7 - CommandProcessor

Iagro CLP Local — CommandProcessor

Worker que consome `write_commands` no PostgreSQL, executa escrita/descoberta/reset no CLP via OPC UA e reconcilia o ciclo de vida do comando pai `create-recipe` (writes iniciais, reset pós-produção, finalização supervisória).

Código: `src/services/command_processor.py` (~1117 linhas)
Persistência: `src/core/database.py` (`get_pending_write_commands`, `has_pending_recipe_commands`, `create_recipe_command`, ...)
Schema: `config/schema.sql` — tabela `write_commands`
Espelho repo: `docs/services/command_processor.md`

> **Info — Fonte de verdade**
> Comportamento descrito a partir do código em execução, não de fluxos legados. O backend insere `recipe-finalize` após `confirm-action`; o CLP enfileira reset no pai com `skip_api_cancel=True` (ERP já foi atualizado no supervisório).

📋 Resumo executivo

Papel	Detalhe
Execução imediata	`write`, `discover`, `create-recipe` (loop interno de filhos)
Execução diferida / reconciliação	`check_executing_recipe_parents`, `sanitize_orphan_create_recipes`
Resets	`reset-defaults`, `reset-cancel-recipe`, `recipe-finalize` → `finalize_create_recipe_with_reset`

Isolamento de fila	Writes `recipe_*` **nunca** entram em `get_pending_write_commands`
ERP após writes OK	`update_parcel_status(..., NA_MAQUINA)`; falha → `BLOQUEADO` + `reset-defaults`

🗄️ Modelo `write_commands`

Colunas relevantes

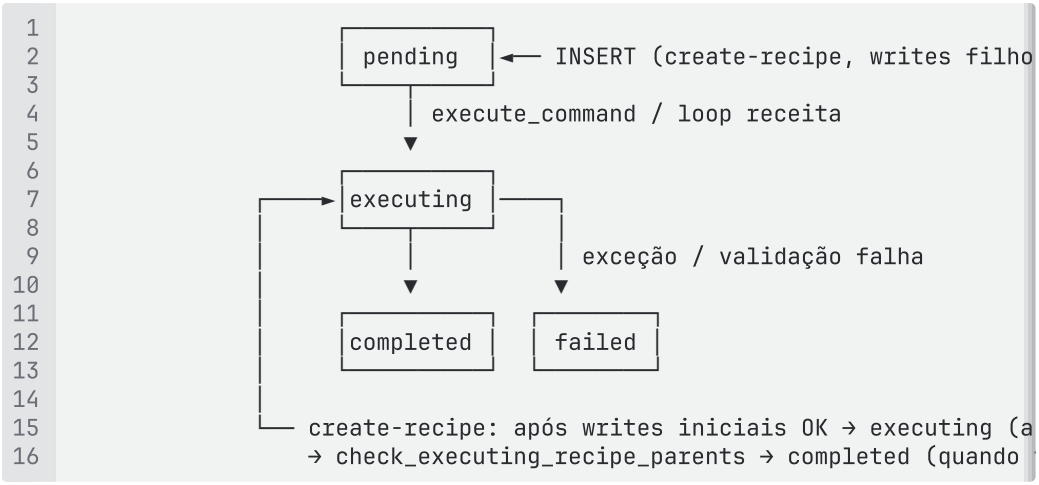
Coluna	Tipo	Semântica
`id`	serial	PK
`command_type`	varchar	`write`, `discover`, `create-recipe`, `reset-defaults`, `reset-cancel-recipe`, `recipe-finalize`
`container_id`	uuid	FK `containers`
`variable_id`	int4	Obrigatório para `write`; **NULL** para tipos "pai"
`value`	jsonb	Payload (receita, sync id, ação supervisório, valor OPC)
`status`	varchar	Ciclo de vida (ver máquina de estados)
`recipe_command_id`	int4	FK self — filhos apontam para pai `create-recipe` / reset pai

