

1- CD Entrega Contínua ou Implantação Contínua Ambos são estágios que sucedem a **Integração Contínua (CI)**.

- **CI (Integração Contínua):** É uma prática de desenvolvimento de software onde os desenvolvedores mesclam suas alterações de código em um repositório central com frequência. Um build automatizado é então executado para verificar se as alterações funcionam corretamente. O objetivo principal é detectar e resolver problemas de integração precocemente.
- **CD (Entrega Contínua - *Continuous Delivery*):** É uma extensão do CI. Garante que o software possa ser liberado para produção a **qualquer momento**, pois cada alteração de código que passa pelos pipelines de build e teste é automaticamente enviada para um ambiente *stage* (preparação) ou de controle de qualidade (QA). A decisão de *implantar* o software para produção é feita por uma pessoa (manual).
- **CD (Implantação Contínua - *Continuous Deployment*):** É um passo além da Entrega Contínua. Aqui, as alterações de código que passaram por todos os estágios de teste são **automaticamente implantadas** no ambiente de produção, sem intervenção humana.


Relação com CI

A **Entrega Contínua (CD)** e a **Implantação Contínua (CD)** dependem da **Integração Contínua (CI)**. A CI é o alicerce do pipeline de CI/CD.

- Primeiro, você integra e testa o código (**CI**).
- Em seguida, você garante que este código testado esteja pronto para liberação (**Entrega Contínua**).
- Por fim, você pode optar por automatizar essa liberação (**Implantação Contínua**).

2- A **Entrega Contínua (CD)** traz uma série de benefícios significativos para o processo de desenvolvimento e entrega de software:

- **Lançamentos Mais Rápidos e Frequentes:** O código está sempre em um estado liberável, o que permite que novas funcionalidades e correções de bugs cheguem aos usuários muito mais rapidamente.
- **Risco de Lançamento Reduzido:** Como o software é liberado em pequenos incrementos, o impacto de qualquer problema é limitado. O processo de lançamento em si se torna uma rotina de baixo risco.
- **Deteção e Correção de Erros Mais Rápida:** O *feedback loop* apertado, construído com testes automatizados, garante que os erros sejam detectados logo após serem introduzidos, tornando-os mais fáceis e baratos de corrigir.
- **Maior Foco no Cliente:** Equipes podem responder mais rapidamente ao *feedback* do cliente, iterando e entregando valor continuamente.
- **Melhor Colaboração:** A automação e a clareza do pipeline de entrega forçam o alinhamento e a melhor comunicação entre os times de desenvolvimento, QA e operações (práticas de **DevOps**).

-  **Custos Reduzidos:** A automação de tarefas manuais de build, teste e implantação libera o tempo dos desenvolvedores para se concentrarem em escrever código e gerar valor.

Gostaria de saber quais **ferramentas** são comumente usadas para implementar um pipeline de CI/CD, como Jenkins ou GitLab CI?

4- Qual é a principal diferença prática entre CI e CD?

A principal diferença prática está no **destino** e na **intervenção humana** no final do processo:

Característica	CI (Integração Contínua)	CD (Entrega/Implantação Contínua)
Objetivo Principal	Manter o código base sempre estável e testável.	Garantir que o código estável esteja disponível para uso (em Staging ou Produção).
Destino do Artefato	O código integrado (artefato <i>buildado</i>) vai para um repositório (ex: Nexus, Artifactory).	O artefato <i>buildado</i> é implantado em um ambiente real (ex: QA, Staging, Produção).
Intervenção Humana	Mínima (foco na automação do <i>build</i> e teste unitário).	Define a diferença entre os dois tipos de CD:
		➡ Entrega Contínua: Requer uma aprovação manual para ir para a Produção.
		➡ Implantação Contínua: Não requer intervenção humana; o <i>deploy</i> é totalmente automático.

que aconteceria se o teste falhasse antes do deploy?

Se um teste falhar em qualquer ponto do pipeline de CI/CD, o processo é **interrompido imediatamente**.

1. **Parada Imediata:** O *pipeline* para de rodar, impedindo que o código defeituoso avance para os próximos estágios, como o *deploy* em um ambiente de homologação ou produção.
2. **Notificação da Equipe:** Os responsáveis (desenvolvedores, *scrum master*, ou *on-call*) são notificados imediatamente sobre a falha (geralmente por e-mail, Slack ou outra ferramenta de comunicação).
3. **Identificação da Causa:** A mensagem de falha e os *logs* do teste são usados para identificar qual teste falhou e, crucialmente, qual foi a **última mudança de código** que causou o problema.
4. **Correção Rápida (Rollback ou Fix):** A equipe deve priorizar a correção da falha. Como a CI exige integrações pequenas e frequentes, é fácil identificar o autor e reverter a pequena mudança (ou aplicar uma correção rápida) para que o *pipeline* volte a ficar **verde** (bem-sucedido) o mais rápido possível.

O principal objetivo é **falhar rapidamente** (*fail fast*) e evitar que a falha se propague.

Como a entrega contínua aumenta a confiança do time no processo?

A Entrega Contínua (CD) aumenta drasticamente a confiança do time por meio de três pilares: **Previsibilidade, Visibilidade e Repetição**.

1. **Automação e Repetição (Rotina):** O processo de *release* deixa de ser um evento raro, estressante e manual para se tornar uma **rotina automatizada e previsível**. A repetição constante elimina o medo do "o que pode dar errado desta vez?".
2. **Visibilidade e Feedback Imediato:** O *pipeline* fornece uma visão clara do estado da aplicação em tempo real. Se o código passar por todos os testes e chegar ao ambiente de homologação, o time sabe com alta confiança que ele está **pronto**. O *feedback* instantâneo dos testes automatizados elimina a incerteza.
3. **Redução da Ansiedade na Implantação:** A CD garante que o *deploy* em si é um processo simples, de baixo risco, muitas vezes com o apertar de um botão (no caso de Entrega Contínua) ou totalmente automático (no caso de Implantação Contínua). Como as mudanças são pequenas e testadas, o estresse associado a grandes lançamentos ("*big bang deployments*") desaparece.