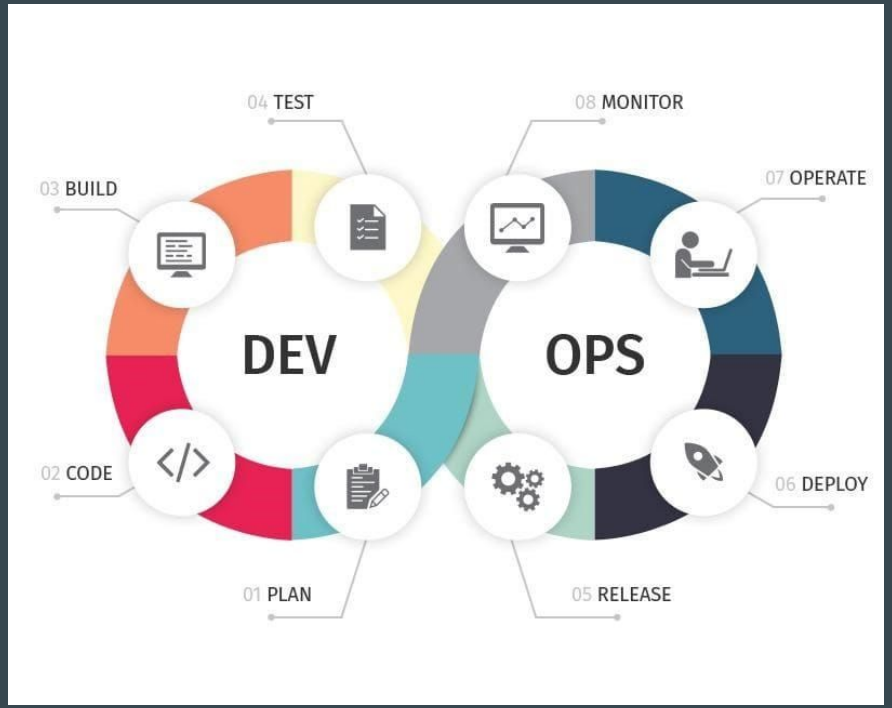
Dia 02

DevOps

DevOps


04 TEST
08 MONITOR
03 BUILD
07 OPERATE
DEV
OPS
02 CODE
06 DEPLOY
01 PLAN
05 RELEASE

DevOps - Planning

Planejamento (Planning):

- Esta etapa envolve a definição de requisitos e o planejamento das tarefas a serem executadas.
- É aqui que a equipe identifica as necessidades do projeto e estabelece metas e objetivos.
- Ferramentas como JIRA, Trello ou Asana podem ser usadas para rastrear tarefas e histórias de usuários.

DevOps - Coding

Codificação (Coding):

- Os desenvolvedores escrevem o código da aplicação nesta fase.
- Utilizam-se práticas como programação em pares e revisões de código para garantir a qualidade.
- O código é armazenado em sistemas de controle de versão, como Git.

DevOps - Build

Construção (Build):

- O código é compilado, testado e empacotado para implantação.
- As ferramentas de integração contínua, como Jenkins ou Travis CI, podem ser usadas para automatizar essa etapa.
- Esta fase garante que o software seja construído corretamente antes de ser implantado.

DevOps - Testing

Teste (Testing):

- O software é submetido a uma série de testes para garantir sua qualidade.
- Testes unitários, de integração, funcionais, entre outros, são realizados.
- Ferramentas como Selenium ou JUnit podem ser usadas para automação de testes.

DevOps - Release

Lançamento (Release):

- Esta etapa envolve a preparação para a implantação do software em produção.
- Versões específicas do software são definidas e marcadas, garantindo que o que foi testado seja o que será implantado.
- Ferramentas como GitLab, Jenkins ou Spinnaker podem ser usadas para gerenciar e automatizar os processos de release.

DevOps - Deployment

Implantação (Deployment):

- Uma vez testado, o software é implantado em ambientes de produção.
- Esta fase pode ser automatizada para permitir implantações frequentes e confiáveis.
- Ferramentas como Docker e Kubernetes podem ser usadas para facilitar implantações.

DevOps - Operations

Operação (Operations):

- Esta etapa envolve monitorar e manter o software em produção.
- Problemas são identificados e resolvidos rapidamente para garantir a disponibilidade.
- Ferramentas como Nagios ou Datadog podem ser usadas para monitoramento.

DevOps - Monitor

Monitoramento (Monitor):

- O feedback contínuo é coletado de várias partes interessadas, incluindo usuários finais, para melhorar o software.
- Monitoramento constante permite identificar problemas antes que eles afetem os usuários.
- Percepções desta etapa ajudam a informar as etapas futuras do ciclo DevOps.

