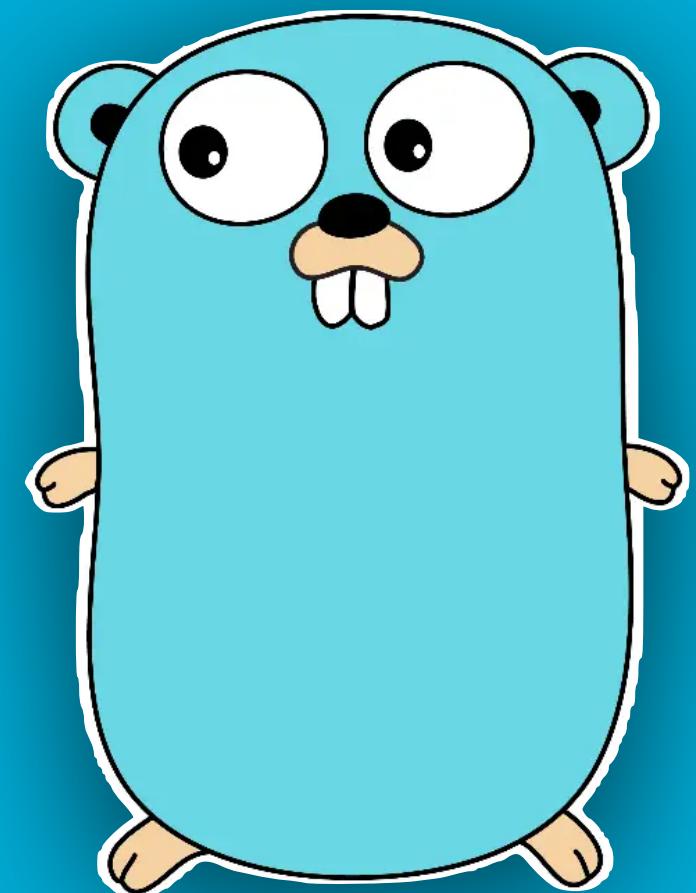


Gemini

Integrando com Golang



Guilherme Silva

Software Engineer at **ReclameAQUI**



Formação

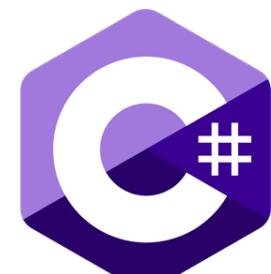
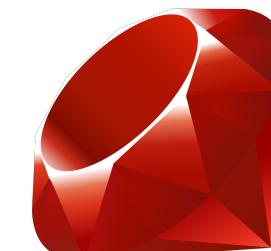
- **Tecnólogo em Segurança da Informação- FATEC-American (2015)**
- **Arquitetura de Software- Faculdade VINCIT**

Experiência

- Desenvolvedor de software desde 2014, com experiência em empresas como Reclame Aqui e Guide Investimentos, utilizando tecnologias como Ruby, Go, Angular e TypeScript.
- Abordagem versátil em diferentes projetos, desde MVPs até otimização de sistemas legados. Domínio de arquiteturas como MVC, microserviços e DDD, além de Kubernetes e CI/CD.
- Acredita no desenvolvimento de software como uma arte e compartilha seu conhecimento com a comunidade.

Certificações

- Arquitetura de Software
- Business Intelligence Pentaho
- DevOps Essentials
- Cert Prep: ITIL® 3 Foundation
- StoryTelling



Evolução da tecnologia no tempo

On-Premise

Cloud

Serverless



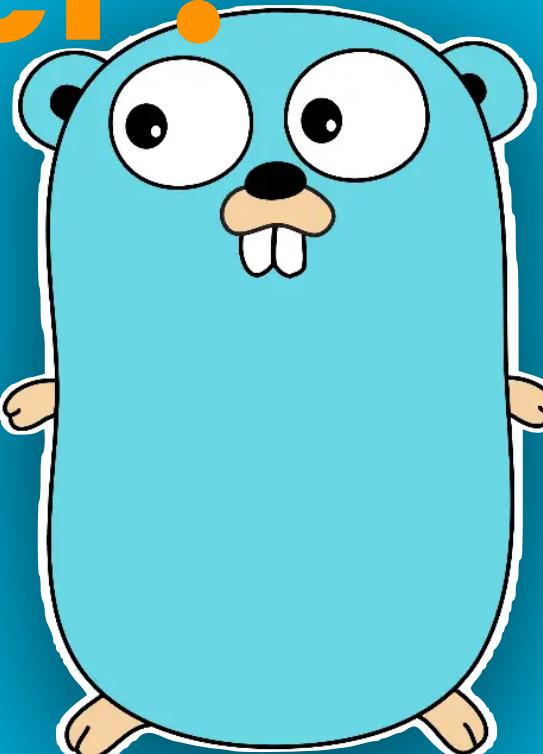
Análise de Dados

Descriptiva: O que aconteceu?

Diagnóstica: Por que isso aconteceu?

Preditiva: Quando isso vai acontecer?

Prescritiva: Como posso fazer isso acontecer?



