8 - ProductionMonitorService

Iagro CLP Local — ProductionMonitorService

Camada de **feedback máquina → ERP**: interpreta mudanças de variáveis OPC (via `CLPContainer._on_data_updated`) usando `configurationJson.production_monitor` e dispara PATCH de status, sequência de processo e enfileiramento de **resets** no PostgreSQL.

Código: `src/services/production_monitor_service.py` (~1560 linhas)

> **Info — Supervisório vs CLP**
> Em `on_variable_updated` (~L316–324), `_check_end_production` e `_check_cancel_production` estão **comentados**.
> Conclusão/cancelamento terminal passam pelo backend (`confirm-action`) + `recipe-finalize` (**7 - CommandProcessor**). O PM continua ativo para **início**, **pausa/retomada**, **falha** e **process_sequence**.

📋 **Resumo**

Mecanismo	Detalhe
Entrada	`on_variable_updated(container_id, values)` — dict nome→valor
Cache receita	`get_active_recipe` TTL ~10 s → `current_active_recipe_id`
Debounce status	2 s (`_should_skip_update`)
Debounce sequência	1 s
Reset	`_reset_plc_parameters` → INSERT writes filhos vinculados ao pai
Bloqueio captura	`has_pending_reset()` → gate **6 - RecipeMonitor**

🎯 Mapa `production_monitor` (config)

Chave	Handler	Status ERP	Ativo no loop?
`start_production`	`_check_start_production`	`EM_PRODUCAO`	Sim
`end_production`	`_check_end_production`	`PROCESSADA` + timer reset	**Não** (comentado)
`pause_production`	`_check_pause_production`	`PAUSADO`	Sim
`resume_production`	`_check_resume_production`	`EM_PRODUCAO`	Sim
`cancel_production`	`_check_cancel_production`	`CANCELADO`	**Não** (comentado)
`fail_production`	`_check_fail_production`	conforme config	Sim
`process_sequence`	`_check_process_sequence`	PATCH `process-sequence`	Sim
`recipe_cancell`	`_build_products_used_payload`	payload cancel	Indireto

Modelo de trigger (`_check_trigger`)

`trigger_type`	Comportamento
`boolean`	Compara valor normalizado bool
`numeric`	Igualdade / limiar conforme config
`string`	Match de string
`edge`	Detecção de borda com `previous_values`

Variáveis monitoradas: apenas nomes referenciados nas configs (não todas as tags do container).

📁 API pública

Método	Contrato
`on_variable_updated(values)`	Pré: `_running` e CLP conectado
`has_pending_reset()`	Timer reset ativo **OU** `has_pending_reset_commands(smart_calda_id)`
`reset_plc_to_defaults(recipe_command_id?, requested_by_override?)`	Delega `_reset_plc_parameters`
`cancel_active_recipe_if_any()`	PATCH `CANCELADO` na receita ativa — `reset-cancel-recipe`
`start()` / `stop()` / `is_running()`	Lifecycle (sem thread de poll)
`get_diagnostic_info()`	Config + receita + timers (suporte)

🔄 Pipeline `on_variable_updated`

```
1. Guard: _running, is_connected_callback
2. _update_active_recipe()  // cache API
3. _check_start_production(values)
4. [SKIP] _check_end_production
5. _check_pause_production / _check_resume_production
6. [SKIP] _check_cancel_production
7. _check_fail_production(values)
8. _check_process_sequence(values)
9. Atualizar previous_values só para vars das configs
```

`_update_recipe_status`

```
SE NOT _is_valid_status_transition(atual, novo): log; RETURN
SE _should_skip_update (debounce): RETURN
api_service.update_parcel_status(sync_id, novo, branch_id)
```

Transições inválidas são ignoradas (evita regressão ERP por ruído OPC).

🔄 Reset PLC — `_reset_plc_parameters`

Pré-condição: `database_manager.is_connected()`.

Tag `requested_by`:

- `requested_by_override` se passado (ex.: `reset_after_production_{syncId}` do finalize).
- Senão `reset_after_production_{current_active_recipe_id}`.

Etapas:

1. **Controles** — tags de `start/end/pause/cancel/fail` → write `False` (`create_write_command` com `recipe_command_id`).
2. **Receita*** — variáveis mapeadas em `recipe_mapping` → valores padrão da config.
3. Filhos ficam `pending`; **7 - CommandProcessor** executa via `get_pending_write_commands`.

Consumidores do mesmo código:

Chamador	`recipe_command_id`	`skip_api_cancel`
`finalize_create_recipe_with_reset`	id `create-recipe`	N/A (não cancela API)
`reset-defaults` / `reset-cancel-recipe`	pai resolvido ou None	cancel só em reset-cancel
`_check_end_production` (se reativado)	timer `_execute_delayed_reset`	—

🔄 `has_pending_reset` — impacto na fila

```
return (
  _reset_timer is not None and not expired
  OR database_manager.has_pending_reset_commands(smart_calda_id)
)
```

Enquanto True, **RecipeMonitor** não consulta `NA_FILA` (gate 5).

🎯 Integração

Componente	Ligação

`CLPContainer`	Instancia após connect + `configurationJson`; `_on_data_updated`
`CommandProcessor`	`_get_production_monitor_for_container` — PM lazy se container sem instância
`RecipeMonitorService`	callback `has_pending_reset`
`ApiService`	PATCH status / process-sequence

⚙️ Configuração

Carregada em `_load_configuration` a partir de `SmartCalda.configurationJson.production_monitor`:

- `variable_name` por gatilho
- `trigger_value`, `trigger_type`
- Debounces internos por tipo de evento

Sem `configurationJson` válido, resets iniciados pelo CP falham com exceção explícita.

⚠️ Matriz operacional

Cenário	Resultado
Sem receita ativa	Checks de produção retornam cedo
end/cancel CLP desligados	Terminal só supervisório + finalize
DB offline no reset	Log warning; sem INSERT
Transição ERP inválida	Sem PATCH
Debounce 2 s	Ignora repetição mesmo status
Reativar end no CLP	Descomentar L318–319 e L323–324 — **conflita** com fluxo supervisório se ambos ativos

🔗 Referências

- **7 - CommandProcessor** (`finalize`, fila reset)

- **6 - RecipeMonitorService** (gate)
- **4 - ApiService** · **9 - CLPContainer**