Semana 19 - Aula 1

Tópico Principal da Aula: Servidores web avançados

Subtítulo/Tema Específico: Integração contínua e deploy contínuo (CI/CD)

Código da aula: [SIS]ANO2C2B3S19A1

Objetivos da Aula:

- Configurar e utilizar pipelines de CI/CD para melhorar o fluxo de desenvolvimento.
- Compreender a importância de CI/CD para reduzir riscos e acelerar entregas.
- Explorar ferramentas populares, como Jenkins e GitHub Actions.
- Aprender a estruturar pipelines para garantir qualidade antes do deploy.

Recursos Adicionais:

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slide 05 e 10 - O que é Integração Contínua e Deploy Contínuo (CI/CD)?

- **Definição:** **A integração contínua (CI) e a entrega/implantação contínua (CD) formam uma prática de desenvolvimento de software que visa automatizar as etapas do ciclo de vida do desenvolvimento, desde a integração do código até a entrega em produção.** O objetivo principal é tornar as entregas mais rápidas e seguras. A
- CI foca na automação da integração do código de vários desenvolvedores em um repositório central, disparando testes automatizados para detectar falhas rapidamente.
- O CD (seja entrega contínua ou deploy contínuo) automatiza a liberação do código validado para um ambiente, que pode ser de teste ou produção.
- **Aprofundamento/Complemento:** **A Integração Contínua (CI) é uma prática que exige que os desenvolvedores integrem o código em um repositório compartilhado várias vezes ao dia. Cada *commit* (envio de código) aciona um *build* automatizado e a execução de testes. Isso permite que as equipes detectem problemas o mais cedo possível. A Entrega Contínua (Continuous Delivery) é o próximo passo, onde as alterações de código aprovadas na fase de CI são automaticamente liberadas para um ambiente de teste. Já o Deploy Contínuo (Continuous Deployment) vai além, implantando automaticamente cada alteração que passa por todo o pipeline diretamente em produção, sem intervenção humana. A escolha entre entrega e deploy contínuo depende do contexto de negócio e dos requisitos do projeto.**
- **Exemplo Prático:** Imagine um time trabalhando em um aplicativo de e-commerce. Um desenvolvedor implementa uma nova funcionalidade de "carrinho de compras" e

envia o código para o repositório (ex: GitHub). Imediatamente, o pipeline de CI/CD é acionado: a ferramenta (ex: Jenkins) compila o código, executa testes unitários para a função do carrinho e testes de integração para garantir que a nova funcionalidade não quebrou outras partes do site. Se tudo passar, o pipeline de CD pode implantar a nova versão em um ambiente de "homologação" para testes manuais. Com o deploy contínuo, se todos os testes automatizados passarem, a alteração poderia ir diretamente para produção para um pequeno subconjunto de usuários (canary release).

Referência do Slide: Slide 10 e 11 - Benefícios e Ferramentas de CI/CD

- **Definição:** A implementação de CI/CD traz benefícios como a detecção precoce de erros, entregas mais rápidas e confiáveis, redução de falhas humanas pela automação, melhoria na colaboração entre desenvolvedores e aumento da qualidade geral do código. Para implementar essas práticas, existem diversas ferramentas, como Jenkins e GitHub Actions. A escolha da ferramenta ideal depende de fatores como a integração com o repositório de código (GitHub, GitLab), facilidade de uso, suporte a diferentes tecnologias, escalabilidade e custo.
- **Aprofundamento/Complemento:** Jenkins é uma das ferramentas de automação de código aberto mais populares e flexíveis, com uma vasta gama de plugins que permitem a integração com praticamente qualquer tecnologia. É conhecido por sua robustez e por ser altamente customizável. GitHub Actions é uma solução mais recente, integrada diretamente ao GitHub, que permite criar workflows de automação diretamente no repositório do projeto usando arquivos de configuração YAML. Sua principal vantagem é a simplicidade e a integração nativa com o ecossistema do GitHub. Outras ferramentas notáveis incluem GitLab CI/CD (similar ao GitHub Actions, mas para o GitLab), CircleCI (conhecida pela sua velocidade e simplicidade na nuvem) e Travis CI.
- **Exemplo Prático: Uma startup decide usar GitHub Actions por sua integração direta com o repositório de código no GitHub. Eles configuram um arquivo .github/workflows/main.yml que define o pipeline. Este arquivo especifica que, a cada *push* para a *branch* main, o GitHub Actions deve: 1) configurar o ambiente com a versão correta do Node.js; 2) instalar as dependências do projeto; 3) rodar os testes automatizados; 4) se os testes passarem, fazer o *build* da aplicação e publicá-la em um serviço de hospedagem como a Vercel ou AWS. Tudo isso sem que o desenvolvedor precise sair da interface do GitHub.**

Semana 19 - Aula 2

Tópico Principal da Aula: Servidores web avançados

Subtítulo/Tema Específico: Implementação de servidores web

Código da aula: [SIS]ANO2C2B3S19A2

Objetivos da Aula:

- Configurar e otimizar servidores web para alta performance.
- Explorar técnicas de cache, compressão e balanceamento de carga.
- Aprender práticas para melhorar a experiência do usuário durante picos de tráfego.

