Semana 12

Linguagens de programação back-end

Ferramentas avançadas para desenvolvimento back-end

Aula 1: Docker e containers

Código da aula: [SIS]ANO2C2B2S12A1

Containers são uma tecnologia de virtualização no nível do sistema operacional que permite empacotar e isolar aplicações com todas as suas dependências (código, bibliotecas, ferramentas e arquivos de configuração). Isso garante que a aplicação funcione de forma consistente em diferentes ambientes de computação.

- **Temas e Subtemas:**
  - **Conceito de Containers:**
    - Isolamento de processos e recursos.
    - Leveza e rapidez em comparação com máquinas virtuais tradicionais.
  - **Benefícios dos Containers:**
    - **Portabilidade:** Executam de forma idêntica em qualquer ambiente que suporte containers.
    - **Eficiência:** Menor consumo de recursos (CPU, memória) do que VMs.
    - **Escalabilidade:** Facilidade para escalar aplicações horizontalmente.
    - **Consistência:** Ambientes de desenvolvimento, teste e produção idênticos.
    - **Agilidade:** Ciclos de desenvolvimento e deployment mais rápidos.
  - **Ferramentas Populares:**
    - **Docker:** Plataforma líder para criação, gerenciamento e execução de containers.
      - *Exemplo:* Um desenvolvedor usa um `Dockerfile` para definir a imagem de sua aplicação Node.js, garantindo que todas as dependências estejam incluídas. Essa imagem pode ser executada em qualquer máquina com Docker.
    - **Kubernetes (K8s):** Sistema de orquestração de containers para automatizar a implantação, o dimensionamento e a gestão de aplicações em containers.
      - *Exemplo:* Uma empresa usa Kubernetes para gerenciar múltiplos containers de seus microserviços, garantindo alta disponibilidade e balanceamento de carga.

        - **Podman, containerd, CRI-O:** Outras ferramentas e runtimes de container.
    - **Aplicações e Casos de Uso:**
        - **Microserviços:** Arquitetura onde cada serviço é um container independente.
        - **Aplicações Web:** Empacotamento de servidores web e suas dependências.
        - **Ambientes de Desenvolvimento:** Criação de ambientes padronizados para equipes de desenvolvimento.
        - **CI/CD (Integração Contínua/Entrega Contínua):** Utilização de containers para construir, testar e implantar aplicações de forma automatizada.
        - **Big Data e Machine Learning:** Empacotamento de ferramentas e modelos para processamento e análise de dados.

Link de Vídeo

Sugestão de vídeo para aprofundamento:

1. **O mínimo que você precisa saber sobre Docker!** https://youtu.be/ntbpIfS44Gw?si=X4Uur3Hb_VkwutEp

2. **Docker para Iniciantes** - Link: https://www.youtube.com/watch?v=3c-iBn73dDE (Full Cycle)

Semana 12

Linguagens de programação back-end

Ferramentas avançadas para desenvolvimento back-end

Aula 2: Integração contínua (CI) e a entrega contínua (CD)

Código da aula: [SIS]ANO2C2B2S12A2

Integração Contínua (CI) é a prática de automatizar a integração de alterações de código de múltiplos contribuidores em um único projeto de software. Isso envolve a compilação e teste automatizados sempre que uma nova alteração é enviada ao repositório principal. **Entrega Contínua (CD)** expande a CI, automatizando a liberação de software validado para um ambiente (como produção ou homologação) após a fase de CI. Em alguns casos, pode se referir a **Implantação Contínua**, onde cada alteração aprovada é automaticamente implantada na produção.

- **Temas e Subtemas:**
  - **Integração Contínua (CI):**
    - **Conceito:** Fusão frequente de código em um repositório central.
    - **Processo:**
      1. Desenvolvedores fazem commit das alterações.
      2. Servidor de CI detecta a alteração.
      3. Código é compilado (build).
      4. Testes automatizados são executados (unitários, integração).
      5. Feedback rápido para o desenvolvedor.
    - **Benefícios:** Detecção precoce de erros, melhor colaboração, software mais estável.
    - *Exemplo:* Um desenvolvedor envia um novo código para o GitLab. O GitLab CI/CD automaticamente inicia um pipeline que compila o código e executa testes unitários. Se algum teste falhar, o desenvolvedor é notificado imediatamente.
  - **Entrega Contínua (CD):**
    - **Conceito:** Automatização da liberação do software para ambientes.
    - **Processo (pós-CI):**
      1. Se a CI for bem-sucedida, o artefato (pacote de software) é gerado.
      2. O artefato é implantado automaticamente em ambientes de teste/homologação.
      3. Testes de aceitação e outros testes de maior nível podem ser executados.
      4. A implantação em produção pode ser um passo manual ou automatizado (Implantação Contínua).
    - **Benefícios:** Releases mais rápidos e frequentes, menor risco nas implantações, feedback do usuário mais rápido.
    - *Exemplo:* Após a aprovação dos testes em CI, uma pipeline do Jenkins automaticamente implanta a nova versão da aplicação em um ambiente de homologação. Após a validação manual pela equipe de QA, a mesma pipeline pode ser acionada para implantar em produção.
  - **Ferramentas Comuns de CI/CD:**
    - **Jenkins:** Servidor de automação open-source altamente extensível.
    - **GitLab CI/CD:** Funcionalidade de CI/CD integrada à plataforma GitLab.
    - **GitHub Actions:** Automação de workflows diretamente no GitHub.