Evolução da tecnologia no tempo

Simbólica(Regras/Fórmulas)

Aprendizado(Padrões/Decisões)

Conexionista(Padrões)

Generativa(Padrões/Criar)



Sobre o Gemini

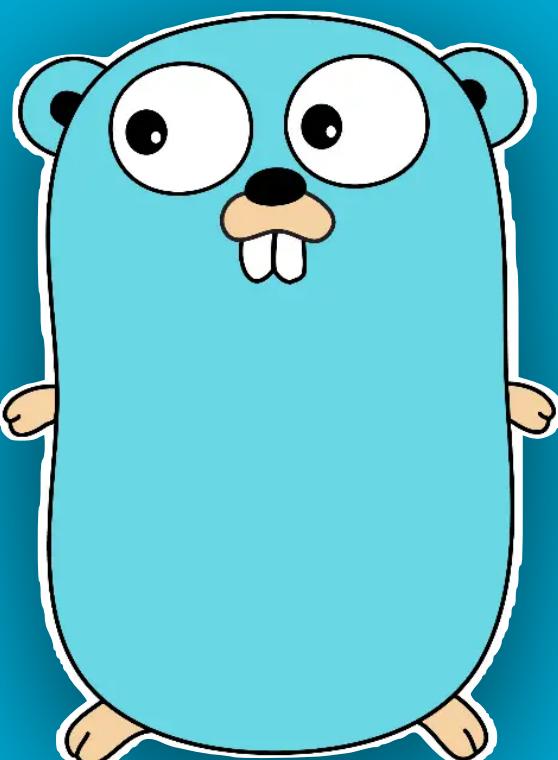
Multimodal

Nano, Pro e Ultra



Sobre o Gemini

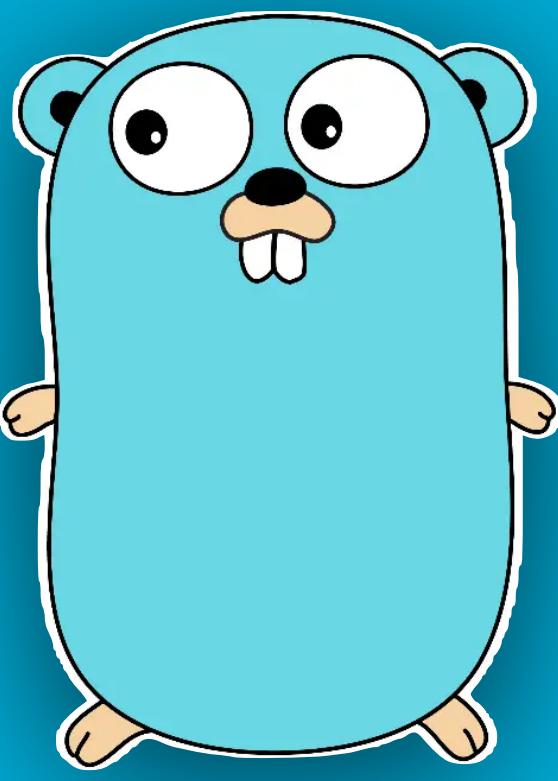
- Busca de informações
- Reconhecimento de objeto
- Compreensão de conteúdo digital
- Geração de conteúdo estruturado
- Criação de legenda/descrição automáticos
- Retornar informações sobre itens em uma imagem
- Entenda telas e interfaces
- Compreender diagramas técnicos
- Fazer uma recomendação com base em várias imagens
- Gerar descrição do vídeo