Recursos Adicionais:

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slide 06 e 11 - Otimização de Servidores Web: Cache e Compressão

- **Definição:** **A otimização de servidores web é crucial para garantir alto desempenho e uma boa experiência do usuário. Duas técnicas fundamentais são o cache e a compressão. O cache consiste em armazenar temporariamente cópias de arquivos (como imagens, CSS, JavaScript) para que, em solicitações futuras, eles possam ser entregues mais rapidamente, reduzindo a carga no servidor. A compressão, por sua vez, reduz o tamanho dos arquivos enviados do servidor para o cliente (navegador), o que diminui o tempo de carregamento da página e economiza largura de banda.**
- **Aprofundamento/Complemento:** O cache pode ser implementado em diferentes níveis: no navegador do cliente (browser cache), em uma CDN (Content Delivery Network), ou diretamente no servidor (server-side cache). Para o cache de servidor, ferramentas como Nginx e Varnish são amplamente utilizadas para armazenar conteúdo estático ou até mesmo respostas dinâmicas. Já a compressão é geralmente realizada com algoritmos como Gzip ou Brotli. Os servidores web, como Nginx e Apache, podem ser configurados para comprimir automaticamente os tipos de arquivo especificados antes de enviá-los ao navegador, que então os descomprime. O Brotli, desenvolvido pelo Google, oferece taxas de compressão superiores ao Gzip, mas é importante garantir que os navegadores dos usuários o suportem.
- **Exemplo Prático: O administrador de um portal de notícias configura o servidor Nginx. Para o cache, ele adiciona uma diretiva `expires max;` para todos os arquivos de imagem (JPG, PNG), fazendo com que o navegador do visitante armazene essas imagens por um longo período. Para a compressão, ele habilita o Gzip no arquivo de configuração do Nginx (`nginx.conf`) para comprimir arquivos de texto, como HTML, CSS e JavaScript. Como resultado, quando um usuário acessa o site, as imagens são carregadas do cache local**

do navegador (após a primeira visita) e os textos chegam de forma mais rápida por serem menores, resultando em um carregamento de página visivelmente mais ágil.

Referência do Slide: Slide 06 e 11 - Balanceamento de Carga e Escolha do Servidor (Nginx vs Apache)

- **Definição: O balanceamento de carga é uma técnica utilizada para distribuir o tráfego de rede ou de aplicações entre múltiplos servidores.** Isso previne que um único servidor fique sobrecarregado, aumentando a disponibilidade, a performance e a confiabilidade do serviço. Na escolha de um servidor web, Nginx e Apache são as opções mais populares. A decisão entre eles depende de fatores como necessidade de desempenho, suporte a módulos específicos e facilidade de configuração.
- **Aprofundamento/Complemento:** O Nginx é conhecido por sua arquitetura orientada a eventos, que lida com um grande número de conexões simultâneas de forma muito eficiente e com baixo consumo de memória. Ele se destaca como um excelente proxy reverso, balanceador de carga e servidor para conteúdo estático. O Apache, por outro lado, tem uma arquitetura baseada em processos ou threads e é extremamente flexível devido ao seu sistema de módulos dinâmicos, como o
- .htaccess, que permite configurações por diretório. Enquanto o Nginx geralmente supera o Apache no serviço de conteúdo estático e em cenários de alta concorrência, o Apache pode ser mais fácil de configurar para iniciantes e possui um ecossistema de módulos muito maduro.
- **Exemplo Prático: Um site de e-commerce espera um grande volume de acessos durante a Black Friday. Para evitar que o site saia do ar, a equipe de TI implementa um balanceador de carga. Eles configuram o Nginx como balanceador de carga na frente de três servidores de aplicação idênticos (servidores "backend"). Quando um cliente acessa o site, a requisição chega primeiro ao Nginx, que a redireciona para um dos três servidores backend, utilizando um algoritmo como o *round-robin* (distribuição sequencial). Se um dos servidores backend falhar, o Nginx para de enviar tráfego para ele, garantindo que os usuários continuem sendo atendidos pelos outros dois servidores**.

