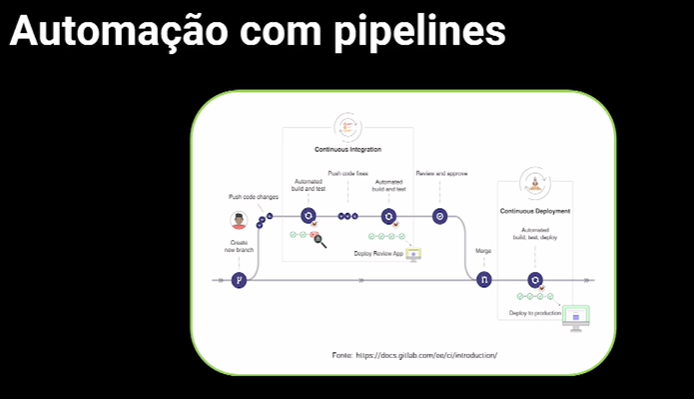
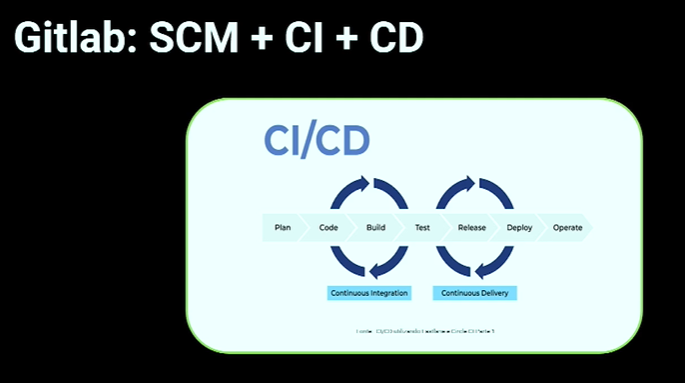
**Eficiência operacional em entrega de software:**

SCM: Source Control Management (SCM) é uma ferramenta essencial para controle de versão e colaboração em desenvolvimento de software, evitando problemas de sincronização e perdas de código.

As ferramentas de SCM permitem trabalho paralelo sem conflitos, histórico de versões para recuperação de código e exigem boas práticas, como manter o código atualizado e integrar código com maturidade, definindo um integrador responsável por lidar com conflitos.

Ferramentas como Git permitem colaboração em paralelo, merge de código e acompanhamento histórico, facilitando a atualização e manutenção do código de forma organizada e eficiente.

**Gitlab:**



Um pipeline é um conjunto de etapas automatizadas que conduzem o código por diferentes estágios, desde a integração até a entrega. Ele funciona como um fluxo, onde o código é verificado, testado e integrado continuamente, garantindo qualidade e feedback rápido.

As pipelines utilizam scripts para automatizar tarefas, reduzindo erros humanos e garantindo a execução consistente do processo. Através da execução automatizada de scripts, a pipeline elimina a necessidade de intervenção manual, minimizando o risco de falhas e garantindo a entrega eficiente do código.



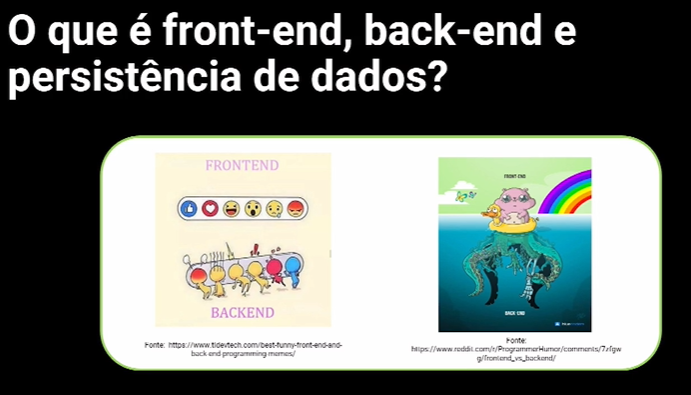
Docker é uma ferramenta de gerenciamento de containers que possibilita a execução de aplicações em ambientes isolados com os recursos mínimos necessários.

A pipeline do GitLab é composta por etapas definidas em um script, permitindo visualizar a execução de cada passo, como build, teste e deploy, com logs detalhados de cada comando.

O Docker oferece uma interface gráfica para gerenciar containers, enquanto o GitLab CI/CD, através de scripts, permite automatizar o processo de desenvolvimento e deploy, criando e gerenciando containers no Docker.

**Front-end,back-end e persistência de dados:**

Front-end, back-end e persistência de dados são componentes essenciais no desenvolvimento web. O front-end representa a interface com o usuário, enquanto o back-end lida com a lógica, regras de negócio e interação com o banco de dados, onde os dados são armazenados de forma persistente.



A interação entre front-end, back-end e banco de dados é crucial. O front-end coleta dados do usuário e os envia para o back-end, que processa, aplica regras de negócio e interage com o banco de dados para armazenar ou recuperar informações. O back-end então retorna os resultados para o front-end, que os exibe ao usuário.

**Integração com API:**

GET para consulta, POST para inclusão, PUT para alteração e DELETE para exclusão de dados.

Através de uma API, é possível manipular os dados do banco de dados H2 sem a necessidade de interação direta com o banco. As requisições HTTP, como GET, POST, PUT e DELETE, enviadas para a API, são refletidas no banco de dados, permitindo a criação, leitura, atualização e exclusão de dados.

