

# Integrador SR → SimpliRoute

---

Plataforma que conecta o banco IW/Oracle ao SimpliRoute com três componentes principais:

- **CLI** para gerar, inspecionar e enviar payloads manualmente.
- **Serviço FastAPI** que roda continuamente (polling + webhook) para sincronizar os dados a cada hora.
- **Persistência de retorno**: cada webhook recebido atualiza `TD_OTIMIZE_ALTSTAT` preenchendo `ACAO`, `STATUS` e `INFORMACAO`.

## Sumário

- Integrador SR → SimpliRoute
  - Sumário
  - 1. Arquitetura resumida
  - 2. Pré-requisitos
    - Estrutura do Instant Client
  - 3. Configuração inicial
    - Estrutura de diretórios relevantes
  - 4. Variáveis de ambiente principais
  - 5. CLI (`src/cli/send_to_simpliroute.py`)
    - Subcomandos
    - Argumentos frequentes
    - Tipos de visita enviados
  - 6. Serviço FastAPI (`simpliroute_service` / `integrador_service`)
    - Como executar localmente
    - Endpoints
    - Webhook → Oracle
  - 7. Execução via Docker
    - Ambiente de desenvolvimento
    - Ambiente de produção (dry-run e real)
  - 8. Testes e validação
  - 9. Troubleshooting
  - 10. Fluxo operacional sugerido
  - 11. Contato e autoria

## 1. Arquitetura resumida

1. **Polling Oracle**: o serviço lê as views configuradas (visitas e entregas) com `fetch_grouped_records`.
2. **Mapeamento**: cada registro passa por `mapper.py`, que normaliza endereço, contatos, itens e define `visit_type` (`med_visit`, `enf_visit` ou `rota_log`).
3. **Envio ao SimpliRoute**: o CLI ou o serviço chama `POST /v1/routes/visits/` usando o token configurado.
4. **Webhook**: o SimpliRoute devolve o status via `POST /webhook/simpliroute`. O JSON é salvo em `data/work/webhooks/` e o Oracle é atualizado.
5. **Colunas atualizadas no Oracle**:
  - `ACAO`: **A** (aguardando), **E** (entregue) ou **S** (suspensa).

- **STATUS**: valores numéricos distintos por **TPREGISTRO** (visitas ou entregas).
- **INFORMACAO**: JSON completo para auditoria.

2. Pré-requisitos

- Python 3.11+ para uso local do CLI e Docker/Docker Compose para execução containerizada.
- Oracle Instant Client (Windows e Linux) armazenado em `settings/instantclient/`.
- Arquivo `settings/.env` com credenciais Oracle, tokens SimpliRoute e parâmetros de polling.
- Acesso de **SELECT/UPDATE** na tabela `TD_OTIMIZE_ALTSTAT` (colunas `ACAO`, `STATUS`, `INFORMACAO`).

Estrutura do Instant Client

- `settings/instantclient/windows/instantclient_23_0/...`: executa localmente no Windows.
- `settings/instantclient/linux/*.zip`: utilizados no build do Docker e instalados em `/opt/oracle/instantclient`.
- `ORACLE_INSTANT_CLIENT`: permite apontar para outro diretório específico, se necessário.

3. Configuração inicial

1. **Clonar o repositório** e copiar um `settings/.env` seguro para a máquina.
2. **Instalar dependências (local)**:

```
python -m venv .venv
.\.venv\Scripts\Activate.ps1
pip install -r requirements.txt
```

3. **Validar a conexão Oracle** ajustando `ORACLE_*` em `settings/.env` e executando `python -m src.cli.send_to_simpliroute preview --limit 1`.
4. **Configurar Docker (opcional)**: `docker compose build` já inclui o Instant Client quando os zips estão no diretório correto.

