|RL:RM:RH|[CITY=<city>=1|]MER=<merchant\_id>:<sumCents>

• Faz o **parse**:

◦ Health: deduz A/B/C a partir de 1:A:B:C

◦ Risk: deduz LOW/MED/HIGH a partir de RL:RM:RH

◦ City (se presente e risk=HIGH)

◦ Merchant e soma em centavos (para Top-K)

• Cria StateMerchantAggWritable, seta k (de top.merchants.k) e chama addOneMerchant(...).

• Emite:

**◦ Key**: UF

**◦ Value**: StateMerchantAggWritable preenchido com **1 merchant**.

**Objetivo**

• Converter a saída textual do Job 1 em um **value estruturado** agregável por UF, pronto para Combiner/Reducer.

**Entrada:** linhas do stage1\_out.  
**Saída (map output):** (UF, StateMerchantAggWritable).

**7) StateAggCombiner.java (Combiner do Job 2)**

**Pacote:** routines.advanced.merchanthrisk  
**Papel:** Fazer **merge local** (no nó do mapper) de múltiplos StateMerchantAggWritable da mesma UF.

**O que faz**

• Cria um acumulador StateMerchantAggWritable por UF.

• Para cada value recebido, chama acc.merge(v).

• Emite o **acumulado** por UF.

**Objetivo**

**• Reduzir tráfego** de rede somando contadores, mesclando **mapas de hotspots** e consolidando **Top-K** já no nó local, de forma **associativa** (segura).

**Entrada:** (UF, StateMerchantAggWritable...) do Mapper.  
**Saída:** (UF, StateMerchantAggWritable) pré-agregado.

**8) StateAggReducer.java (Reducer do Job 2)**

**Pacote:** routines.advanced.merchanthrisk  
**Papel:** Consolidar, por **UF**, **todas** as métricas e **emitir linha final legível** com percentuais, Top-K e hotspots.

**O que faz**

• Mergeia todos os StateMerchantAggWritable da mesma UF.

• Aplica supressão opcional de UF com poucos merchants (min.uf.merchants).

• Calcula **percentuais** para Health e Risk.

• Ordena e **exibe Top-K merchants** por valor (desc).

• Ordena e **exibe hotspots** (top cidades por merchants HIGH).

• Emite **1 linha por UF**:  
UF Merchants: T | Health A:x (p%) | B:y (p%) | C:z (p%) |

• Risk Low:rL (p%) | Med:rM (p%) | High:rH (p%) |

• Top Merchants: MER1:$... | MER2:$... | ...

• High-Risk Hotspots: CITY1: n | CITY2: m | ...

•

**Objetivo**

**• Fechar o pipeline** com uma saída imediatamente **acionável**: distribuição de saúde/risco por UF, **quem são os maiores merchants** e **onde estão os focos de risco**.

**Entrada:** (UF, StateMerchantAggWritable...) do Combiner/Mapper.  
**Saída:** (UF, Text) com a linha final.

perfeito — segue a **adição** pedida, no mesmo estilo dos seus docs, deixando explícito **o que coloca um merchant em Saúde A/B/C** e **Risco LOW/MED/HIGH**, com base nos *thresholds parametrizáveis* (podem ser alterados via -D no driver).

**Regras de Classificação**

**1) Saúde do Comerciante (Health A/B/C)**

**Objetivo:** refletir o “porte/robustez” do merchant pela **receita agregada** (e opcionalmente ticket médio com volume mínimo).

**Parâmetros (podem ser sobrescritos via -D):**

• health.revenue.high\_cents *(default: 2000000)* → limiar para **A** (receita alta)

• health.revenue.med\_cents *(default: 500000)* → limiar para **B** (receita média)

• health.avg.med\_cents *(default: 8000)* → limiar opcional de **avg** para consolidar B

• health.tx.min\_for\_avg *(default: 100)* → nº mínimo de transações para considerar o **avg**

**Cálculos usados:**

• sumCents = soma de valores em centavos do merchant

• tx = número de transações

• avgCents = sumCents / tx (apenas interpretado se tx >= health.tx.min\_for\_avg)

**Regras (em ordem):**

**1 Health A**  
Se sumCents >= health.revenue.high\_cents  
Interpretação: alto volume financeiro agregado — merchant “grande”.

**2 Health B**

◦ Se **não** for A **e** sumCents >= health.revenue.med\_cents  
**OU**

◦ (Opcional) Se **não** for A e tx >= health.tx.min\_for\_avg **e** avgCents >= health.avg.med\_cents

3 Interpretação: receita total consistente **ou** ticket médio robusto já com volume razoável.

**4 Health C**  
Caso contrário.  
Interpretação: baixo porte/participação (até aqui, sem juízo de *risco*).

**Notas:**

• Você pode usar **só receita** (A/B/C por sumCents) ou reforçar B com o **avg** quando houver tx suficiente (evita classificar por avg com 2–3 vendas grandes).

• Ajuste health.tx.min\_for\_avg e health.avg.med\_cents conforme sazonalidade ou setor (ex.: categorias com tíquete naturalmente mais alto).

**2) Risco do Comerciante (LOW/MED/HIGH)**

**Objetivo:** capturar **probabilidade/gravidade de problemas** operacionais e de fraude/chargeback a partir de **erros, canal online** e **picos**.

**Parâmetros (via -D):**

• risk.error.high *(default: 0.05)* → taxa de erro para **HIGH**

• risk.error.med *(default: 0.02)* → taxa de erro para **MED**

• risk.online.high *(default: 0.90)* → share de transações **online** para **HIGH**

• risk.online.med *(default: 0.70)* → share **online** para **MED**

• risk.max.high\_cents *(default: 500000)* → **pico** (maior valor) para **HIGH**

**Cálculos usados:**

• errorRate = errors / tx

• onlineRate = onlineTx / tx *(“ONLINE TRANSACTION” conta; “SWIPE TRANSACTION” não)*

• maxCents = maior valor de transação

**Regras (em ordem):**

**1 HIGH** se **qualquer** condição abaixo for verdadeira:

◦ errorRate >= risk.error.high **e** onlineRate >= risk.online.med  
*(erro alto combinado com canal online significativo)*

◦ onlineRate >= risk.online.high  
*(predominância forte de canal online)*

◦ maxCents >= risk.max.high\_cents  
*(pico muito alto — exposição elevada)*

2 Interpretação: perfil de maior exposição (fraude, chargeback, instabilidade).

**3 MED** se **não for HIGH** e **qualquer** condição abaixo:

◦ errorRate >= risk.error.med

◦ onlineRate >= risk.online.med

4 Interpretação: sinais relevantes, porém sem ultrapassar os gatilhos de HIGH.

**5 LOW** caso contrário.  
Interpretação: ausência de sinais de risco relevantes na janela observada.