Aplicações back-end devem fornecer funcionalidades para manipular o banco de dados através de APIs, ocultando a complexidade da interação direta com o banco de dados. Essa abordagem facilita a manutenção e o desenvolvimento da aplicação, além de garantir a segurança dos dados.

O front-end, back-end e banco de dados foram integrados, demonstrando como o front-end acessa o back-end e como isso se reflete na camada de dados. O site em React utiliza uma API para se comunicar com o back-end, que por sua vez, interage com o banco de dados H2.

O fluxo de dados entre o front-end, back-end e banco de dados foi demonstrado, juntamente com a importância da segurança na arquitetura. A API atua como intermediária entre o front-end e o banco de dados, garantindo que o front-end não acesse o banco de dados diretamente.

No desenvolvimento de aplicações web, a arquitetura em camadas, também conhecida como n-tier architecture, é um modelo amplamente utilizado que organiza os componentes de um sistema em camadas distintas, cada uma com responsabilidades específicas. Essa abordagem oferece diversas vantagens, como a separação de responsabilidades, a reutilização de código e a maior facilidade de manutenção.

Em uma arquitetura em camadas típica, temos três camadas principais: **apresentação, domínio e dados**.

A camada de apresentação, representada pelo front-end, é responsável pela interface com o usuário, ou seja, é a parte visual da aplicação com a qual o usuário interage, como um site ou aplicativo mobile. É nessa camada que são tratados os elementos visuais, a interação do usuário e a navegação.

A camada de domínio, por sua vez, representa o back-end e abriga as regras de negócio, a lógica da aplicação e o processamento das informações. É nessa camada que os dados recebidos do front-end são processados, validados e utilizados para realizar as operações necessárias. O back-end atua como um intermediário entre o front-end e a camada de dados, garantindo a integridade e a segurança das informações.

A camada de dados, como o próprio nome sugere, é responsável por persistir os dados da aplicação, geralmente em um banco de dados. É nessa camada que são realizadas as operações de leitura, escrita, atualização e exclusão de dados. A camada de dados deve ser projetada para garantir a integridade, a consistência e a segurança dos dados armazenados.

A comunicação entre as camadas é fundamental para o funcionamento da aplicação. O front-end se comunica com o back-end por meio de APIs (Application Programming Interfaces), que definem um conjunto de regras e especificações que permitem a comunicação entre diferentes sistemas. O back-end, por sua vez, se comunica com a camada de dados utilizando bibliotecas e frameworks específicos para cada tipo de banco de dados.

Com a evolução da arquitetura de software, o modelo de microserviços surgiu como uma alternativa para lidar com a crescente complexidade das aplicações. Diferentemente da arquitetura monolítica, em que todos os componentes da aplicação estão interligados em um único bloco, os microserviços permitem dividir a aplicação em serviços menores, independentes e autônomos.

Essa abordagem traz diversas vantagens, como a possibilidade de utilizar diferentes tecnologias e linguagens de programação em cada serviço, a escalabilidade independente de cada serviço, a maior facilidade de manutenção e a tolerância a falhas. Cada microserviço é responsável por uma funcionalidade específica da aplicação e se comunica com outros microserviços por meio de APIs.

A segurança é um aspecto crucial em qualquer aplicação, especialmente em arquiteturas em camadas e microserviços. É fundamental garantir que a comunicação entre as camadas seja segura, utilizando mecanismos como autenticação, autorização e criptografia. Além disso, é importante proteger a camada de dados contra acessos não autorizados, utilizando firewalls, senhas fortes e outros mecanismos de segurança.

Em resumo, a arquitetura em camadas e os microserviços são modelos de desenvolvimento de software que oferecem diversas vantagens em termos de organização, escalabilidade, manutenção e segurança. A escolha do modelo mais adequado depende das necessidades específicas de cada aplicação, mas é fundamental compreender os princípios e as características de cada abordagem para tomar a decisão mais acertada.

**Docker:**

Docker é uma ferramenta de gerenciamento de containers que permite a criação de ambientes isolados e portáteis para aplicações. Ele funciona como um hypervisor, gerenciando os recursos da máquina e permitindo a execução de containers com diferentes sistemas operacionais.

Docker permite o gerenciamento de versões de software através de imagens. Cada imagem representa um estado específico da aplicação, permitindo a fácil reversão para versões anteriores. A utilização de containers agiliza o processo de deploy, pois o sistema operacional já está carregado.

O Docker Hub é um repositório online de imagens Docker, contendo desde imagens oficiais de empresas até imagens personalizadas de outros usuários. Ele é usado para baixar imagens e subir containers, agilizando o processo de desenvolvimento e implantação.

Uma imagem Docker contém tudo o que a aplicação precisa para ser executada, incluindo bibliotecas e dependências. Ela é composta por camadas, com uma base (como OpenLiberty e OpenJ9 para aplicações Java), bibliotecas e a aplicação em si no topo.

Imagens Docker são construídas em camadas, com uma base imutável e camadas superiores que podem ser modificadas. A base fornece o ambiente de execução, como um sistema operacional e bibliotecas principais, enquanto as camadas superiores contêm a aplicação e suas dependências específicas.