2 - Services — Visão geral

Iagro CLP Local — Services

Camada `src/services/` entre **backend HTTP** (NestJS), **PostgreSQL** (`write_commands` , `variables` , `containers`) e **CLP** (OPC UA via `CLPContainer`). O orquestrador de runtime é **9 - CLPContainer**; o agendador opcional é **10 - SequentialCoordinator**.

Código: `src/services/` · **Espelho:** `docs/services/README.md`

📋 Mapa de módulos

Módulo	Thread própria?	Persistência	Efeito externo
AuthService	Não	—	Token Bearer + tenant
ApiService	Não	—	HTTP ERP
RecipeMonitorService	Opcional	—	Poll `NA_FILA`
RecipeProcessor	Não	INSERT `write_commands`	Enfileira receita
CommandProcessor	Opcional	UPDATE `write_commands`	OPC + ERP status
ProductionMonitorService	Callback OPC	INSERT resets	ERP transições + reset CLP

🏗️ Arquitetura de runtime

```
[IAgroCLPAppCTk / main.py]
        |
        +-- AuthService ──► ApiService (headers Bearer, X-Tenant-ID, X-
        |
        v
   [ContainerManager] ──► N × [CLPContainer]
        |
        +-- OPCConnection ──────────────► [CLP / OPC UA]
        +-- DatabaseManager ────────────► [PostgreSQL]
        |
        +-- [SequentialCoordinator]?  (se monitor de receita ON)
        |       ├─ _task_process_commands ──► CommandProcessor.process_
```

```
|       ├─ _task_monitor_recipes ──► RecipeMonitor poll
|       └─ _task_read_variables ──► leituras OPC agendadas
|
+-- [CommandProcessor thread]?     (se monitor OFF)
+-- [RecipeMonitorService thread]? (se monitor OFF)
+-- [ProductionMonitorService]       (subscrições OPC → ERP)
```

Regra de exclusão: com monitor ON, CP e RM **não** iniciam thread própria; o coordenador serializa tarefas para evitar corrida OPC/DB.

🔄 Ciclo de vida receita (ERP ↔ fila ↔ CLP)

Estados ERP (`work_order_recipe_plc_sync.status`)

Status	Quem define	Pré-condição típica
`NA_FILA`	ERP / operador	Aguardando calda
`NA_MAQUINA`	**CommandProcessor**	Todos writes iniciais `completed`
`EM_PRODUCAO` / `PAUSADO`	ProductionMonitor (gatilhos OPC)	Feedback máquina
`PROCESSADA` / `CANCELADO`	Supervisório (backend)	`confirm-action`
`BLOQUEADO`	CommandProcessor	Falha escrita / validação

Estados fila (`write_commands`)

Fase	Linhas DB	Observação
Captura	—	RecipeMonitor só lê API
Enfileiramento	`create-recipe` pending + N `write` pending	RecipeProcessor
Execução CLP	pai → `executing`; filhos `completed` / `failed`	CP loop interno

Produção	monitor OPC; ERP muda	PM (sem fechar pai)
Supervisório	backend INSERT `recipe-finalize` pending	Desktop não chama confirm
Reset CLP	filhos `reset_after_production_{syncId}`	finalize / sanitizer
Encerramento	pai `create-recipe` **completed**	`check_executing_recipe_parents`

Diagrama temporal (happy path)

```
T0  ERP: NA_FILA
T1  RM captura → RP.create_recipe_command → DB pending
T2  CP create-recipe → writes OPC → ERP NA_MAQUINA; pai executing
T3  PM detecta produção → ERP EM_PRODUCAO / PAUSADO / …
T4  Operador confirm-action → ERP PROCESSADA|CANCELADO + recipe-finalize
T5  CP finalize → resets no pai → todos completed → pai completed
T6  has_pending false → RM pode pegar próxima NA_FILA
```

Caminhos de exceção

Situação	Mecanismo	Resultado
Falha write receita	`_fail_recipe_write`	ERP `BLOQUEADO`; `reset-defaults`
Pai `executing`, filhos OK, ERP terminal, sem reset	`sanitize_orphan_create_recipes`	Enfileira reset legado
Pai `executing`, sem filhos pendentes	**Não** bloqueia `NA_FILA`	Sanitizer ou manual
CLP offline em reset/finalize	`execute_command` early return	Comando permanece `pending`

🎯 Matriz de dependências

Componente	Criado em	Depende de

AuthService	`main.py`	`API_BASE_URL`, credenciais UI
ApiService	pós-login	token, `tenant_id`
RecipeMonitorService	`CLPContainer.start_recipe_monitoring`	ApiService, DB, callback conexão
RecipeProcessor	`RecipeMonitor._update_processor`	RecipeMapper, `configurationJson`
CommandProcessor	`CLPContainer.start_command_processing`	container, DB, OPC
ProductionMonitorService	`CLPContainer.start_production_monitoring`	ApiService, `configurationJson`, DB

Propagação API: `CLPContainer.set_api_service` atualiza monitors e processor indiretamente via reinicialização de serviços dependentes.

⚙️ Modo sequencial vs paralelo

Atividade	Sequencial (monitor ON)	Sequencial (monitor OFF)	Paralelo (monitor OFF)
Poll comandos	`Coordinator._task_process_commands`	Thread CP	Thread CP
Captura `NA_FILA`	`_task_monitor_recipes`	Thread RM	Thread RM
Leitura variáveis	read worker coordenador	—	`VariableMonitorThread`
`check_executing_recipe_parents`	Início de cada `process_comm`	idem	idem

	`ands` + pós-reconexão		

Variáveis de intervalo: `COMMAND_POLL_INTERVAL_MS`, `RECIPE_POLLING_FAST_MS`, `RECIPE_POLLING_SLOW_MS`, `RECIPE_ADAPTIVE_TIMEOUT` — ver páginas **6**, **7**, **10**.

⚠️ Contratos de fila (leitura obrigatória)

1. **Writes** `recipe_*`: exclusivos de `_execute_recipe_command` — nunca `get_pending_write_commands`.
2. **Writes** `reset_after_production_*`: fila global; fecham pai via `check_executing_recipe_parents`.
3. `has_pending_recipe_commands`: três contagens SQL — ver **7** (não confundir com "qualquer pai executing").
4. **ERP** `NA_MAQUINA` ≠ pai `completed`: NA_MAQUINA após writes iniciais; `completed` após resets.

🔗 Índice de páginas

#	Tópico
3	AuthService
4	ApiService
5	RecipeProcessor
6	RecipeMonitorService
7	CommandProcessor (referência fila/receita)
8	ProductionMonitorService
9–12	Core, UI, OpenAPI