`order`	int4	Ordem de execução dos writes filhos
`requested_by`	varchar	Rastreio + filtro de fila (`reset_after_production_{syncId}`, `recipe_blocked_fail_{id}`, ...)
`executed_at`	timestamp	Preenchido ao terminal
`error_message`	text	Falha persistida

Constraint `write_commands_command_type_variable_check`

- `write` → `variable_id IS NOT NULL`
- `create-recipe`, `reset-*`, `recipe-finalize`, `discover` → `container_id IS NOT NULL`; tipos pai → `variable_id IS NULL`

Máquina de estados (por linha)



Invariante crítica: `NA_MAQUINA` no ERP só após **todos** writes iniciais (`requested_by` **não** `reset_after_production_*`) com `status = completed`. O pai `create-recipe` permanece `executing` até resets pós-produção terminarem.

🎯 Responsabilidades por `command_type`

Tipo	Handler	Efeito
`write`	`_execute_write_command`	Leitura tag OPC + `write_value` +

		`update_command_status(completed|failed)`
`discover`	`_execute_discover_command`	Descoberta de variáveis no CLP; loading UI `discover`
`create-recipe`	`_execute_recipe_command`	Loop sequencial de filhos; ERP `NA_MAQUINA` ou `_fail_recipe_write`
`reset-defaults`	`_execute_reset_defaults_command`	Reset tags; opcional vínculo ao pai via `_resolve_create_recipe_for_reset`
`reset-cancel-recipe`	idem + `cancel_active_recipe=True`	Cancela receita ativa na API + reset no pai aberto
`recipe-finalize`	`_execute_recipe_finalize_command`	`finalize_create_recipe_with_reset(..., skip_api_cancel=True)`

📁 API pública — classe `CommandProcessor`

Método	Comportamento
`__init__(container, database_manager, interval_ms)`	Thread daemon; callbacks opcionais
`run()`	`while running: process_commands(); sleep(interval_ms)`

`stop()`	`_running=False`; `join(COMMAND_PROCESSOR_JOIN_TIMEOUT_S)`
`process_commands()`	`check_executing_recipe_parents` → batch `get_pending` → `execute_command` cada um
`execute_command(command)`	Gate conexão; `executing`; dispatch; exceção → `failed`
`set_interval(ms)`	Ajuste dinâmico do poll

Callbacks: `on_command_received`, `on_command_executed`, `on_error` (também usado para logs de receita na UI).

📁 Funções de módulo (reconciliação sem thread)

`check_executing_recipe_parents(container, processor?)`

Consulta todos `create-recipe` com `status = executing` do container.

Para cada pai `recipe_id`:

Condição	Ação
∃ filho `pending` ou `executing`	`continue` (aguarda worker)
∃ filho `failed`	`_fail_recipe_write` / `_fail_recipe_write_impl` → pai `failed`, ERP `BLOQUEADO`, `reset-defaults`
Sem filhos `reset_after_production_*`	`continue` (writes iniciais OK; aguarda finalize/reset ser criado)
Com resets e `completed_count == total_count`	`update_command_status(pai, completed)`
Inconsistência de contagem	`continue` (log debug)

Chamado no início de **cada** `process_commands` e pelo `SequentialCoordinator` após reconexão.

`sanitize_orphan_create_recipes(container, processor?)`

Recuperação legado: pai `executing`, **todos** filhos terminalizados, **sem** linhas `reset_after_production_*`, ERP já `CANCELADO` ou `PROCESSADA`.

1. `get_recipe_sync_by_id` → status terminal
2. `finalize_create_recipe_with_reset(..., source=orphan_sanitizer, skip_api_cancel=True)`
3. Resets entram na fila; `check_executing_*` fecha o pai

Não altera `has_pending_recipe_commands` para o caso órfão puro (sem filhos pendentes).

`finalize_create_recipe_with_reset`

```
_get_production_monitor_for_container
  → (opcional) cancel_active_recipe_if_any se cancel_active_recipe e não
  → pm.reset_plc_to_defaults(
        recipe_command_id = create_recipe_id,
        requested_by_override = "reset_after_production_{recipe_sync_id}
    )
```

Writes filhos de reset passam a ser elegíveis em `get_pending_write_commands` (prefixo reconhecido na SQL).

