9 - CLPContainer

Iagro CLP Local — CLPContainer

Agregado de runtime **por SmartCalda**: metadados API, duas conexões OPC UA, `DatabaseManager`, serviços de receita/comando/produção e ponte para UI.

Código: `src/core/clp_container.py`

📋 Resumo

Responsabilidade	Detalhe
Conexão	`OPCConnection` monitor + `connection_realtime` — ambas obrigatórias para `is_connected()`
Fila	Delega a `CommandProcessor` + PostgreSQL
Modo leitura	Sequencial (`SequentialCoordinator`) ou `VariableMonitorThread`
UI	`on_data_updated`, `emit_loading`, `ui_activity_logs`

📁 Modelo de estado

Campo / API	Semântica
`status`	`connected`, `disconnected`, `error`
`is_connected()`	**AND** das duas sessões OPC
`is_monitoring()`	Coordenador `is_running()` **ou** `monitor_thread.is_alive()`

```
[disconnected] --connect OK--> [connected]
[connected] --start_monitoring--> [monitoring ativo]
```

```
[monitoring] --stop_monitoring--> [connected] (serviços podem realinhar
[*] --disconnect / mark_connection_lost --> [disconnected]
```

🔄 `connect()` — sequência

1. Abrir OPC monitor + realtime.
2. `status=connected`, evento UI, persistência DB.
3. Pós-conexão (ordem típica):
   - `start_command_processing(start_thread=False)` se sequencial
   - `start_recipe_monitoring(start_thread=False)` idem
   - `start_production_monitoring` (callback OPC)
   - `start_monitoring` se existem variáveis
4. `_auto_reconnect_worker` se flag habilitada.

Falha parcial OPC → `error` / disconnect controlado.

🔄 `disconnect()` / `mark_connection_lost`

1. Desabilita auto-reconnect; encerra worker.
2. `_stop_plc_services_and_opcua` — cada serviço em try/except isolado:
   - para coordenador / monitor thread
   - `command_processor.stop()`
   - `recipe_monitor.stop()`
   - `production_monitor.stop()`
   - fecha subscriptions OPC
3. `status=disconnected`, salva DB.

Comandos `reset-*` / `recipe-finalize` pendentes **permanecem** `pending` para retry na reconexão.

🏗️ `_sync_command_processor_thread`

`is_monitoring()` (sequencial ON)	Ação
True	Para thread CP se rodando — poll só via coordenador
False	Inicia `CommandProcessor.start()` se instância existe e parada

Evita **dois** polls simultâneos em `write_commands`.

Chamado em: `start_monitoring`, `stop_monitoring`, transições de modo.

🎯 Serviços — matriz de lifecycle

Serviço	Criado em	Thread própria (sequencial + monitor ON)
`CommandProcessor`	`start_command_processing`	**Não** — `process_commands` via **10**
`RecipeMonitorService`	`start_recipe_monitoring`	**Não** — `_task_monitor_recipes`
`ProductionMonitorService`	`start_production_monitoring`	Nunca (callback)
`SequentialCoordinator`	`start_monitoring` (sequencial)	`read_worker` + `serial_worker`

`set_api_service(api)` — propaga token para reinicialização de monitors quando necessário.

🔄 Pipeline de dados OPC

```
OPC read (coordenador ou VariableMonitorThread)
  → _detect_changes / buffer
  → _on_data_updated
       ├→ UI scheduler (on_data_updated)
       └→ production_monitor.on_variable_updated
```

`emit_loading(event, kind, message, persist?, current?, total?)` — usado por CP (receita, discover) e refletido em logs UI.

⚙️ Variáveis e performance

Método	Função
`load_variables_from_database`	Hidrata `variables` do container

`variable_lookup`	Dict nome → metadata O(1)
`_try_start_opcua_subscription`	Snapshot + subscription monitor
`apply_performance_intervals`	Repassa env ao coordenador

🎯 Integração UI

Callback	Consumidor
`on_data_updated`	Cards de variáveis, supervisório
`on_connection_changed`	Status sidebar
`emit_loading`	`log_loading` / indicador progresso

⚠️ Falhas comuns

Sintoma	Causa provável
Comandos não executam	`is_connected()` False (só uma sessão OPC)
Duplo poll DB	CP thread + coordenador sem `_sync`
PM silencioso	`_running` False ou desconectado
NA_FILA não captura	Gates **6** — ver página 6

🔗 Referências

- **10 - SequentialCoordinator**
- **2 - Services — Visão geral** · **7 - CommandProcessor**
- **11 - UI — IAgroCLPAppCTk**