**Guia de Estudos – Prova 1 de CI/CD**

1. **Controle de Versão de Código**

**Integração Contínua (CI)** é a prática de desenvolvimento onde os desenvolvedores integram as suas alterações de código a um repositório compartilhado (como o Git) de forma frequente. Cada integração aciona um *pipeline* de build e testes automatizado. O objetivo é encontrar e corrigir problemas de integração rapidamente.

O **Git** é a base de todo o processo de CI/CD. Ele é o sistema que rastreia e gerencia todas as mudanças no código do seu projeto.

* **Ferramenta principal:** Git
* **Objetivo:** Manter histórico de alterações, permitir colaboração, rastrear mudanças no código.

**Principais Conceitos do Git**

| **Comando** | **Função** | **Exemplo** |
| --- | --- | --- |
| git init | Cria um novo repositório local | git init projeto |
| git clone <url> | Copia um repositório remoto | git clone https://github.com/user/projeto.git |
| git status | Mostra estado dos arquivos | git status |
| git add <arquivo> | Adiciona arquivo ao **stage** | git add index.html |
| git commit -m "msg" | Salva alteração no histórico | git commit -m "Corrige bug de login" |
| git push | Envia commits locais para o repositório remoto | git push origin main |
| git pull | Atualiza seu repositório local | git pull origin main |
| git log | Histórico de commits | git log --oneline |
| git diff | Mostra diferenças entre versões | git diff HEAD~1 |
| git tag <nome> | Cria marcação (release) | git tag v1.0.0 |

⚠️ **Atenção:**

* Não renomear arquivos manualmente → o Git registra como exclusão + criação, o que pode causar conflitos.
* Para renomear corretamente: git **mv** antigo.txt novo.txt

#### Renomeação e Refatoração de Código

O Git é inteligente, mas não tem um comando nativo para "renomear". Ele rastreia arquivos pelo seu conteúdo. Quando você renomeia um arquivo, ele vê isso como uma **exclusão do arquivo antigo** e a **adição de um arquivo novo** com o mesmo conteúdo. Isso pode levar a erros e dificultar a análise do histórico.

* **Solução:** Sempre use o comando git mv <nome\_antigo> <novo\_nome>. Isso informa ao Git que você moveu ou renomeou um arquivo, e ele mantém o histórico completo.
* **Refatoração de Código:** O Git lida com a refatoração rastreando as alterações linha por linha, o que permite que você veja como a estrutura do código foi mudada sem alterar seu comportamento.

1. **Controle de Configuração e Versionamento Semântico**

As **tags** no Git são como marcadores que você pode anexar a um *commit* específico, geralmente para marcar um ponto importante na história do projeto, como o lançamento de uma nova versão. O **Versionamento Semântico** é um sistema para definir o número das versões, no formato **x.y.z**:

* **Formato:** x.y.z
  + **x (major):** mudança **incompatível** (quebra de compatibilidade).
  + **y (minor):** adição de **nova funcionalidade**, mantendo compatibilidade.
  + **z (patch):** correção de **bugs ou melhorias pequenas**.

Exemplo:

* 1.0.0 → versão inicial estável.
* 1.1.0 → adiciona funcionalidade.
* 1.1.2 → corrige bug sem quebrar nada.
* 2.0.0 → reestruturação que quebra compatibilidade.

1. **Integração Contínua**

### Conceito

* Prática de integrar código frequentemente (várias vezes ao dia).
* Cada integração dispara **build automático + execução de testes**.
* Objetivo: detectar falhas cedo, antes que cheguem à produção.

### Fluxo da CI

1. Dev altera código → git commit → git push.
2. Servidor de CI detecta alteração.
3. Roda build (compilação, análise de qualidade).
4. Executa testes automatizados.
5. Gera relatórios/logs.
6. Caso falhe → dev precisa corrigir.
7. **Testes Automatizados e Fluxo de Testes**

Os testes automatizados são a espinha dorsal da Integração Contínua. A cada *push* para o repositório, o sistema de CI executa uma bateria de testes para validar se o novo código quebra a funcionalidade existente.