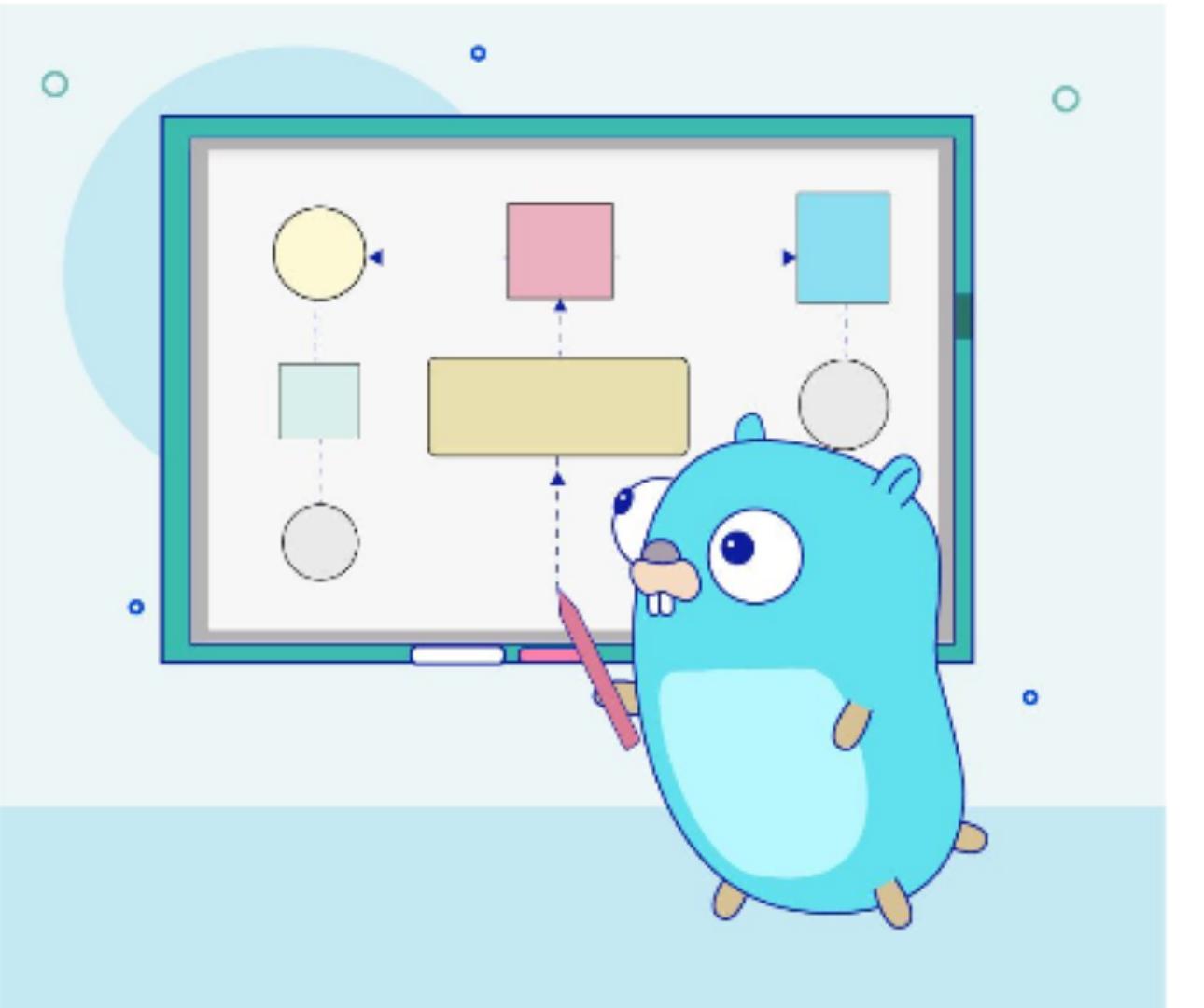


Go
O C++
que Tomou um chá de Camomila



velocidade de desenvolvimento
performance.
segurança