Dúvidas

Revisão de Git

Git - Contexto

- TCC
- Execício de todo mundo trabalhar no mesmo TCC
- História de quando eu deletei uma base de produção em Access
- História de quando eu apaguei um projeto inteiro de duas semanas
- Sexta feira da maldade
- Dropbox, GDrive, Onedrive, etc.

Git - Problemática

- Colocou um bug no código de um dia para outro e só percebeu na semana seguinte
- Excluiu uma parte do código sem querer
- Renomear ou apagar arquivos que não devia
- Após errar algumas vezes, você vai começar a duplicar os arquivos antes de editar
- Vai ficar difícil de descobrir qual a versão correta (*_final, _ultima, _bkp, _1, _2, etc.)
- Utilização de espaço desnecessária com várias versões de arquivo na mesma pasta
- Trabalhar um dia inteiro no arquivo e no dia seguinte descobrir que fechou o arquivo sem salvar e perdeu um dia de trabalho

Git - Problemática

- Criação de várias versões
- Dificuldade para trabalho em equipe
- Problemas com gestão de conflitos
- Voltar para uma versão anterior
- Evoluir de modo sustentável
- Várias pessoas trabalhando no mesmo arquivo ao mesmo tempo
- Perder o trabalho do dia, da semana, ou pior, de alguém

Git - Conceitos

- **Diff:** diferença entre dois arquivos (performático e economiza espaço)
- **Patch:** aplicar o resultado da diferença entre dois arquivos no mais antigo para gerar o mais novo
- Armazenar apenas as alterações é mais econômico do que armazenar o novo arquivo inteiro ou ter duplicatas de arquivos
- Diff arq1.txt arq2.txt > arq1.patch && patch arq1.txt < arq1.patch

Git - Conceitos

- **Repositórios:** pasta controlada pelo git, onde será realizado o versionamento dos arquivos

Git - Estágios

- **Status:** estágios dos arquivos
  - **Untracked/Não rastreado:** git não está versionando esse arquivo.
  - **Staged/Adicionado:** Selecionado para o versionamento na área de stage. (não está versionado ainda)
  - **Modified/Modificado**
  - **Deleted/Deletado**
  - **Commited/Cofirmado:** Arquivos selecionados estão versionados.

Git - Commit

- **Commit/Confirmação:** Depois da confirmação qualquer alteração vai ser reversível, alguns metadados são salvos para histórico.
  - **Messagem de commit:** o que foi alterado (em resumo)
  - **Usuário cadastrado:** quem realizou a alteração (user.name)
  - **Email cadastrado:** contato de quem realizou (user.email)

Git - Galhos

- **Branchs/Galhos:** uma “linha do tempo” independente de desenvolvimento (não interfere nas outras ramificações)
  - **Main/Master/Prod:** ramificação principal (alterações aprovadas)
  - **QA/Staging/Homologação:** teste e validação das alterações (geralmente é para outra equipe)
  - **Desenvolvimento:** desenvolve novas alterações (desenvolvedor)
    - **Nome de funcionalidade:** alterações daquela funcionalidade
    - **Nome do desenvolvedor:** alterações do desenvolvedor em teste/desenvolvimento

Git - Mescla

- **Merge:** Mesclar duas branchs
  - **Pull request (Merge request):** solicitação de mesclagem entre duas ramificações.
  - **Conflitos:** Durante o merge algumas alterações podem entrar em conflito.

Git - Comandos

- **Init:** inicializa um repositório.
- **Status:** verificar o status de cada arquivo
- **Add/rm:** Comandos para modificar arquivos
- **Commit:** Criar uma versão (estado da pasta)
  - Mensagem, email, usuário (obrigatórios)
- **Configs:** gerenciar configurações do git (--global) ou repositório (--local)
  - Configurar email: git config --local user.email “exemplo@gmail.com”
  - Configurar o nome: git config --local user.name “Exemplo”
- **Configurar o vscode como editor padrão:** git config --local core.editor “code --wait”
- **Log:** registro dos commits
- **Reset --hard:** restaura o repositório para uma versão anterior descartando as alterações após aquela versão(cuidado!)

Demo

...

Criando o repositório do projeto

Github

Git vs Github

- **Git:** ferramenta (algo instalável) que serve para versionar código, conteúdo, etc.
- **Github:** armazenamento em nuvem (similar ao google drive), serve para salvar arquivos da máquina local (da máquina pessoal) na nuvem (máquina remota).