`_resolve_create_recipe_for_reset`

`find_open_create_recipe(container_id, recipe_sync_id)` → id do pai; fallback `recipe_sync_id` no `value` do comando reset.

🔄 `_execute_recipe_command` — algoritmo detalhado

Fase 0 — Parse `value`

Extrai `recipe_id` (PK `work_order_recipe_plc_sync`), `parcel_id`, `recipe_data` de dict ou JSON string. Log UI `log.recipe_found`.

Fase 1 — Primeira passagem de writes

1. `get_recipe_write_commands(command_id)` ordenado por `"order"`.
   - Vazio → exceção → `_fail_recipe_write`.
2. `get_variables_by_ids_batch` — cache OPC (evita N+1).
3. `emit_loading(start, recipe_write, persist=True)`.
4. **Loop** `for write_cmd in write_commands`:
   - Se `not container.is_connected()` → marca restantes `failed`, `container_disconnected=True`, **break**.

- `_execute_write_command(write_cmd, var_data_cache)` — sucesso incrementa `success_count`; falha marca filho `failed` mas **continua** próximos.
- Loading `update` com `current/total`.

Fase 2 — Segunda passagem (isolamento transacional)

`get_recipe_write_commands` novamente; executa filhos `pending` cujo `id ∉ processed_ids` (comandos invisíveis na primeira leitura).

Fase 3 — Limpeza de pendentes órfãos

Filhos ainda `pending` e não processados → `failed` com mensagem fixa ("não executado antes da conclusão...").

Fase 4 — Validação de tentativa

Condição	Resultado
`container_disconnected`	`_fail_recipe_write` (sem NA_MAQUINA)
`total_attempted < total_commands`	`_fail_recipe_write`
Writes iniciais com `status != completed`	`_fail_recipe_write` com detalhes ids
Writes `executing` na validação (corrida fila global antiga)	**Recuperação:** reexecuta `_execute_write_command` no fluxo da receita se CLP online

Fase 5 — Sucesso parcial ERP / pai executing

1. `update_command_status(command_id, 'executing')` — **não** `completed` ainda.
2. `update_parcel_status(recipe_id, 'NA_MAQUINA')` via `ApiService`.
3. `_stop_recipe_loading` + `on_command_executed(True)`.
4. Resets pós-produção e `check_executing_recipe_parents` fecham o pai depois.

Separação semântica: ERP "na máquina" ≠ comando DB `completed` (este exige resets supervisório/legado).

🔄 `execute_command` — gates e dispatch

```
SE command_type IN (reset-defaults, reset-cancel-recipe, recipe-finaliz
   SE NOT container.is_connected():
```

```
      RETURN sem alterar status (permanece pending — retry)
SENÃO:
   SE NOT container.is_connected():
      raise → failed

update_command_status(id, executing)

SWITCH command_type:
  write → _execute_write_command
  discover → _execute_discover_command
  create-recipe → _execute_recipe_command
  reset-defaults → _execute_reset_defaults_command
  reset-cancel-recipe → _execute_reset_defaults_command(cancel_active_r
  recipe-finalize → _execute_recipe_finalize_command
  default → failed (tipo desconhecido)
```

🔄 Fluxo supervisório (ERP terminal → CLP reset → pai completed)

```
View: confirm-action (cancel | complete)
  → Backend: write variável fixa + poll write_commands até completed
  → Backend: ERP status CANCELADO | PROCESSADA
  → Backend: INSERT recipe-finalize (value: recipe_sync_id, action)
  → CLP poll: execute_command(recipe-finalize)
  → finalize_create_recipe_with_reset(create_id, skip_api_cancel=True)
  → N writes filhos reset_after_production_{syncId} (recipe_command_id
  → get_pending processa resets na fila global
  → check_executing_recipe_parents: todos completed → create-recipe com
  → has_pending_recipe_commands = false → RecipeMonitor pode capturar N
```

🔄 `get_pending_write_commands` — semântica SQL

Objetivo: evitar corrida em que a fila global marca write `recipe_*` como `executing` enquanto `_execute_recipe_command` ainda roda (estado “preso”, receita bloqueada).