Go is a Platform

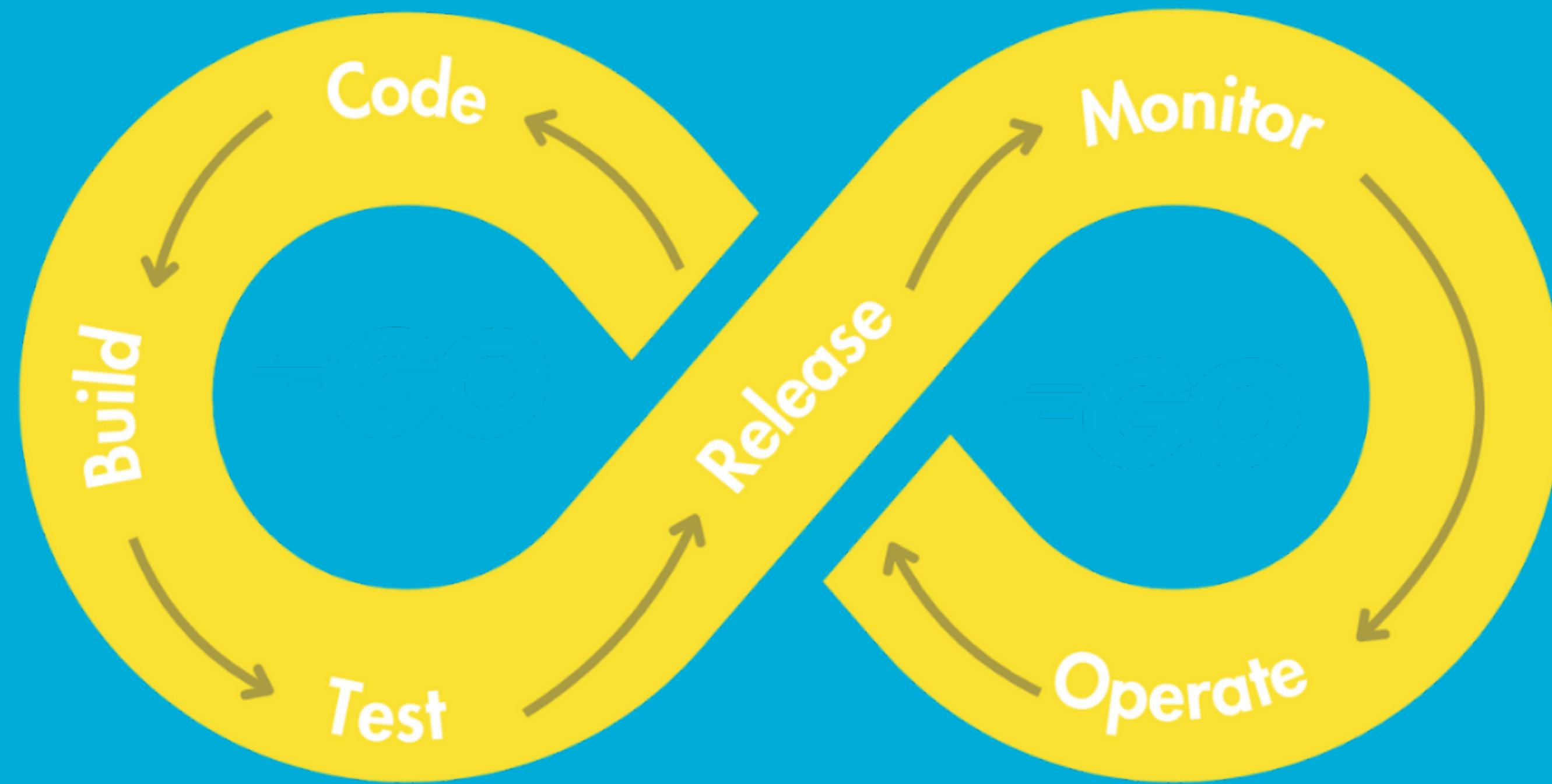
Go is not just a language, it is a
complete developer experience
across the development lifecycle



