Documentação Técnica: Pipeline de Análise de Chamadas Duplicadas

1.0 Introdução e Visão Geral da Solução

Este documento detalha a arquitetura técnica e o funcionamento de um pipeline de dados automatizado, projetado para processar registros de chamadas, identificar duplicidades de discagem e popular um dashboard de Business Intelligence (BI). A precisão e a atualização contínua dos dados de performance de discagem são de importância estratégica para a operação, permitindo a otimização de recursos, a redução de custos e a melhoria da experiência do cliente.

O processo é orquestrado de ponta a ponta por um conjunto de componentes que se interligam de forma lógica e resiliente. O fluxo completo da solução é executado na seguinte sequência:

1. **Orquestração:** O processo é iniciado pela execução do script Python `ALIMENTA_ESTEIRAS.py`.
2. **Configuração:** O script realiza a leitura do arquivo `config.json` para obter todos os parâmetros necessários, como credenciais de acesso ao banco de dados, a lista de processos a serem executados e as configurações de notificação.
3. **Identificação de Gaps:** Para cada processo definido, o script calcula o "delta" entre os dias que *deveriam* estar na tabela de destino (últimos 7 dias) e os que *realmente* estão, criando uma lista de dias pendentes.
4. **Execução Orientada a Gaps:** O script itera sobre cada data ausente identificada e executa a Stored Procedure correspondente (ex: `SP_TAB_CHAMADAS_DUPLICADAS`), passando a data pendente como parâmetro para processamento.
5. **Processamento de Dados (SQL):** A Stored Procedure, executada no banco de dados, lê os dados brutos de chamadas do dia especificado, aplica a lógica de negócio para agregar e identificar chamadas duplicadas, e por fim, insere o resultado consolidado na tabela de destino.
6. **Visualização (BI):** A tabela de destino (`TAB_CHAMADAS_DUPLICADAS`) atua como a fonte de dados primária para um relatório em Excel (ou outra ferramenta de BI), que apresenta as métricas de forma clara para a análise gerencial.
7. **Notificação:** Ao final de todo o ciclo de execução, o script compila um resumo do status de cada tarefa (sucessos e falhas) e envia uma notificação consolidada para um canal pré-configurado no Microsoft Teams.

Para compreender plenamente a engenharia por trás desta solução, é fundamental primeiro entender o objetivo de negócio que ela se propõe a resolver.

2.0 O Objetivo de Negócio: Análise de Eficiência na Discagem

A finalidade estratégica da análise de chamadas duplicadas é medir e otimizar a eficiência da operação de discagem. Em um ambiente de contact center de alto volume,

discar repetidamente para o mesmo número de telefone em um curto espaço de tempo é um sintoma de problemas que podem incluir ineficiência na gestão do mailing, falhas técnicas no discador ou uma configuração de campanha inadequada. Essas duplicidades não apenas geram custos operacionais desnecessários com telefonia, mas também impactam negativamente a experiência do cliente.

A solução transforma dados transacionais brutos em um conjunto de métricas de alto valor, que são consolidadas e apresentadas em um dashboard. Note que os nomes das colunas gerados pelo procedimento SQL (ex: `DUAS_LIGACOES`) podem ser renomeados na ferramenta de BI para melhorar a legibilidade (ex: '2 Ligações'). A tabela a seguir detalha o significado de cada métrica chave calculada pelo pipeline.

Métrica	Descrição de Negócio	Impacto na Análise
`TOTAL_DE_DISCAGENS`	Volume bruto de tentativas de chamadas realizadas.	Serve como a linha de base para medir a escala da operação e contextualizar as demais métricas.
`TOTAL_DUPLICADAS`	Quantidade de chamadas "extras" realizadas para números que já haviam sido contatados no mesmo dia e hora.	Indica diretamente o desperdício de recursos. Um valor alto é um forte indicador de ineficiência operacional ou técnica.
`TOTAL_DE_NUMEROS_AFETADOS`	Número único de clientes (ou números de telefone) que receberam mais de uma ligação no período.	Mede o alcance do problema, ou seja, quantos clientes distintos foram impactados pela discagem repetida.
`DUAS_LIGACOES`, `TRES_LIGACOES`, `QUATRO_LIGACOES_OU_MAIS`	Segmentação dos números afetados pela intensidade da duplicidade (quantas vezes foram contatados).	Permite à gestão identificar a gravidade do problema, diferenciando casos pontuais de falhas sistêmicas que geram múltiplas repetições.

Para alcançar essa visão de negócio e entregar essas métricas de forma confiável e automatizada, foi implementada uma arquitetura de componentes técnicos robustos, que serão detalhados a seguir.

