# Otimização de CI/CD para Aplicações NestJS

05/03/2025

# Sumário



# Contexto

# O que é CI/CD?

CI/CD (Continuous Integration/Continuous Deployment) é um conjunto de práticas que visa automatizar e otimizar o desenvolvimento de software, garantindo que novos códigos sejam integrados e implantados de maneira eficiente e segura.

- **Continuous Integration (CI)**: Refere-se à prática de integrar frequentemente as mudanças de código ao repositório principal. Isso permite que problemas sejam detectados e corrigidos rapidamente, pois cada alteração passa por testes automatizados antes de ser mesclada.
- **Continuous Deployment (CD)**: É a extensão natural da CI, garantindo que as mudanças aprovadas sejam automaticamente implantadas nos ambientes de produção ou de testes, reduzindo a intervenção manual e aumentando a confiabilidade das entregas.



# Benefícios do CI/CD

Implementar um pipeline de CI/CD traz diversos benefícios, entre eles:

- **Maior agilidade no desenvolvimento**: O feedback rápido sobre novas implementações reduz o tempo entre a escrita do código e sua disponibilização.
- **Detecção antecipada de erros**: Testes automatizados ajudam a identificar problemas antes que cheguem à produção.
- **Menos retrabalho**: Como os erros são encontrados mais cedo, é mais fácil e barato corrigi-los.
- **Implantações mais seguras e frequentes**: Reduzindo riscos e melhorando a confiabilidade dos sistemas.
- **Automatização de processos repetitivos**: Permitindo que a equipe foque em desenvolvimento ao invés de tarefas operacionais.



# Estrutura do Projeto

Nosso projeto possui uma arquitetura baseada em microsserviços e conta com quatro ambientes distintos:

- **Desenvolvimento**: Ambiente utilizado pelos desenvolvedores para testes e implementação de novas funcionalidades.
- **Homologação**: Utilizado pelos QAs (Quality Assurance) para validação de novas funcionalidades antes da liberação.
- **Demonstração**: Um ambiente semelhante à produção, usado como Sandbox por nossos parceiros.
- **Produção**: O ambiente final onde a aplicação é utilizada pelos usuários.



# Estrutura do Projeto

Nosso pipeline de CI/CD é dividido em duas partes principais:

1. **CI de MRs/PRs**: Responsável por validar a integridade do código antes de ser mesclado na branch do ambiente.
2. **CI/CD da branch do ambiente**: Valida novamente o código e realiza a implantação no ambiente correspondente.

HotFixes são aplicados diretamente em produção, validados e depois propagados para os outros ambientes.

**Nota:** MR (Merge Request) é um processo utilizado em plataformas como GitLab para solicitar a integração de um conjunto de alterações em uma branch principal. É similar ao PR (Pull Request) utilizado no GitHub.



# Tecnologias e Arquitetura Utilizadas

O projeto segue uma arquitetura de **microsserviços**, onde cada serviço é responsável por uma funcionalidade específica e se comunica com outros serviços por meio de APIs e mensageria.

A otimização realizada neste contexto foi aplicada a um dos **microsserviços críticos da operação**, que desempenha um papel essencial no funcionamento do sistema. As tecnologias principais utilizadas incluem:

- **NestJS** para a aplicação backend.
- **GitLab** como ferramenta de CI/CD e versionamento de código.
- **Jest** para testes unitários e de integração.
- **Docker** para conteinerização e gerenciamento de dependências.

