Funcionamento Detalhado da Pipeline de Deploy (deploy.yml)

O arquivo `deploy.yml` orquestra o processo de Continuous Deployment (CD) para os três ambientes (Dev, Preprod, Prod), garantindo o isolamento de dados e a atualização automática da página HTML e do banco de dados em cada ciclo.

1. Matriz de Gatilhos (Triggers)

A pipeline é ativada por diferentes eventos, conforme a regra de negócio do ambiente alvo.

Ambiente	Gatilho	Condição de Execução
Produção (main/master)	`push`	**Automático:** Sempre que houver um `commit` ou um *Pull Request* for mesclado nessas branches.
Pré-Produção (preprod)	`schedule` E `workflow_dispatch`	**Agendado:** Diariamente às 10:00 UTC E **Manual** via interface do GitHub.
Desenvolvimento (dev)	`workflow_dispatch`	**Apenas Manual:** Acionado sob demanda pela interface do GitHub Actions.

2. Fluxo de Execução da Pipeline

O trabalho (`job: deploy`) é dividido em etapas que garantem a segurança e a correta aplicação das alterações em ambiente isolado.

Etapas Críticas:

Passo	Objetivo	Detalhe da Implementação
Definir Variáveis (`set_env`)	Determina qual ambiente (`prod` , `preprod` , `dev`) foi acionado e carrega os Secrets (`DB_URL` , `DEPLOY_HOOK`) correspondentes.	Usa lógica `if/elif` baseada em `github.event_name` e `target_environment` .
Checkout do Código	Baixa o código-fonte da branch alvo.	Garante que o runner do GitHub tenha acesso aos scripts SQL e ao HTML.
🧱 **INICIALIZAÇÃO: Criar Schema**	Garante o **isolamento de dados** no banco de dados.	Usa o `psql` para executar comandos que **criam um SCHEMA** (`prod` , `preprod` , ou `dev`) com base na variável `TARGET_ENV` e define o `search_path` para esse novo schema.
🚀 **Deploy do Frontend**	Atualiza a página HTML.	Envia um comando `curl POST` para o **Deploy Hook** exclusivo do Vercel (`PROD_DEPLOY_HOOK` , `PREPROD_DEPLOY_HOOK` , etc.), acionando o build e deploy do site estático.
🔄 **Executar Migrações**	Atualiza a infraestrutura do banco de dados.	Executa um *loop* que encontra e roda todos os seus scripts SQL (`V*.sql`) em ordem. Graças ao passo anterior, isso ocorre **dentro do Schema isolado** do

ambiente, garantindo que o `prod` nunca seja afetado pelo `dev`.

3. Isolamento de Ambientes (Schema Based)

O projeto utiliza um único cluster de banco de dados (Supabase) com isolamento por *Schema* (Esquema):

- **Separação Lógica:** Em vez de três bancos de dados físicos, existem três schemas (`prod`, `preprod`, `dev`) no mesmo servidor.
- **Segurança:** A pipeline usa o `psql` para **criar/selecionar** o schema correto no início da sessão. Todas as migrações (criação de tabelas e inserção de dados) ocorrem exclusivamente dentro do schema do ambiente em questão.
- **Idempotência:** Os scripts de migração (`V1__cria_tabela_cliente.sql` , etc.) incluem `DROP TABLE IF EXISTS` e `CREATE TABLE IF NOT EXISTS` para garantir que a re-execução em um ambiente de teste (como `dev`) não cause falhas por tabela já existente.

A conclusão bem-sucedida de todas essas etapas garante que cada ambiente (Dev, Preprod, Prod) esteja sempre atualizado e isolado.