Git - Comandos

- **Clone:** copia um repositório para a máquina local
- **Remote:** máquina remota onde é possível sincronizar repositórios
  - Add: adicionar um repositório remoto
  - Remove: remover o link ao repositório
  - Rename: renomear a tag do repositório
- **Push:** Envia (empurra) as alterações para o repositório remoto
- **Fetch:** buscar se há novas alterações na máquina remota sem puxá-las.
- **Pull:** puxar as alterações da máquina remota para a máquina local
- **Branch:** criar uma ramificação
- **Checkout:** mudar para uma ramificação ou commit
- **Merge:** mesclar duas branchs

Demo

...

Adicionando repositório ao github

Git para times de dados

https://theplumbers.com.br/

Dúvidas

DBT

DBT

- Ferramenta para construção de pipelines de dados seguindo as boas práticas de desenvolvimento (modularização, colaboração, versionamento de código).
- Utiliza linguagem SQL
- Feita em Python
- Open Source (Apache 2.0)
- Integração com diversos Data Warehouses
- Utilização de templates (Jinja)
- Implementa o ciclo DevOps/DataOps
- Teste de dados
- Documentação automática
- Paralelismo (Threads)

DBT

“O poder do DBT vem de uma combinação de linguagem, SQL para definir transformações de dados; YAML para configurar, documentar e testar essas transformações; Jinja (python) para modelagem e linhagem; e Python no fundo unindo esses três.”

DBT

Tem como objetivo levar princípios de engenharia de software para a transformação de dados. Como por exemplo:

- Código modular (macros, reuso, don’t repeat yourself, jinja)
- Controle de versão de código (auditoria, recuperação de versão, log de alteração, colaboração, branches, git)
- Teste de software (evitar bugs, melhorar a qualidade do código, impedir código ruim em produção, dbt test)
- Seguir o ciclo DevOps (Package, Release, CI/CD, Monitoring)
- Documentação atualizada

DBT - Benefícios

Envia e gerencia comandos para que os dados sejam transformados diretamente no Data Warehouse, com isso alguns benefícios:

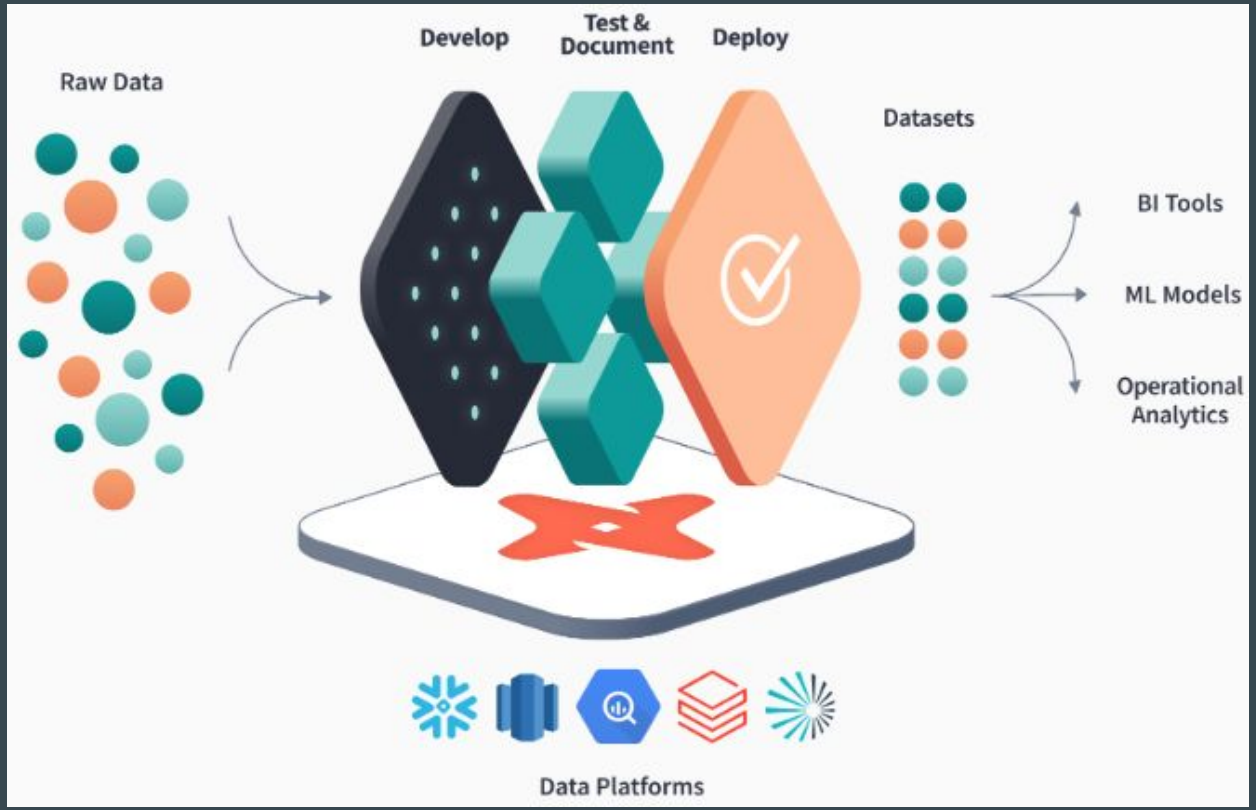
- Melhorando a segurança, governança, e confiabilidade dos dados.
- Remove a necessidade de uma nova extração dos dados crus e carga dos dados transformados (economiza tempo e computação).
- Remove a necessidade de um segundo servidor similar ao data warehouse para transformação de dados (economiza dinheiro).
- Transformação de dados utilizando SQL e Python* (baixa curva de aprendizagem).
- Implementação de Testes e geração automática de documentação