Estrutura de diretórios relevantes

- `data/input/`: arquivos de entrada opcionais para testes.
- `data/output/`: payloads gerados pelo CLI (`send_to_sr_*.json`).
- `data/work/`: logs do serviço (`service_events.log`) e webhooks (`data/work/webhooks/*.json`).
- `tests/manual/webhook_sample.http`: requisição pronta para testar o webhook.

4. Variáveis de ambiente principais

Categoria	Variável	Descrição
Oracle	<code>ORACLE_HOST</code> , <code>ORACLE_PORT</code> , <code>ORACLE_SERVICE</code> , <code>ORACLE_USER</code> , <code>ORACLE_PASS</code> , <code>ORACLE_SCHEMA</code>	Configuração da conexão.
	<code>ORACLE_VIEW_VISITAS</code> , <code>ORACLE_VIEW_ENTREGAS</code> ou <code>ORACLE_VIEWS</code>	Views lidas pelo polling/CLI.

Categoria	Variável	Descrição
	<code>ORACLE_POLL_WHERE</code>	Filtro adicional aplicado em tempo de execução.
Serviço → Oracle	<code>SIMPLIRROUTE_TARGET_TABLE</code> (padrão <code>TD_OTIMIZE_ALTSTAT</code> )	Tabela que recebe o retorno.
	<code>SIMPLIRROUTE_TARGET_ACTION_COLUMN</code> ( <code>ACAO</code> ), <code>SIMPLIRROUTE_TARGET_STATUS_COLUMN</code> ( <code>STATUS</code> ), <code>SIMPLIRROUTE_TARGET_INFO_COLUMN</code> ( <code>INFORMACAO</code> )	Colunas atualizadas pelo webhook.
SimpliRoute	<code>SIMPLIR_ROUTE_TOKEN</code> (ou <code>SIMPLIRROUTE_TOKEN</code> )	Token para chamadas REST.
	<code>SIMPLIR_ROUTE_WEBHOOK_TOKEN</code>	Assinatura exigida no header <code>Authorization</code> dos webhooks.
Serviço	<code>POLLING_INTERVAL_MINUTES</code> (padrão 60)	Frequência da execução automática.
	<code>SIMPLIRROUTE_POLLING_LIMIT</code>	Limite de registros por ciclo.
	<code>WEBHOOK_PORT</code>	Porta exposta pelo FastAPI (9000/8000 etc.).

Consulte `settings/config.yaml` para valores padrão e exemplos completos.

## 5. CLI (`src/cli/send_to_simpliroute.py`)

### Subcomandos

- `preview`: gera payloads sem enviar e salva em `data/output/` (use `--no-save` para imprimir em tela).
- `send`: gera payloads e, com `--send`, dispara o endpoint do SimpliRoute.
- `auto`: executa o comando definido em `SIMPLIRROUTE_AUTO_COMMAND` (padrão `send --send`), ideal para Docker.

### Argumentos frequentes

- `--limit 10`: controla quantos registros buscar em cada view (padrão `ORACLE_FETCH_LIMIT` ou 25).
- `--where "ID_ATENDIMENTO = 40367"`: aplica filtro adicional.
- `--view VWPACIENTES_COMVISITAS / --views view1 view2`: restringe as views consultadas.
- `--file caminho.json`: usa um arquivo local em vez do Oracle.
- `--send`: habilita o POST para o SimpliRoute (somente no subcomando `send`).

### Tipos de visita enviados

- `med_visit`: visitas médicas (`ESPECIALIDADE/TIPOVISITA`).
- `enf_visit`: visitas de enfermagem.
- `rota_log`: entregas (`TPREGISTRO = 2` ou views de entregas).

- Valores não homologados são ignorados para manter o catálogo alinhado ao SimpliRoute.

## 6. Serviço FastAPI (`simpliroute_service` / `integrador_service`)

Como executar localmente

```
uvicorn src.integrations.simpliroute.app:app --reload --host 0.0.0.0 --port 8000
```

### Endpoints

- `GET /health`, `/health/live`, `/health/ready`: usados pelos healthchecks do Docker e monitoria.
- `POST /webhook/simpliroute`: recebe eventos, valida o token opcional e dispara `persist_status_updates`.