3.0 Análise Detalhada dos Componentes

A solução é composta por três elementos principais: o motor de processamento em SQL, o orquestrador em Python e o arquivo de configuração em JSON. Cada um desempenha um papel crucial na funcionalidade do pipeline.

3.1 Componente 1: O Motor de Processamento de Dados (`SP_TAB_CHAMADAS_DUPLICADAS`)

A Stored Procedure `SP_TAB_CHAMADAS_DUPLICADAS` é o núcleo da lógica de negócio deste pipeline. Sua responsabilidade é transformar milhões de registros de chamadas brutas em um conjunto agregado e conciso de métricas de inteligência.

A estratégia de funcionamento da procedure pode ser decomposta nos seguintes pontos:

- **Parâmetro de Entrada:** A procedure foi projetada para receber um único parâmetro, `@DataExecucao`. Essa abordagem a torna modular e reutilizável, permitindo que o orquestrador a invoque para processar qualquer dia específico sob demanda.
- **Construção de Query Dinâmica:** A procedure constrói dinamicamente o nome da tabela de origem (ex: `Atlas.dbo.tb_Dialer_Calls_20251019`) com base no parâmetro de data. Essa técnica é essencial para consultar tabelas de log que são particionadas por dia, evitando a necessidade de modificar o código SQL para analisar diferentes períodos.
- **Lógica de Agregação (CTE):** A lógica principal é encapsulada em uma Common Table Expression (CTE) chamada `ContagemChamadasPorNumero`. Esta CTE agrupa os registros de chamadas por múltiplas dimensões de negócio, incluindo `origem`, `FILA`, `SERVIDOR`, `CLIENTE` e `mailing`. Essa granularidade é crucial, pois permite que a análise no BI identifique a origem exata do problema, determinando se a duplicidade está concentrada em uma campanha específica (`mailing`), uma equipe operacional (`FILA`) ou uma falha de tecnologia (`TECNOLOGIA`).
- **Cálculo das Métricas Finais:** A consulta final utiliza o resultado da CTE para calcular as métricas de negócio. Expressões `SUM` e `CASE WHEN` são usadas para derivar o `TOTAL_DUPLICADAS` (contando apenas as chamadas que excedem uma por número), o `TOTAL_DE_NUMEROS_AFETADOS` e a segmentação por intensidade (`DUAS_LIGACOES`, etc.).
- **Manutenção de Dados:** A procedure implementa uma estratégia de `DELETE` com duplo propósito antes da inserção dos dados. Ela remove registros da `@DataExecucao` para garantir a **idempotência** do processo (permitindo reexecuções seguras para o mesmo dia) e, simultaneamente, funciona como uma **política de arquivamento**, removendo dados com mais de 90 dias para evitar o crescimento indefinido da tabela e manter sua performance.

Este poderoso motor SQL, no entanto, requer um mecanismo externo para ser executado de forma automática e inteligente. Esse papel é desempenhado pelo script orquestrador.

3.2 Componente 2: O Orquestrador Automatizado (`ALIMENTA_ESTEIRAS.py`)

A arquitetura do script `ALIMENTA_ESTEIRAS.py` é intencionalmente genérica. Ao externalizar toda a lógica de negócio e o estado para SQL e JSON, o código Python é reduzido a um motor de orquestração puro e reutilizável. Sua função é automatizar a execução, garantir a atualização dos dados e notificar as equipes sobre a saúde do pipeline.

Os princípios de design do orquestrador são:

- **Abstração via Configuração:** O script não contém nomes de procedures, tabelas ou qualquer lógica de negócio "hardcoded". Em vez disso, ele opera

iterando sobre uma lista de `processos` definida no `config.json`, tratando cada um como uma unidade de trabalho atômica e independente.

- **Flexibilidade e Escalabilidade:** Esta abordagem orientada à configuração torna o sistema altamente extensível. Novos pipelines de ETL podem ser integrados à automação simplesmente adicionando um novo objeto JSON à lista de `processos`, sem exigir *nenhuma* alteração no código Python.
- **Lógica de Verificação de "Gaps":** O script implementa uma lógica de resiliência e autocorreção. Para cada processo, ele consulta a tabela de destino para identificar as datas já processadas no intervalo de verificação (últimos 7 dias) e calcula os "dias faltantes". Isso garante que, mesmo que uma execução diária falhe, o sistema processará automaticamente os dias pendentes na próxima execução.
- **Tratamento de Erros e Notificações:** Blocos `try...except` robustos capturam erros de banco de dados ou exceções inesperadas. A função `enviar_notificacao_teams` centraliza toda a comunicação de status, garantindo que, independentemente do resultado (sucesso, falha parcial ou erro crítico), uma mensagem clara seja enviada, proporcionando total visibilidade sobre a saúde do pipeline.