        - **Azure DevOps, CircleCI, Travis CI:** Outras plataformas populares.
    - **Cultura DevOps:**
        - CI/CD são pilares fundamentais da cultura DevOps, promovendo colaboração, automação e feedback rápido.

Vídeos Sugeridos – Aula 02

1. **CI/CD (Conceitos e Fundamentos) // DevOps** - Link: https://youtu.be/ZX0L6fHcBb4?si=RbtzqlbCmwJM92MQ
2. **O que é CI/CD? Integração e Entrega Contínua Explicado!** - Link: https://youtu.be/AZtTd3pFVTY?si=3T9VHLR3nWsgTMyf (Código Fonte TV)

Semana 12

Linguagens de programação back-end

Ferramentas avançadas para desenvolvimento back-end

Aula 3: Ferramentas de monitoramento

Código da aula: [SIS]ANO2C2B2S12A3

Ferramentas de monitoramento são essenciais para observar a saúde, o desempenho e a disponibilidade de sistemas de TI, aplicações e infraestrutura. Elas coletam dados, apresentam informações de forma compreensível (dashboards, alertas) e ajudam a identificar e diagnosticar problemas proativamente.

- **Temas e Subtemas:**
    - **Importância do Monitoramento:**
        - Garantir a disponibilidade e o desempenho dos serviços.
        - Detectar problemas antes que afetem os usuários.
        - Facilitar a resolução de incidentes (troubleshooting).
        - Planejamento de capacidade e otimização de recursos.
        - Obter insights sobre o comportamento do sistema e do usuário.
    - **Tipos de Monitoramento (Pilares da Observabilidade):**
        - **Métricas (Metrics):** Dados numéricos coletados ao longo do tempo (ex: uso de CPU, tempo de resposta, número de erros).
            - *Exemplo:* Monitorar a utilização da CPU de um servidor a cada minuto.

        - **Logs:** Registros de eventos que ocorrem em sistemas e aplicações. Contêm informações detalhadas sobre atividades e erros.
            - *Exemplo:* Logs de um servidor web registrando cada requisição recebida, incluindo o status HTTP e o IP do cliente.
        - **Rastreamento Distribuído (Distributed Tracing):** Acompanhamento de uma requisição através de múltiplos serviços em uma arquitetura de microserviços.
            - *Exemplo:* Visualizar o caminho completo e o tempo gasto em cada microserviço para uma única transação de compra online.
    - **Principais Categorias de Ferramentas:**
        - **Monitoramento de Infraestrutura:** Foco em servidores, redes, armazenamento (Nagios, Zabbix, Prometheus).
        - **Monitoramento de Performance de Aplicação (APM):** Foco no desempenho e erros de aplicações (Dynatrace, New Relic, Datadog APM).
        - **Gerenciamento de Logs:** Coleta, armazenamento e análise de logs (Elastic Stack - Elasticsearch, Logstash, Kibana; Splunk, Grafana Loki).
        - **Visualização e Alerta:** Criação de dashboards e configuração de alertas (Grafana, Kibana, Prometheus Alertmanager).
    - **Ferramentas Populares e Exemplos:**
        - **Prometheus:** Coleta de métricas, especialmente bom para ambientes Kubernetes.
            - *Exemplo:* Usar Prometheus para coletar métricas de uso de memória dos containers e Grafana para visualizar esses dados em um dashboard.
        - **Grafana:** Plataforma de visualização e analytics, frequentemente usada com Prometheus, Elasticsearch, etc.
            - *Exemplo:* Criar um dashboard no Grafana que exibe o tempo de resposta de uma API, a taxa de erros e o número de requisições por segundo.
        - **Zabbix/Nagios:** Soluções tradicionais e robustas para monitoramento de infraestrutura e serviços.
            - *Exemplo:* Configurar o Zabbix para alertar a equipe de TI quando o espaço em disco de um servidor crítico estiver abaixo de 10%.
        - **Datadog/New Relic:** Plataformas SaaS abrangentes que oferecem APM, monitoramento de infraestrutura, logs, etc.
            - *Exemplo:* Utilizar o Datadog para monitorar a performance de uma aplicação web, rastrear transações

entre microserviços e analisar logs de erro em um só lugar.

- **Elastic Stack (ELK/EFK):** Para coleta, busca e visualização de logs (Elasticsearch, Logstash/Fluentd, Kibana).
  - *Exemplo:* Enviar logs de todas as aplicações para o Elasticsearch e usar o Kibana para pesquisar erros específicos ou criar visualizações sobre a frequência de determinados eventos.

Vídeos Sugeridos – Aula 03

1. **O que é Prometheus?** Link: https://youtu.be/glAT7303Dss?si=N7VvoDqWuDU2nghu