Inclui `pending` quando:

Ramo	Significado
`recipe_command_id IS NULL`	Comandos raiz independentes
`command_type = create-recipe`	Novo lote de receita
`command_type IN (reset-defaults, reset-cancel-recipe, recipe-finalize)`	Resets e finalize
`write` + pai em `executing` OU pai `completed` com `executed_at > now()-5min`	Janela para resets tardios
`requested_by` LIKE `reset_after_production_%`	Apenas resets na fila global

OR `reset_default_cmd_%`	

Exclui implicitamente: writes filhos com `requested_by` de batch `recipe` (sem prefixos acima).

ORDER BY: tipos pai (prioridade 1) → `created_at ASC` → `LIMIT`.

🔄 `has_pending_recipe_commands` — bloqueio `NA_FILA`

Soma de três contagens (por `container_id` opcional):

1. `create-recipe` **pending**
2. `recipe-finalize` \| `reset-cancel-recipe` \| `reset-defaults` em **pending\|executing**
3. Filhos **pending\|executing** de pai `create-recipe` **executing**

Cenário	Bloqueia?
Pai `executing`, filhos todos `completed`, sem resets	**Não** (órfão — sanitizer)
`recipe-finalize` pending	**Sim**
Pai `pending`	**Sim**
Filho reset `pending`, pai `executing`	**Sim** (contagem 3)

Implementação: `database.py` ~L1802.

🔄 `_fail_recipe_write` / `_fail_recipe_write_impl`

```
update_command_status(create_id, failed, error_msg)
update_parcel_status(sync_id, BLOQUEADO)
on_command_executed(False)
create_reset_defaults_command(requested_by=recipe_blocked_fail_{create_i
```

`enqueue_reset_defaults=False` usado em caminhos que já enfileiram reset explícito.

🎯 Integração e modos de execução

Origem	Modo
`CLPContainer.start_command_processing`	Instancia CP; `_sync_command_processor_thr`

	`ead` liga thread se monitor OFF
`SequentialCoordinator._task_process_commands`	Chama `process_commands` sem thread CP quando monitor ON
`ProductionMonitorService.reset_plc_to_defaults`	Mesma origem de writes de reset que finalize
UI `container.emit_loading`	Tipos `recipe_write`, `discover`; `persist=True` em receita

⚙️ Configuração

Variável / constante	Padrão	Uso
`COMMAND_POLL_INTERVAL_MS`	via container	Sleep em `run()`
`COMMAND_PROCESSOR_JOIN_TIMEOUT_S`	8 (env)	Join ao parar
`get_pending_write_commands` limit	100 em `process_commands`	Tamanho do batch

⚠️ Matriz de falhas e recuperação

Cenário	Comportamento
CLP offline em reset/finalize	Status **permanece** `pending`
CLP offline durante loop create-recipe	Filhos restantes `failed`; pai `_fail_recipe_write`
Um write filho falha no loop	Continua outros; validação final falha pai
`recipe-finalize` sem pai aberto	Warning; comando pode `completed` sem reset
Pai `executing` + ERP terminal + sem resets	`sanitize_orphan_create_recipes`

Write `executing` na validação final	Reexecução no fluxo receita (anti-corrida)
Sem PM e sem `configurationJson`	Exceção em `_get_production_monitor_for_container`

🔗 Referências cruzadas

- **5 - RecipeProcessor** — cria pai + filhos `recipe_*`
- **6 - RecipeMonitorService** — gates + `has_pending`
- **8 - ProductionMonitorService** — `reset_plc_to_defaults`, feedback OPC (end/cancel comentados)
- **9 - CLPContainer** · **10 - SequentialCoordinator**
- **4 - ApiService** — `update_parcel_status`, `get_recipe_sync_by_id`
- **12 - API — contrato OpenAPI** — `recipe-finalize`, confirm-action