* **Por que testar na integração?**
  + Evitar falhas em produção.
  + Garantir compatibilidade entre módulos.

**Tipos de Testes**

* **Teste de Unidade:** valida pequenas partes isoladas (funções, métodos).
* **Teste de Integração:** verifica se módulos funcionam juntos.
* **Teste de Aceitação:** valida se o sistema atende requisitos do usuário.
* **Homologação:** teste em ambiente similar ao de produção, antes da liberação.

| **Tipo** | **O que verifica?** | **Exemplo** |
| --- | --- | --- |
| Unidade | Funções/métodos isolados | Testar função de soma |
| Integração | Comunicação entre módulos | API + Banco de dados |
| Aceitação | Requisitos do usuário | Tela de login funcionando |
| Homologação | Ambiente próximo da produção | Simulação antes de liberar |

⚙️ **Automatização:**

* Sempre que houver alteração → testes devem rodar automaticamente.
* Se não for 100% automatizado, registrar os erros via **logs**.

**5. Ferramentas de CI/CD**

### Instaláveis

* **Jenkins**: muito usado, altamente configurável.

### Nuvem

* **GitHub Actions**: integrado ao GitHub.
* **GitLab CI/CD**: pipelines nativas.
* **Travis CI, CircleCI, Azure DevOps**: outras opções.

**6. Empacotamento e Deploy**

* **Objetivo:** entregar software em formato distribuível (executável ou container).

**Opções de empacotamento**

* **Binários:** .exe, .jar, etc.
* **Containers:** imagens Docker → executáveis portáveis.

**7. Docker e Kubernetes**

O **Docker** é uma ferramenta de virtualização leve que usa **contêineres** para empacotar uma aplicação e todas as suas dependências.

* + Criação de ambientes isolados (containers).
  + Utilizado em desenvolvimento, testes, homologação e produção.
  + **Vantagens:**
    - Economia de espaço (compartilha camadas).
    - Rapidez de execução.
    - Consistência entre ambientes.
  + **Atenção:** dados que precisam ser persistidos devem estar fora do container (volumes).

#### Por que o Docker é Essencial?

* **Consistência:** Um contêiner garante que a aplicação rode exatamente da mesma forma em qualquer ambiente: na máquina do desenvolvedor, no servidor de testes e na produção.
* **Eficiência:** Contêineres são leves e usam recursos do sistema operacional hospedeiro de forma inteligente. Eles aproveitam e reciclam camadas (partes do código e dependências) para economizar espaço em disco e acelerar o tempo de *build*.
* **Isolamento:** Cada contêiner é isolado, então uma aplicação não interfere na outra, mesmo que usem dependências diferentes.
* **Volumes:** Como os contêineres são efêmeros (podem ser destruídos e recriados), os dados que precisam ser persistentes (como em um banco de dados) devem ser armazenados em **volumes** que ficam fora do contêiner.

#### Kubernetes: O Orquestrador de Contêineres

Enquanto o Docker lida com contêineres individuais, o **Kubernetes** é uma plataforma que gerencia um grande número de contêineres. Ele automatiza o deploy, o escalonamento e o gerenciamento, garantindo que a sua aplicação esteja sempre disponível e funcione com alta performance.

* **Kubernetes:**
  + Sistema de **orquestração de containers**.
  + Gerencia múltiplos containers, escalabilidade, balanceamento de carga.

**8. QA e Qualidade de Software**

### Ferramentas

* **QArun**: ferramenta de testes automatizados.
* Outras: Selenium, JUnit, PyTest.

### 8.2. Prática

* Sempre validar código antes da entrega.
* Automatizar testes sempre que possível.
* Caso não seja 100% automatizado → documentar erros via **logs**.