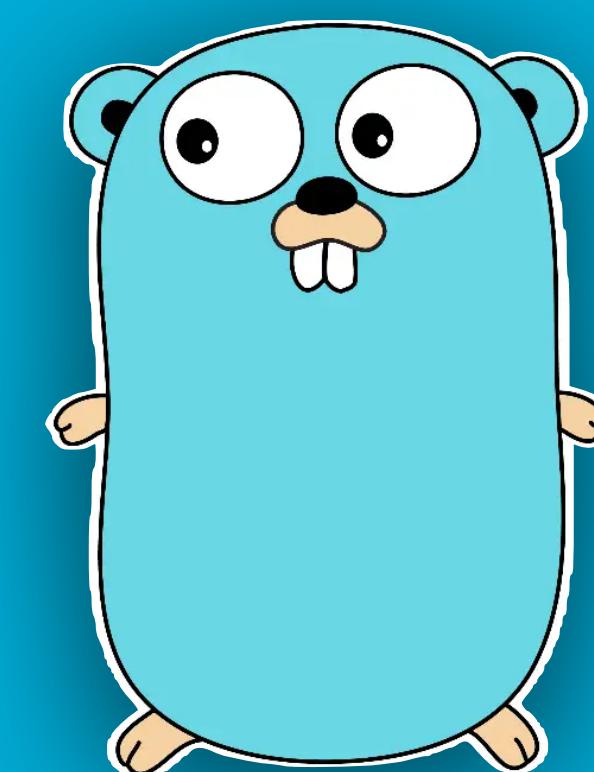
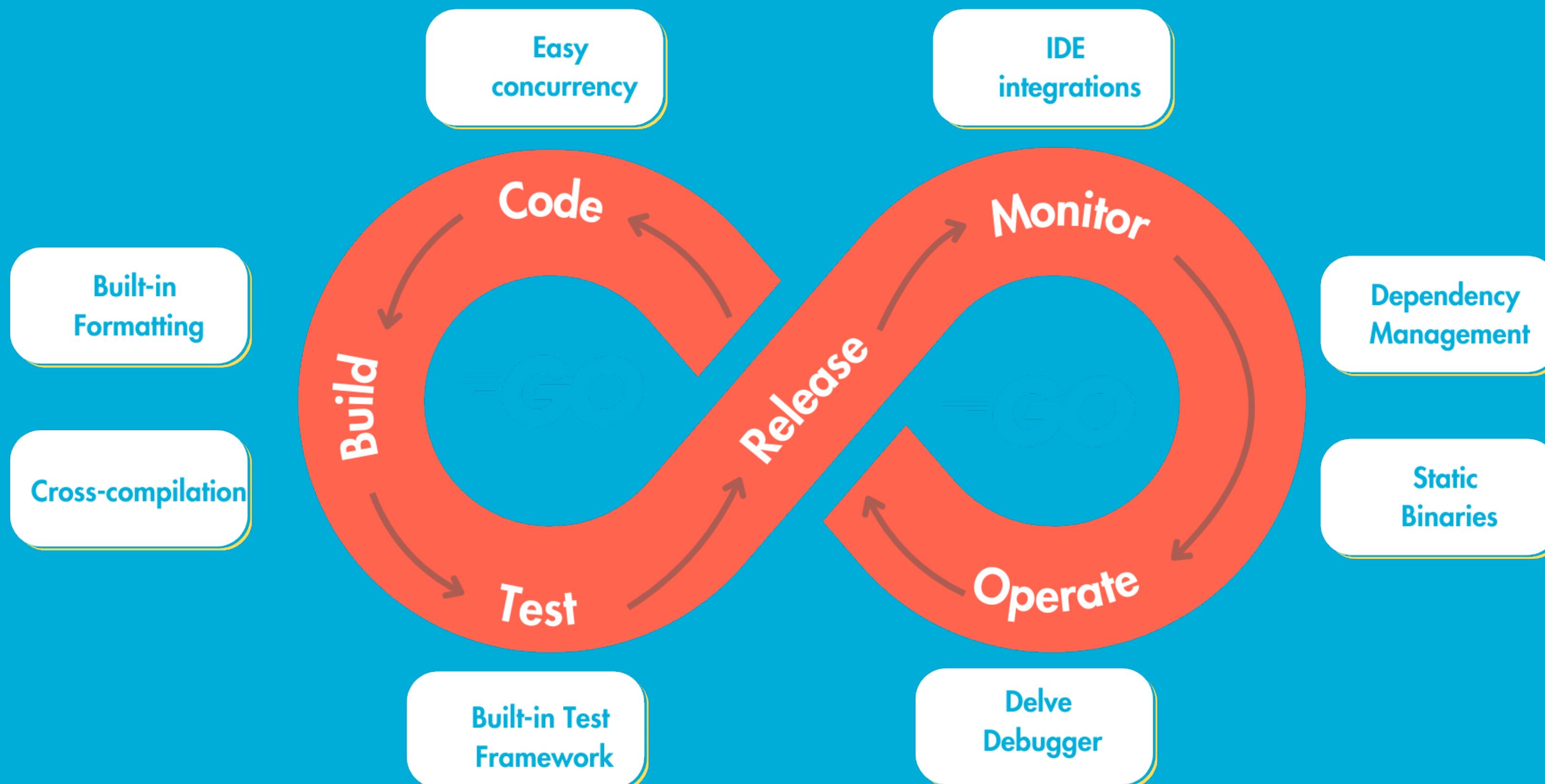
DevOps Cycle