A notável flexibilidade do orquestrador Python depende inteiramente da estrutura e do conteúdo do seu arquivo de configuração, que atua como o centro de controle de todo o sistema.

3.3 Componente 3: O Centro de Controle (`config.json`)

O arquivo `config.json` atua como o painel de controle de todo o pipeline. Ele desacopla as configurações operacionais (como credenciais, nomes de objetos e endpoints) do código da aplicação. Essa separação é uma prática recomendada de engenharia de software, pois permite que administradores ou analistas ajustem parâmetros de execução sem editar o código fonte Python.

A estrutura do arquivo é dividida em seções lógicas, detalhadas na tabela abaixo:

Seção	Chave	Descrição da Finalidade
`conexao_banco`	`servidor`, `banco_de_dados`, etc.	Centraliza todas as credenciais e parâmetros para a conexão com o SQL Server. Permite migrar facilmente entre ambientes (desenvolvimento, produção) alterando apenas este bloco.
`objetos_banco`	`processos` (lista)	**O coração da automação.** Define quais pipelines devem ser executados. A presença de múltiplos processos, como `"Tempo Atendimento"` e `"Chamadas Duplicadas"`, demonstra a capacidade do orquestrador de gerenciar ETLs completamente distintos com uma única engine, validando o poder do design desacoplado.
`notificacoes`	`teams` / `webhook_url`	Contém os endpoints para os sistemas de notificação. Permite alterar o canal do Teams ou desativar as notificações sem a necessidade de qualquer alteração no código Python.

Com a compreensão da arquitetura e da função de cada componente, o próximo passo é detalhar as ações práticas necessárias para configurar e executar o sistema.

4.0 Guia de Implementação e Modificações Necessárias

Esta seção funciona como um guia prático para um novo usuário ou mantenedor do sistema, listando os passos essenciais para configurar o ambiente e garantir a execução bem-sucedida do pipeline. Todas as alterações são feitas exclusivamente no arquivo `config.json`.

Para que o pipeline funcione corretamente, as seguintes modificações de configuração são obrigatórias:

1. **Configurar a Conexão com o Banco de Dados:**
    - No arquivo **`config.json`**, localize a seção **`"conexao_banco"`**.
    - Substitua os valores de placeholder pelas credenciais reais do ambiente de produção para as chaves: **`"servidor"`**, **`"banco_de_dados"`**, **`"usuario"`** e **`"senha"`**.
2. **Configurar as Notificações:**
    - No arquivo **`config.json`**, localize a seção **`"notificacoes"`** e, dentro dela, o objeto **`"teams"`**.
    - Substitua o valor da chave **`"webhook_url"`** pela URL do Webhook de entrada do canal do Microsoft Teams onde as notificações de status devem ser enviadas.
3. **Ajustar Nomes de Banco de Dados nos Processos:**
    - No arquivo **`config.json`**, dentro da lista **`"processos"`**, os valores para a chave **`"tabela_verificacao"`** contêm um placeholder com um erro de digitação (ex: `"databese_name" [sic]`).
    - Substitua **`"databese_name" [sic]`** pelo nome real do banco de dados onde a tabela de destino está localizada.
    - *Importante:* Este valor deve ser um nome totalmente qualificado (`nome_banco.schema.tabela`), pois é usado diretamente em uma consulta pelo script Python para verificar as datas ausentes.
4. **Verificar Nomes das Stored Procedures:**
    - **Atenção:** Existe uma inconsistência entre os arquivos de origem. O `config.json` referencia a procedure como `"dbo.SP_PROCEDURE_CHAMADAS_DUPLICADAS"`, enquanto o script SQL a define como `[dbo].[SP_TAB_CHAMADAS_DUPLICADAS]`.
    - Para evitar erros de execução, localize o processo `"Chamadas Duplicadas"` no arquivo **`config.json`** e garanta que o valor da chave **`"procedure_executar"`** corresponda **exatamente** ao nome da procedure implantada no SQL Server.

Uma vez concluídas essas configurações, o sistema está pronto para ser executado, oferecendo uma solução robusta e automatizada para a análise de dados.

5.0 Conclusão

A solução apresentada demonstra o poder de uma arquitetura modular que combina a eficiência do processamento de dados do SQL Server com a flexibilidade de orquestração do Python e a simplicidade de um arquivo de configuração JSON. O resultado é um pipeline de dados que não é apenas poderoso e totalmente automatizado, mas também resiliente a falhas e facilmente extensível para incorporar novas análises de dados no futuro. Ao seguir este design, o sistema transforma com sucesso dados operacionais brutos em inteligência acionável, fornecendo à gestão de negócios as ferramentas necessárias para otimizar processos, reduzir custos e tomar decisões mais bem informadas.