6 - RecipeMonitorService

Iagro CLP Local — RecipeMonitorService

Serviço de **captura** de receitas ERP em status `NA_FILA` para uma SmartCalda e delegação ao **5 - RecipeProcessor**. Não executa OPC nem altera `write_commands` diretamente.

Código: `src/services/recipe_monitor_service.py`

> **Info**
> Bloqueio de nova captura: `DatabaseManager.has_pending_recipe_commands` — semântica restrita (“trabalho real”). Pai `executing` órfão **não** bloqueia; ver **7 - CommandProcessor** (`sanitize_orphan_create_recipes`).

📋 Resumo

Aspecto	Detalhe
Modo thread	`threading.Thread` quando `start_thread=True` (paralelo)
Modo sequencial	`SequentialCoordinator._task_monitor_recipes`
Polling adaptativo	2 s após sucesso; 10 s após 60 s sem captura
Limite API	50 (thread) / 1 (coordenador) por ciclo

🎯 Responsabilidades

1. Manter `RecipeProcessor` sincronizado com variáveis OPC cadastradas e `recipe_mapping` .
2. Consultar API apenas quando **todos** os gates passam.
3. Ordenar candidatos por `createdAt` (FIFO).
4. Chamar `process_recipe` até primeira aceitação ou esgotar lista.

📁 API pública

Método	Efeito
`__init__(…, start_thread=True)`	`_update_processor()`; opcionalmente `start()` thread
`run()`	Loop infinito com gates + sleep
`stop()`	`_running = False`
`is_running()`	Estado thread
`update_container_variables(list)`	Atualiza vars + `_update_processor`
`update_configuration(json)`	Novo `recipe_mapping`
`_update_processor()`	`RecipeProcessor(...)` novo

Parâmetros notáveis do construtor:

- `production_monitor_callback` — tipicamente `lambda: production_monitor.has_pending_reset()`
- `is_connected_callback` — `container.is_connected`

📁 Contratos de dados

Entrada API (`get_work_order_recipe_sync`)

Campos consumidos no loop:

Campo	Uso
`id`	Dedup / active check
`parcelId`	Log / value pai
`recipeData`	Mapper
`status`	Deve ser `NA_FILA` no filtro
`createdAt`	Ordenação ASC

Saída

- `process_recipe` → `True`: intervalo **rápido** (reconsulta em 2 s).
- `False` para todas: mantém ciclo; após timeout adaptativo → intervalo **lento**.

🔄 Gates (ordem de avaliação)

Todos devem ser verdadeiros para chamar a API:

#	Gate	Implementação	Se falhar
1	CLP online	`is_connected_callback()`	Log debug; sleep
2	DB online	`database_manager.is_connected()`	idem
3	Fila livre	`not has_pending_recipe_commands(container_id)`	idem
4	Sem receita ativa ERP	`get_active_recipe` vazio ou mesmo id	idem
5	Sem reset pendente PM	`production_monitor_callback()` False	idem

Semântica gate 3 (`has_pending_recipe_commands`)

Conta **> 0** se existir:

- `create-recipe` **pending**
- `recipe-finalize` \| `reset-cancel-recipe` \| `reset-defaults` em **pending\|executing**
- Write filho **pending\|executing** com pai `create-recipe` **executing**

Não conta:

- Pai `executing`, todos filhos `completed`, sem trabalho pendente (órfão).

Semântica gate 4

`get_active_recipe` retorna receita em `NA_MAQUINA`, `EM_PRODUCAO` ou `PAUSADO`. Se id ≠ candidato `NA_FILA` → não enfileira segunda receita.

Semântica gate 5

Evita capturar nova receita enquanto `ProductionMonitorService` tem comando reset pendente na fila (`has_pending_reset`).

🔄 Loop detalhado (thread `run`)

```
_running = True
last_capture = now
interval = FAST

while _running:
  if gates_fail:
    sleep(interval); continue

  recipes = API.get_work_order_recipe_sync(NA_FILA, limit=50)
  sort recipes by createdAt ASC
  captured = False

  for recipe in recipes:
    if process_recipe(recipe, is_connected):
      captured = True
      last_capture = now
      interval = FAST
      break

  if not captured:
    if (now - last_capture) > ADAPTIVE_TIMEOUT:
      interval = SLOW
  sleep(interval)
```

Modo sequencial (`_task_monitor_recipes`)

Equivalente lógico com:

- `limit=1` na API (menos carga, uma tentativa por tick do coordenador).
- Intervalos vindos de config do coordenador (`RECIPE_POLL_*`).
- Sem thread RM — `start_thread=False` em `start_recipe_monitoring` .

🎯 Integração

Componente	Relação
`CLPContainer.start_recipe_monitoring`	Cria RM; `start_thread=not sequential_mode`
`RecipeProcessor`	Recriado em `_update_processor`
`CommandProcessor`	Consome fila criada após captura
`ProductionMonitorService`	Callback gate 5

`SequentialCoordinator`	Substitui thread RM

⚙️ Configuração

Constante / env	Padrão	Efeito
`RECIPE_POLLING_FAST_MS`	2000	Pós-captura
`RECIPE_POLLING_SLOW_MS`	10000	Ociosidade prolongada
`RECIPE_ADAPTIVE_TIMEOUT`	60 s	Limiar FAST→SLOW

⚠️ Cenários operacionais

Cenário	Comportamento
API retorna `[]`	Nenhum `process_recipe`; timeout adaptativo
Primeira receita False, segunda True	Processa segunda no mesmo ciclo (limit 50)
Órfão pai executing	Gate 3 libera; CP sanitizer pode fechar ciclo
Reset PLC pendente	Gate 5 bloqueia até PM drenar fila
Parada app	`stop()`; coordenador deixa de agendar tarefa

🔗 Referências

- **5 - RecipeProcessor** · **7 - CommandProcessor** (`has_pending`, sanitizer)
- **8 - ProductionMonitorService** (`has_pending_reset`)
- **4 - ApiService** · **9 - CLPContainer** · **10 - SequentialCoordinator**