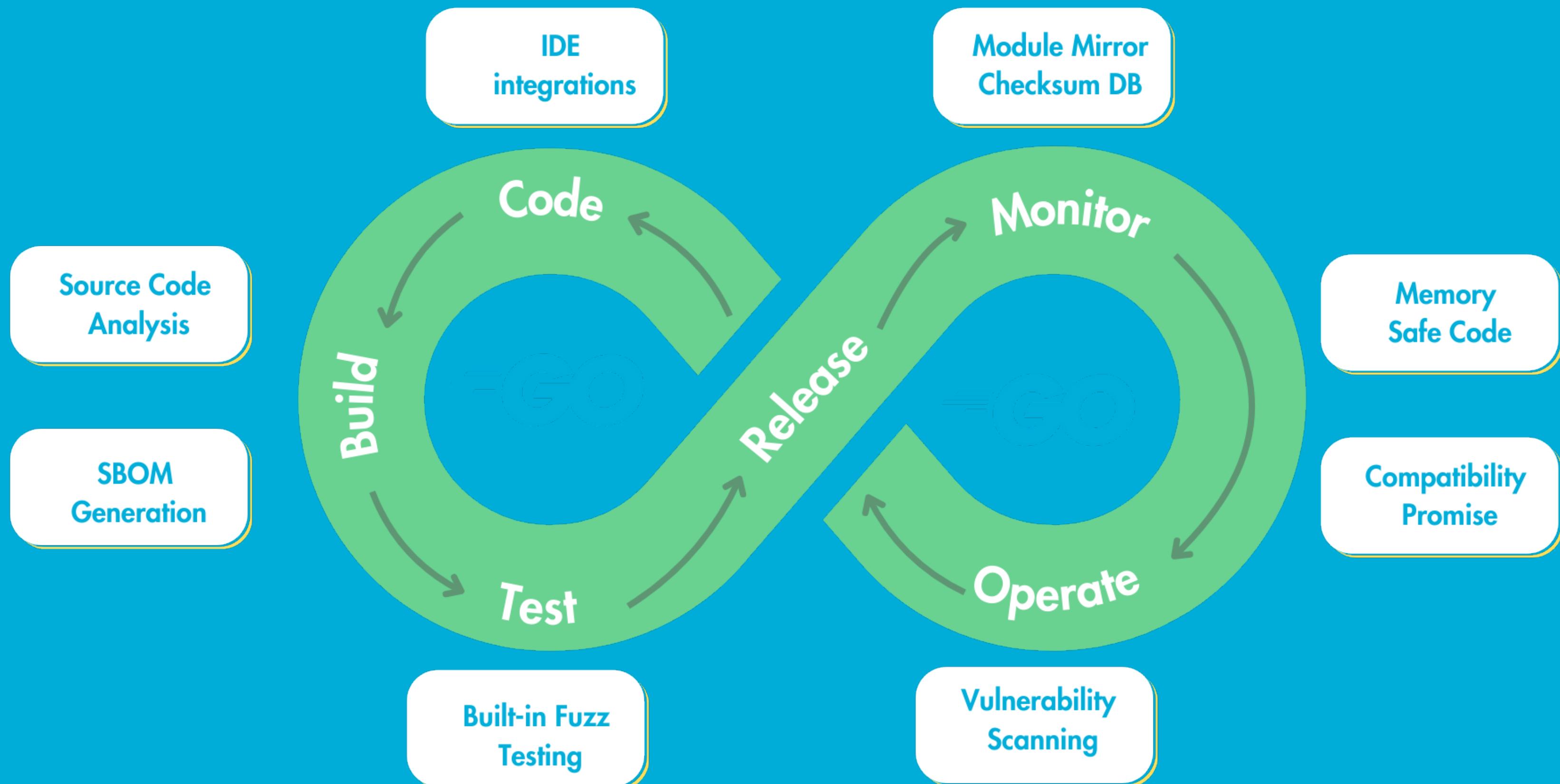
DevOps Cycle



Soluções ponta a ponta para velocidade do desenvolvedor



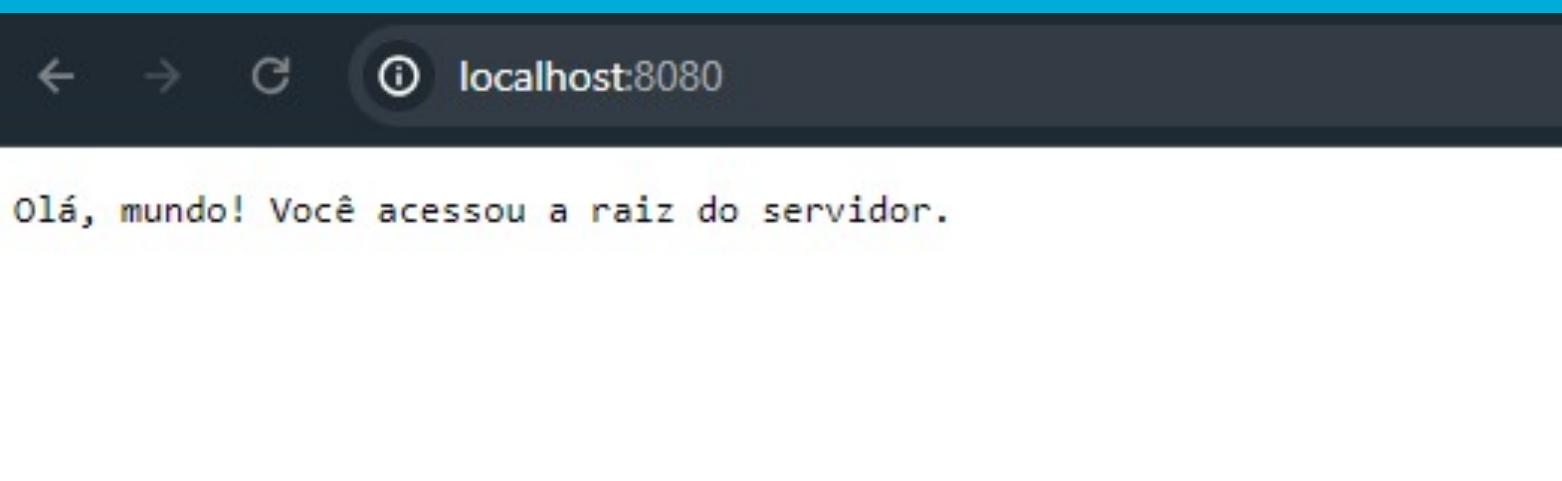
Soluções ponta a ponta para segurança



Simples?

```
● ● ●  
package main  
  
import (  
    "fmt"  
    "net/http"  
)  
  
func main() {  
    // Cria um manipulador para a rota raiz ("/")  
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {  
        fmt.Fprintf(w, "Olá, mundo! Você acessou a raiz do servidor.")  
    })  
  
    // Inicia o servidor na porta 8080  
    fmt.Println("Servidor iniciado na porta 8080...")  
    http.ListenAndServe(":8080", nil)  
}
```

```
● ● ●  
O → go run simplicidade/main.go  
Servidor iniciado na porta 8080...
```



