Projeto AtivaAnalyticDataService

O objetivo deste projeto é criar e manter um serviço de fornecimento de informações operacionais da Ativa Investimentos para auxílio na toma de decisões estratégicas, no acompanhamento de indicadores e nos processos de controle. A principal justificativa para esse investimento é o aprimoramento do processo de tomada de decisões de negócios com base em dados. O serviço é composto por uma camada de coleta e organização dos dados e outra camada para interação dos usuários.

A camada que abriga a coleta, organização, armazenamento e atualização dos dados é denominada data warehouse. De maneira ampla, pode-se dizer que data warehouse é um local centralizado para consulta de dados históricos oriundos dos sistemas utilizados pelas áreas operacionais (OLTP), útil para consultas analíticas tanto das próprias áreas operacionais quanto para áreas de controle e de estratégia de negócios. Além dos dados dos sistemas operacionais, o data warehouse também contém dados de ambientes externos, como, por exemplo, cotações de papeis e cotas e fundos.

A segunda camada é a parte visual do serviço. Por meio dela, os usuários terão acesso às ferramentas de dados criadas pela equipe de análise de dados. Como exemplo destas ferramentas, citamos o descarregamento de dados para enviar a órgãos reguladores e o acompanhamento de indicadores para áreas de atendimento ao cliente. Usuários com conhecimentos técnicos podem acessar diretamente a base de dados e construir suas próprias consultas. As questões relativas à segurança e sigilo da informação serão abordadas mais a frente, na sessão que detalha essa camada do sistema.

As duas sessões seguintes, tratam do funcionamento de cada camada do serviço. Em primeiro lugar, a arquitetura do sistema para coleta, organização, armazenamento e atualização dos dados, isto é, o data warehouse. Na sessão seguinte, a arquitetura do sistema para acesso e visualização dos dados. Por último, apresentamos um cronograma para desenvolvimento, testes e implantação do AtivaAnalyticDataService (AADS).

# AADS – Data warehouse

Do ponto de vista técnico, pode-se destacar os seguintes benefícios da utilização de um data warehouse: a) melhoria do desempenho de consultas analíticas; b) possibilidade de análises multidimensionais por meio de consultas a dados de diversas fontes em um local centralizado e; c) mitigação dos riscos inerentes de consultas realizadas diretamente nos servidores de produção, sendo o risco de degradação no desempenho dos sistemas de produção o mais crítico.

O projeto incluirá o design, desenvolvimento, teste, implantação e manutenção contínua do data warehouse. A seguir, estão listadas as principais etapas para implantação e manutenção do data warehouse.

Implantação:

1. Levantamento de requisitos: reuniões com as partes interessadas para entender as necessidades de dados da organização e identificar as fontes de dados que precisam ser integradas ao data warehouse.
2. Projeto do data warehouse: criar um modelo de dados conceitual, lógico e físico para o data warehouse. Isso inclui identificar as fontes de dados, determinar a estrutura de dados e projetar o esquema do data warehouse.
3. Desenvolvimento do data warehouse: projetar e implementar os processos de Extração, Transformação e Carregamento (ETL) dos dados das aplicações operacionais (OLTP).
4. Testes: usar amostras de dados para validar o data warehouse e garantir que ele extraia, limpe e carregue os dados de forma correta e eficiente.
5. Implantação do data warehouse: migrar o data warehouse para o ambiente de produção e disponibilização aos usuários.

Manutenção:

1. Monitoramento e manutenção do data warehouse: monitorar continuamente o data warehouse para garantir que ele esteja funcionando sem problemas. Solucionar e corrigir eventuais problemas que surjam. Executar tarefas de manutenção regulares, como backups, ajuste de desempenho e atualizações para garantir que o data warehouse seja estável e eficiente.
2. Aprimoramento do data warehouse: reunir continuamente feedback de usuários e partes interessadas para identificar novos recursos e aprimoramentos que podem ser adicionados ao data warehouse para melhorar sua funcionalidade e valor para a organização.
3. Governança de dados: implementar e manter processos de governança de dados para garantir que os dados sejam gerenciados adequadamente em todo o seu ciclo de vida, incluindo qualidade de dados, linhagem de dados, governança de dados, linhagem de dados, segurança de dados, privacidade de dados e conformidade de dados.

Um ponto a se observar é que a Ativa já dispõe de um data warehouse que está na sua terceira geração. Logo, entendemos que as etapas 1 e 2 da implantação do sistema já foram cumpridas em boa medida. Contudo, em todas as três gerações do sistema, foram utilizadas ferramentas (Pentaho e MS SSIS) do tipo apontar e clicar (*low code* ou *no code*) para a construção dos processos ETL. Esse tipo de ferramenta produz soluções monolíticas do tipo caixa preta, que dificultam o rastreamento e controle de alterações e a manutenção do código por mais de uma pessoa. Por esses motivos, nosso foco estará no desenvolvimento de um sistema robusto de processos de cópia e tratamento dos dados (ETL pipeline) que será descrito adiante.