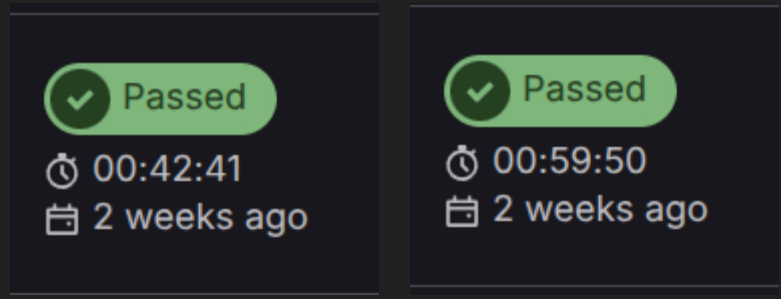


# O Problema

# O Problema

Nosso principal problema era o tempo elevado para que uma correção crítica chegasse em produção após a validação do QA. Somente o tempo de execução dos pipelines de CI/CD levava em torno de **4 horas**, distribuídas da seguinte forma:

- **~40 minutos** para a MR de demonstração.
- **~1 hora** para a branch de demonstração.
- **~40 minutos** para a MR de produção.
- **~1 hora** para a branch de produção.



# Melhorias Implementadas

# Padronização de Mocks

O uso de diferentes abordagens para realizar mocks nos testes resultava em inconsistências e dificuldades na manutenção. Além disso, essa despadronização ocasionava erros intermitentes devido a mocks sendo gerados de forma errada, o que quebrava os testes automatizados. Como consequência, era necessário rodar a mesma job inúmeras vezes até obter uma execução bem-sucedida. Para resolver essa questão, padronizamos o uso da biblioteca **Rosie**, tornando os testes mais consistentes e fáceis de gerenciar.

```
export const agentConfigFactory = Factory.define<AgentConfigEntity>(
  'AgentConfigEntity',
)
  .attrs({
    id: () => faker.string.uuid(),
    ...props
  })
  .after(toInstance(AgentConfigEntity));
```



# Testes Unitários sem Dependências Externas

Nos testes unitários, utilizávamos **Docker e Docker Compose** para rodar dependências como **Redis, Postgres e Localstack**. No entanto, essas dependências não eram necessárias para o contexto dos testes unitários, gerando um overhead desnecessário.

A solução foi remover essa dependência nos testes unitários e substituir por mocks, reduzindo significativamente o tempo de execução.



# Testes E2E com Shard e Parallel

Os testes de E2E (End-to-End) são os mais demorados no pipeline. Para melhorar a performance, utilizamos duas estratégias: **sharding** no Jest e **parallel** no GitLab CI.



# O que é Shard no Jest?

Sharding no Jest permite dividir a execução dos testes em diferentes processos, distribuindo a carga de trabalho e reduzindo o tempo total de execução.

**Exemplo de uso do Shard no Jest:**

```
jest --shard=1/3
```

No exemplo acima, estamos executando o primeiro de três shards, ou seja, dividimos os testes em três partes e estamos rodando apenas uma delas.

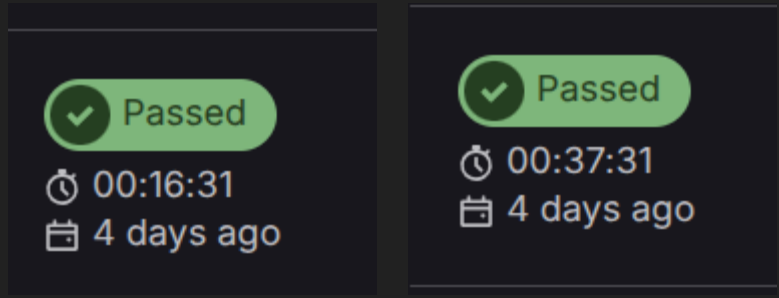


# Resultados Obtidos

# Resultados Obtidos

Após a implementação das otimizações, os tempos de execução do CI/CD foram significativamente reduzidos:

- **CI de MRs/PRs** passou de ~40 minutos para **15 minutos**.
- **CI/CD da branch do ambiente** passou de ~1 hora para **35 minutos**.



# Conclusão

# Referências

Jest Shard



Slides · GitHub

# Muito obrigado!

Se ficou com dúvidas a respeito de uma ou mais partes deste documento, não hesite em entrar em contato conosco.

zrp.com.br
luciano.weslen@zrp.com.br