Concorrência?

```
package main

import (
    "fmt"
    "sync"
    "time"
)

func worker(id int, wg *sync.WaitGroup) {
    defer wg.Done() // Sinaliza que a goroutine terminou

    fmt.Printf("Worker %d iniciando\n", id)
    time.Sleep(time.Second) // Simula alguma tarefa demorada
    fmt.Printf("Worker %d finalizando\n", id)
}

func main() {
    var wg sync.WaitGroup // Cria um WaitGroup para sincronizar as goroutines

    for i := 1; i <= 5; i++ {
        wg.Add(1)          // Incrementa o contador do WaitGroup
        go worker(i, &wg) // Inicia uma nova goroutine para executar a função worker
    }

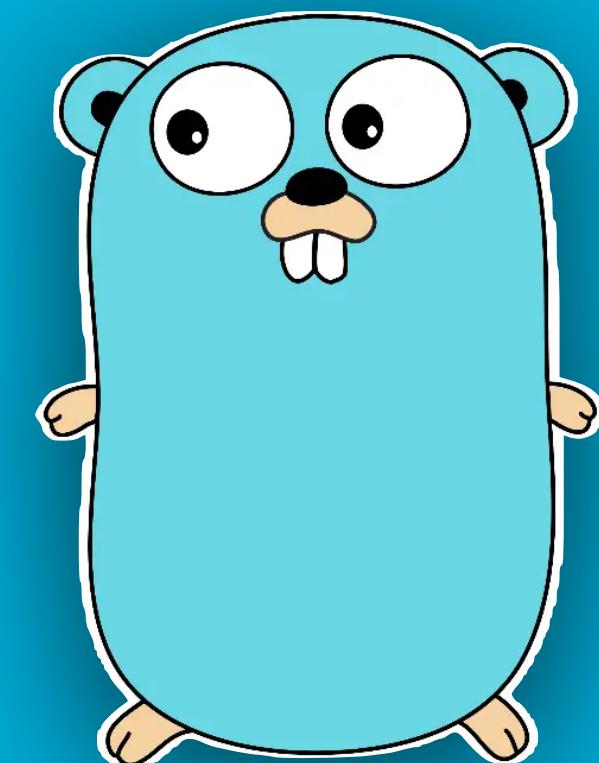
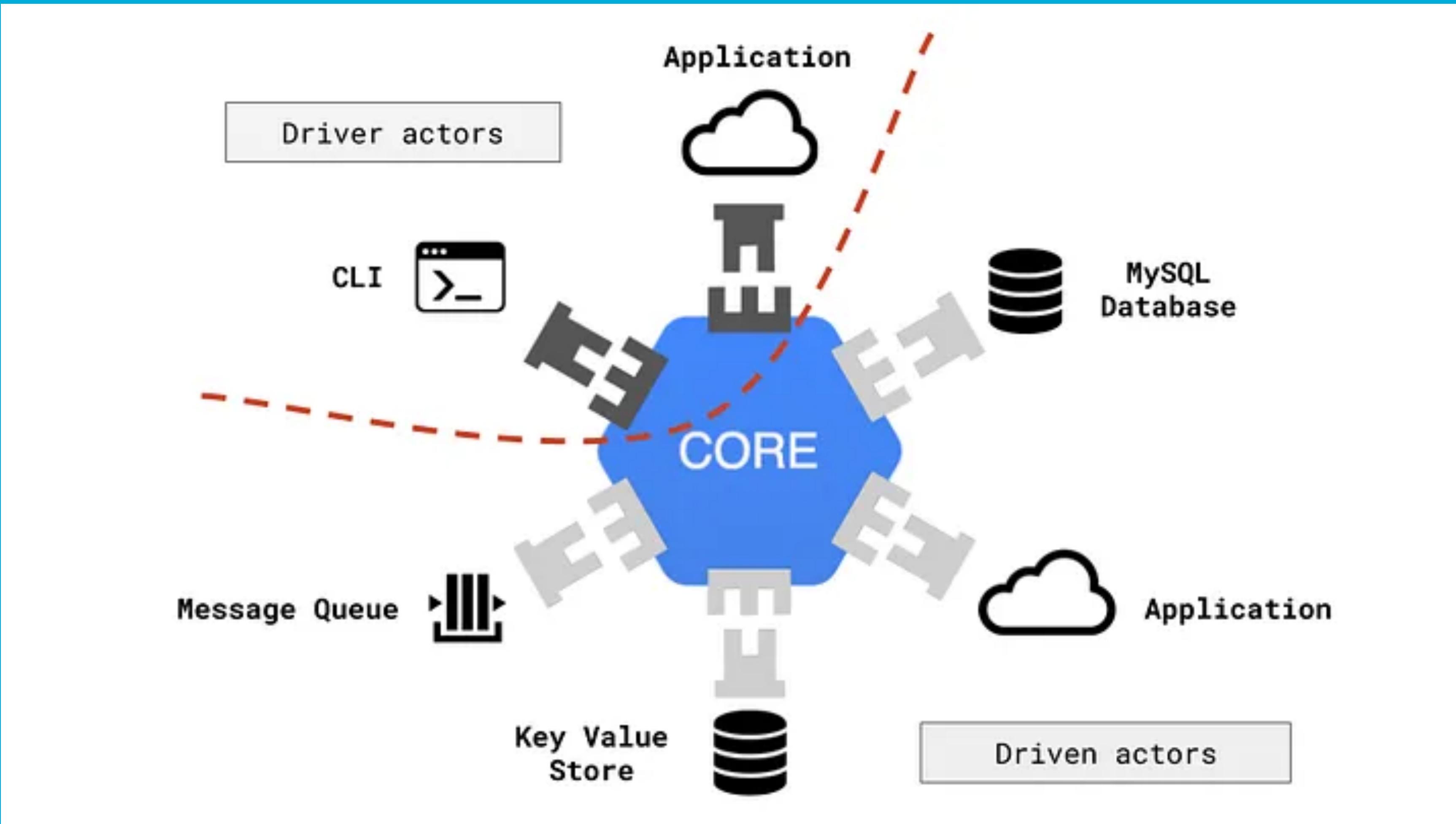
    wg.Wait() // Aguarda todas as goroutines terminarem
    fmt.Println("Todos os workers finalizaram!")
}
```

```
o → go run concorrencia/main.go

Worker 5 iniciando
Worker 1 iniciando
Worker 2 iniciando
Worker 3 iniciando
Worker 4 iniciando
Worker 1 finalizando
Worker 5 finalizando
Worker 4 finalizando
Worker 3 finalizando
Worker 2 finalizando
```

Todos os workers finalizaram!