Quem usa?

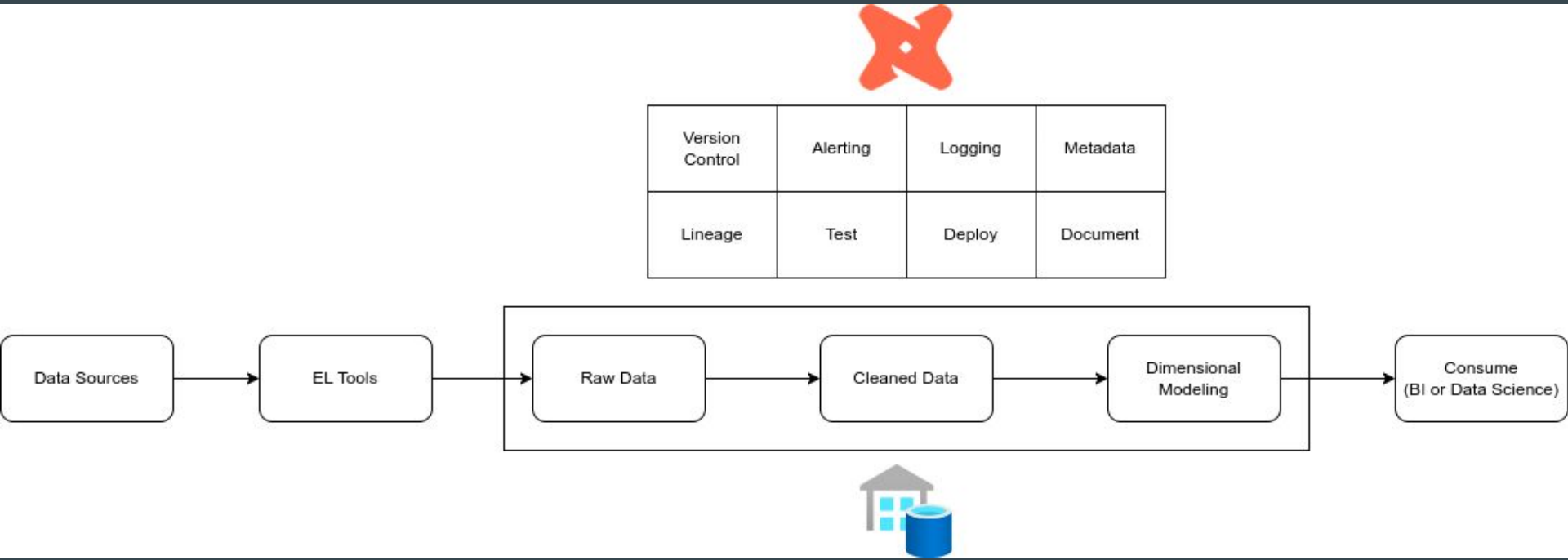
...

https://www.getdbt.com/success-stories/

Como funciona?



Como funciona?



Formas de usar

- DBT Cloud
- DBT Core

DBT - Plataformas

- **DBT Cloud:** é uma plataforma SaaS oferecida pela DBT Labs (antiga Fishtown Analytics) que facilita a orquestração, execução e colaboração de projetos DBT. Ele fornece uma interface amigável, integrações com sistemas de controle de versão e recursos de CI/CD, tornando a engenharia de dados mais acessível e gerenciável para equipes.
- **DBT Core (CLI):** é a versão de linha de comando do DBT, uma ferramenta gratuita e open-source para transformação de dados. Ele permite que os engenheiros de dados escrevam, documentem e testem transformações SQL, usando o terminal para executar e gerenciar projetos DBT em sua infraestrutura local ou em cloud.