## Arquitetura do ETL pipeline

Para a definição da arquitetura do processo de ETL do AtivaAnalyticDataService consideramos dois fatores: a necessidade de atualização (*on time*) dos dados e os recursos físicos (*hardware*) disponíveis. Analisando o funcionamento do sistema atual, depreendemos que é suficiente para as partes interessadas a disponibilização dos dados com defasagem de um dia (d-1). Portanto, não trabalharemos com o conceito de data pipeline, cujo fluxo de dados do operacional para o analítico é contínuo em tempo real[[1]](#footnote-1). Será mantido o conceito de ETL pipeline, que consiste em etapas sequenciais e estanques de cópia e gravação de dados. Esse tipo de abordagem é mais simples e barato. Outro ponto que contribuiu para a decisão de utilizar ETL pipeline é a infraestrutura disponível. Uma solução do tipo data pipeline demanda recursos para escalabilidade. Dado que temos um único ponto para gravação dos dados, disparar processos paralelamente geraria concorrência de escrita e possível lentidão. Por seu turno, a arquitetura do tipo ETL pipeline requer menos recursos de hardware.[[2]](#footnote-2)

O objetivo do ETL pipeline é povoar a base stage do data warehouse. Em favor da simplicidade, o sistema executa exclusivamente uma operação por vez e tem apenas um tipo de operação: ler dados de uma fonte (ponto A) e gravar os dados em uma base (ponto B). O ponto intermediário para tratamento de dados será uma base denominada storage.

Conquanto pareça um domínio extremamente limitado, ao consideramos a diversidade de fontes de dados, tais como sites e SGBDs combinadas à multiplicidade de estruturas de dados como HTML, XML, JSON, podemos inferir que estamos diante de um domínio de complexidade elevada. Outro ponto a ser observado é a perda de velocidade de atualização dos dados que advém da restrição de se executar uma operação por vez. Contudo, entendemos que essa perda é compensada pelo ganho na simplificação do gerenciamento sequencial das operações. Em resumo, processos sequenciais são mais simples que processos paralelos, e defendemos que a perda de velocidade de execução é recompensada pelo ganho na velocidade de desenvolvimento e na mitigação de erros de concorrência de processos.

Para executar as operações de cópia do ponto A para o ponto B, foi criada a classe operator. Ela realiza os controles a fim de garantir que apenas uma operação será executada por vez. Essa classe também é responsável por gerar os logs de execução com as informações resultado de sucesso ou falha, início, fim e tamanho do conjunto de dados processado em bytes e linhas, quando for o caso.

Desta classe operator foram derivadas três classes especializadas: extract, transform e load. A classe extract lê dados de qualquer origem, mas que dependem de tratamento antes do carregamento na stage, logo, o destino é sempre a base storage. A classe transform trabalha única e exclusivamente com a base storage, sempre lendo dados brutos ou tratados parcialmente e gravando os dados tratados na própria storage. Por último, a classe load lê dados estruturados de sistema externos ou da base storage e carrega na stage. Resumindo: extract sempre lê dados externos (ponto A) e grava na storage (ponto B); transform sempre lê e grava dados na storage (ponto A e B); load lê dados externos ou da storage (ponto A) e sempre grava na stage (ponto B). Logo, conclui-se a classe extract nunca carrega dados na stage, que a classe transform tem como domínio único a base storage e que apenas a classe load carrega dados na base stage. Desta forma, tem-se classes especializadas para cada domínio, simplificando tarefas tais como: o gerenciamento de autenticação de acesso às bases, ciclo de operação (leitura/gravação) e o próprio processo de codificação da classe.[[3]](#footnote-3)

Para gerenciamento das operações, foi criada a classe manager. Essa classe tem o conhecimento das operações que devem ser executadas assim como dos seus respectivos disparadores, que podem ser por tempo ou pela existência de um arquivo na storage. Como, no momento, não há necessidade de execução em paralelo, as tarefas são organizadas de forma sequencial com base nos disparadores. A cada ciclo, a fila de execuções é atualizada considerando os disparadores as execuções com sucesso e com falha. De maneira geral, as execuções com falha são reposicionadas no final da fila para nova execução considerando os horários bloqueados quando houver.

Etapas de construção da fila.

1. Tarefas de load que acessam bases externas, isto é, não entram as tarefas de load que leem dados da storage;
2. Tarefas de extract;
3. Tarefas de transform;
4. Tarefas de load que acessam storage.

1. O nome *data pipeline* remete aos gasodutos (*gas pipeline*) onde o fluxo é contínuo. [↑](#footnote-ref-1)
2. Utilizar um gerenciador de data pipeline como o Apache AirFlow acrescentaria uma camada de complexidade para resolver um problema que não temos agora. Precisaríamos criar nossas próprias classes especializadas (*operators*). [↑](#footnote-ref-2)
3. Essa especialização facilita, por exemplo, a criação de processos como a coleta diária do PU Anbima. [↑](#footnote-ref-3)