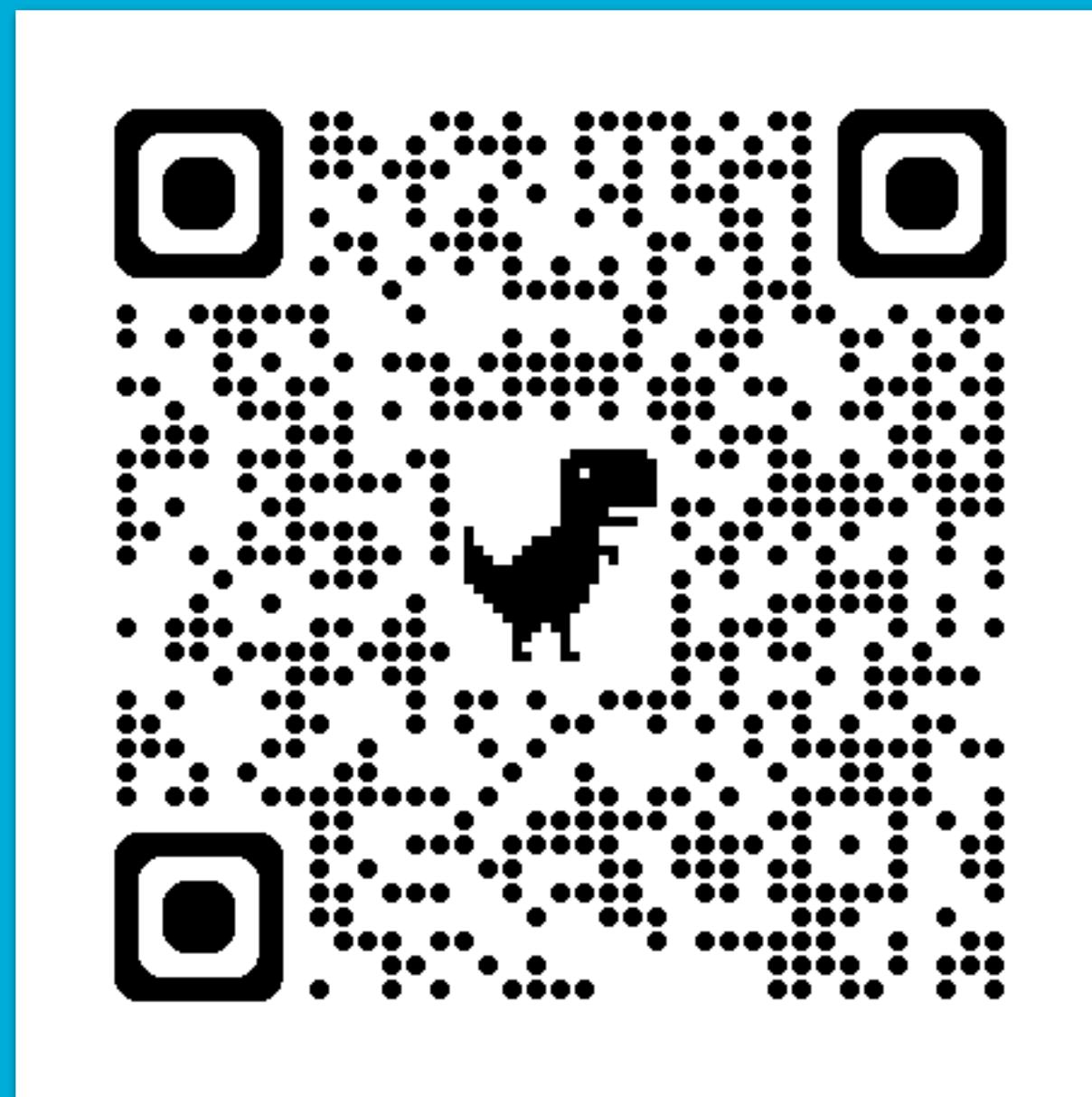
Estrutura que utilizo...

```
cmd
└── migration
    └── migration.go
    config.yaml
    main.go
    wire.go
    wire_gen.go
internal
└── adapter
    └── postgres.go
    core
        ├── domain
        ├── port
        └── usecase
    dto
        └── tenant.dto
    handler
        └── handler.go
    mock
        └── database_mock.go
    model
        └── tenant.go
    repository
        └── tenant_repository.go
pkg
└── validator
server
    ├── echoServer.go
    └── server.go
```

Ponto de Entrada da Aplicação
Migrações do Banco de Dados
Script de Migração (GORM)
Arquivo de Configuração da Aplicação
Ponto de Entrada Principal da Aplicação
Gerador de Injeção de Dependência (Wire)
Código Gerado pelo Wire (Não modificar)
Código Interno da Aplicação (Não acessível externamente)
Adaptadores para Serviços Externos
Adaptador para Banco de Dados PostgreSQL
Lógica de Negócio Principal (Domínio)
Agregados do Domínio (Entidades Relacionadas)
Interfaces de Portas (Abstrações para Adaptadores)
Casos de Uso (Lógica de Aplicação)
Objetos de Transferência de Dados (DTOs)
DTO para Inquilino
Manipuladores de Requisições (REST, etc.)
Manipulador Genérico
Mocks para Testes
Mock de Banco de Dados
Modelos de Dados (Representações)
Modelo de Inquilino
Repositórios de Dados (Implementações)
Repositório de Inquilinos
Pacotes Compartilhados (Reutilizáveis)
Validações de Dados
Servidor Web e Componentes Relacionados
Implementação do Servidor (Echo)
Interface do Servidor



Mãos à Obra



Dúvidas?
Perguntas?



Valeu, galera! 🚀