Formas de usar

- **DBT Cloud:** Aplicação SaaS (na nuvem)
  - IDE online para desenvolvimento
  - Orquestração de Jobs
  - Logging e alertas simplificados
  - Documentação da ferramenta integrada
  - Facilidade para os comandos git
  - Facilidade de configuração de conexão à repositórios e cloud data warehouses
  - Grátis para indivíduos

Formas de usar

- **DBT Core:** Open source, pacote python
  - Maior liberdade de desenvolvimento e controle do ambiente
  - Conexão com data warehouses locais
  - Maior opção de adapters integrados e da comunidade
  - Grátis para uso por times de dados
  - Requer maior configuração para uso

DBT Core - Instalação

DBT - Instalação - Requisitos

- Python
- Git
- Adapter

DBT - Adapter

DBT - Adapter

- São componentes que permitem que o DBT se comunique e opere em diferentes plataformas de data warehouse ou bases de dados. Cada adapter é essencialmente uma interface entre o DBT e uma plataforma específica de data warehouse, permitindo que o DBT leia, escreva e execute SQL nessa plataforma. Graças a estes adapters, o DBT pode ser usado com uma variedade de plataformas, incluindo BigQuery, Snowflake, Redshift, e muitas outras.

DBT - Adapter

Funções principais dos adapters:

- **Conexão:** Estabelecer e gerenciar conexões com o data warehouse.
- **SQL Dialect:** Traduzir as abstrações do DBT para SQL específico de cada plataforma (DDL, DML, etc.).
- **Operações de Dados:** Executar operações como criação de tabelas, visualizações, execução de transformações e carregamento de dados.

DBT - Adapter - Tipos

- **Verificado (Verified):** O rigoroso programa de adaptadores do dbt Labs garante aos usuários adaptadores confiáveis, testados e regularmente atualizados para uso em produção. Adaptadores verificados recebem um status de "Verificado", proporcionando aos usuários confiança e segurança.
- **Comunidade (Community):** Adaptadores da comunidade são de código aberto e mantidos por membros da comunidade.

DBT - Adapter - Oficiais

- Alloy DB
- BigQuery
- Databricks/Spark
- Postgres
- Redshift
- Snowflake
- Starburst/Trino
- Dremio*
- Synapse/Fabric*
- Teradata*

* Não suportados em DBT Cloud

DBT - Adapter - Comunidade

- Athena
- Clickhouse
- DB2
- DuckDB
- Hive
- Impala
- MySQL
- Oracle
- Sql Server
- SQLite
- Outros

DBT - Adapter - profiles.yml

- Arquivo contendo todas as informações necessárias para conexão com banco de dados.
- Este arquivo fica geralmente separado do projeto por conter informações sensíveis.
- Pode utilizar variáveis de ambiente para evitar expor informações sensíveis.
- Para isso utilize a função {{ env_var('<variavel_ambiente>', '<valor_padrao>') }}.

PS: DBT Cloud não utiliza arquivo profiles.yml para se conectar a base de dados.

DBT - Adapter - profiles.yml

- É preciso especificar para o DBT a pasta em que se encontra esse arquivo.
- DBT irá seguir a seguinte ordem de pesquisa:
  1. --profiles-dir option
  2. DBT_PROFILES_DIR environment variable
  3. current working directory
  4. ~/.dbt/ directory

PS: DBT Cloud não utiliza arquivo profiles.yml para se conectar a base de dados.

DBT - Primeiros comandos

- dbt --version
- dbt debug
- dbt --config-dir
- dbt <comando> --profiles-dir <caminho_diretorio>
- export DBT_PROFILES_DIR=<caminho_diretorio>
- dbt <comando> --target=<target>

DBT - Boas práticas

- `pip install dbt` não é mais suportado, utilize `pip install dbt-core` ou `pip install dbt-<adapter>`
- Por conter informações sensíveis, não adicione o arquivo profiles.yml no controle de versão (adicione-o ao .gitignore)
- Variáveis de ambiente que serão utilizadas com DBT devem utilizar o prefixo DBT_
- Para variáveis de ambiente contendo informações sensíveis, nomeie-as com o prefixo DBT_ENV_SECRET_, para evitar exposição acidental. Já que usuários não podem modificar secrets utilizando jinja.

Referência

...

https://docs.getdbt.com/docs/supported-data-platforms

DBT Core

...

https://docs.getdbt.com/docs/core/installation

DBT Core Version

...

https://docs.getdbt.com/docs/dbt-versions/core

Dúvidas

Demo

Primeiro projeto DBT

Dúvidas