### Webhook → Oracle

- Cada evento gera um arquivo em `data/work/webhooks/webhook_<timestamp>.json`.
- `ACAO` recebe `A`, `E` ou `S` conforme o status do SimpliRoute.
- `STATUS` usa códigos diferentes por `TPREGISTRO`:
  - `TPREGISTRO = 1` (visitas): `0` Planejada, `1` Programada, `2` Realizada.
  - `TPREGISTRO = 2` (entregas): `0` Em preparação, `2` Dispensação, `3` Em rota.
- `INFORMACAO` guarda o JSON completo para auditoria.

Teste manualmente com `tests/manual/webhook_sample.http` (VS Code REST Client) ou adapte para `curl`.

## 7. Execução via Docker

Ambiente de desenvolvimento

```
docker compose build simpliroute_service
docker compose up simpliroute_service
docker compose up simpliroute_cli_limit1 # execução pontual do CLI
```

- Os containers montam `./settings` (somente leitura), `./data/output` e `./data/work` (leitura/escrita).
- Healthcheck padrão consulta `http://localhost:8000/health/live`.

Ambiente de produção (dry-run e real)

- `docker-compose.prod.yml` expõe o mesmo serviço com restart `unless-stopped` e logs limitados.
- Para subir apenas o serviço contínuo: `docker compose -f docker-compose.prod.yml up -d integrador_service`.
- Acompanhe os logs com `docker compose -f docker-compose.prod.yml logs -f integrador_service`.

## 8. Testes e validação

- Configure `PYTHONPATH` para a raiz (`set PYTHONPATH=%CD%` no Windows) e rode `pytest tests/test_mapper.py tests/test_mapper_fixed.py`.
- Para validar um ciclo completo manualmente:
  1. `python -m src.cli.send_to_simpliroute preview --limit 1`.
  2. `python -m src.cli.send_to_simpliroute send --limit 1 --send`.
  3. Envie um webhook de teste e confirme `ACAO/STATUS/INFORMACAO` no Oracle.

## 9. Troubleshooting

Sintoma	Causa provável	Ação sugerida
ORA-01031: insufficient privileges ao receber webhook	Usuário Oracle sem <code>UPDATE</code> em <code>TD_OTIMIZE_ALTSTAT</code>	Solicitar grant específico ( <code>UPDATE</code> nas colunas usadas).
ORA-00942: table or view does not exist	<code>ORACLE_SCHEMA/SIMPLIRROUTE_TARGET_TABLE</code> incorretos ou sem permissão	Ajustar <code>.env</code> e confirmar acesso.
HTTP 400 com Object with key=... does not exist	<code>visit_type</code> inexistente no SimpliRoute	Manter <code>med_visit</code> , <code>enf_visit</code> , <code>rota_log</code> ou cadastrar o novo tipo.
Webhook 401	Header <code>Authorization</code> ausente ou token incorreto	Definir <code>SIMPLIR_ROUTE_WEBHOOK_TOKEN</code> e enviar <code>Bearer &lt;token&gt;</code> .

## 10. Fluxo operacional sugerido

1. Verifique se o container `simpliroute_service` está saudável (`docker compose ps`).
2. Acompanhe `data/work/service_events.log` para confirmar execuções horárias.
3. Para ajustes pontuais (por exemplo, reenviar um atendimento), use o CLI com `--where` e `--limit`.
4. Sempre que o SimpliRoute alterar o catálogo de tipos, atualize a lógica em `mapper.py` antes do próximo envio.

## 11. Contato e autoria

Produzido por: **DAVID BARCELLOS CARDOSO TECNOLOGIA DA INFORMACAO LTDA - ME**

CNPJ: 57.929.932/0001-30

Telefone: +55 21 98605-8337