Semana 19 - Aula 3

Tópico Principal da Aula: Servidores web avançados

Subtítulo/Tema Específico: Segurança em banco de dados

Código da aula: [SIS]ANO2C2B3S19A3

Objetivos da Aula:

- Implementar práticas de segurança para proteger dados sensíveis em bancos de dados.
- Explorar técnicas de criptografia para proteger dados sensíveis.
- Compreender o controle de acesso baseado no princípio do menor privilégio.
- Aprender a monitorar e auditar atividades em bancos de dados.

Recursos Adicionais:

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slide 06 e 16 - Criptografia de Dados e Princípio do Menor Privilégio

- **Definição:** A segurança em bancos de dados é fundamental para proteger informações. **A criptografia é uma técnica essencial que torna os dados ilegíveis para pessoas não autorizadas, protegendo-os tanto "em repouso" (armazenados no disco) quanto "em trânsito" (durante a comunicação pela rede). Outro pilar da segurança é o princípio do menor privilégio, que consiste em conceder aos usuários e sistemas apenas as permissões estritamente necessárias para realizarem suas tarefas, minimizando o risco de acessos indevidos e danos em caso de uma conta ser comprometida.**
- **Aprofundamento/Complemento:** A criptografia de dados em repouso pode ser aplicada em vários níveis: no arquivo do banco de dados (TDE - Transparent Data Encryption), em colunas específicas ou até mesmo no nível da aplicação antes de salvar os dados. Para dados em trânsito, é comum o uso de protocolos como TLS/SSL para proteger a conexão entre a aplicação e o servidor de banco de dados. O princípio do menor privilégio (PoLP - Principle of Least Privilege) é uma abordagem proativa de segurança. Em vez de dar acesso amplo e remover permissões conforme necessário, o acesso é negado por padrão e liberado apenas para funções específicas. Isso se aplica a contas de usuários, serviços e aplicações que acessam o banco.
- **Exemplo Prático:** **Uma fintech armazena dados de clientes, incluindo CPF e informações financeiras. Para proteger esses dados, a coluna do CPF na tabela de clientes é criptografada no nível da aplicação. Além disso, a conexão entre o servidor da aplicação e o banco de dados é feita via TLS. Para o controle de acesso, foi criado um usuário no banco de dados específico para o serviço de relatórios. Este usuário tem permissão apenas de leitura (SELECT)**

em tabelas não sensíveis. Ele não pode alterar (UPDATE), inserir (INSERT) ou deletar (DELETE) dados, nem acessar tabelas com informações de cartão de crédito, aplicando assim o princípio do menor privilégio.

Referência do Slide: Slide 06 e 16 - Auditoria e Prevenção de Acessos não Autorizados

- **Definição:** **Um sistema de auditoria eficiente envolve o registro (log) de todas as atividades importantes no banco de dados, como tentativas de login, consultas e modificações de dados. A configuração de alertas para atividades suspeitas ou incomuns é parte crucial desse processo. Para prevenir acessos não autorizados, práticas como a autenticação multifator (MFA), o monitoramento constante e a revisão periódica das permissões de usuário são altamente recomendadas.**
- **Aprofundamento/Complemento:** A auditoria não serve apenas para detectar violações, mas também para investigações forenses após um incidente e para cumprir regulamentações (como a LGPD no Brasil). Os logs de auditoria devem registrar "quem, o quê, quando e onde" para cada ação relevante. O monitoramento constante pode ser automatizado com ferramentas que analisam os logs em tempo real e usam inteligência para identificar padrões anômalos (por exemplo, um usuário acessando o banco de dados de um local incomum ou fora do horário de trabalho). A revisão periódica de privilégios, também chamada de recertificação de acesso, garante que permissões antigas e desnecessárias sejam revogadas, combatendo o "acúmulo de privilégios".
- **Exemplo Prático:** **Um hospital utiliza um sistema de banco de dados para gerenciar prontuários eletrônicos. A equipe de segurança configura a auditoria para registrar todas as leituras e modificações na tabela de pacientes. Eles configuram um alerta que dispara sempre que um único usuário acessa mais de 100 prontuários em uma hora. Além disso, para o acesso administrativo ao banco, é exigido um segundo fator de autenticação (MFA) via aplicativo no celular. A cada três meses, um gerente revisa a lista de usuários com acesso ao banco e suas permissões, removendo contas de funcionários que já saíram da empresa e ajustando privilégios conforme as mudanças de função.**
