Roadmap Complète Go : Du Débutant à l'Expert

🎯 Niveau 1 : Fondamentaux (2-3 semaines)

Installation et Environnement

- Installation de Go (go install, GOPATH, GOROOT)
- Configuration de l'IDE (VS Code + extension Go, GoLand)
- Commandes de base : `go run`, `go build`, `go mod`
- Workspace et modules Go

Syntaxe de Base

- Variables et constantes (`var`, `:=`, `const`)
- Types de données primitifs (int, float, bool, string, byte, rune)
- Opérateurs (arithmétiques, logiques, comparaison)
- Structures de contrôle : `if/else`, `switch`, `for`
- Fonctions : déclaration, paramètres, retours multiples
- Packages et imports
- Visibilité (exported vs unexported)

Types Composés

- Arrays et Slices (différences, manipulation, capacité)
- Maps (création, accès, itération)
- Structs (définition, initialisation, tags)
- Pointeurs (syntaxe, utilisation, différence avec les valeurs)

Gestion des Erreurs

- Pattern `error` natif
- Création d'erreurs personnalisées
- `panic` et `recover`
- Bonnes pratiques de gestion d'erreurs

🚀 Niveau 2 : Intermédiaire (3-4 semaines)

Programmation Orientée Objet en Go

- Méthodes sur types et structs
- Interfaces (définition, implémentation implicite)
- Composition vs héritage
- Interface vide (`interface{}`) et `any`
- Type assertions et type switches
- Embedding de structs

Concurrence (Core de Go)

- Goroutines : création, fonctionnement
- Channels : buffered vs unbuffered
- Select statement
- Patterns de concurrence :
  - Worker pools
  - Pipeline pattern
  - Fan-out/Fan-in
  - Context cancellation
- Mutex et sync primitives (`sync.Mutex`, `sync.RWMutex`, `sync.WaitGroup`)
- Race conditions et `go run -race`

Packages Standard Essentiels

- `fmt` : formatage et I/O
- `strings` et `strconv` : manipulation de chaînes
- `time` : gestion du temps et durées
- `os` et `io` : système de fichiers
- `bufio` : lecture/écriture bufferisée
- `encoding/json` : marshaling/unmarshaling
- `flag` : parsing d'arguments CLI
- `log` : logging simple

Testing

- Tests unitaires avec `testing`
- Table-driven tests
- Benchmarks (`testing.B`)
- Test coverage : `go test -cover`
- Mocking et interfaces
- `testify` (bibliothèque externe)

💼 Niveau 3 : Avancé (4-6 semaines)

Développement Web

- HTTP server natif (`net/http`)
- Routing et middleware
- Frameworks web :
  - **Gin** (performant, populaire)
  - **Echo** (léger, rapide)
  - **Fiber** (inspiré d'Express.js)
  - **Chi** (minimaliste, idiomatique)
- REST API design
- Validation de données
- Authentication/Authorization (JWT)
- WebSockets
- GraphQL avec `gqlgen`

Bases de Données

- `database/sql` (interface standard)
- Drivers : PostgreSQL (pgx), MySQL
- ORM :
  - **GORM** (le plus populaire)
  - **sqlx** (extension de database/sql)
  - **Ent** (Facebook, type-safe)
- Migrations (golang-migrate)
- Connection pooling
- Transactions
- Prepared statements

Context Package

- Propagation de contexte
- Timeouts et deadlines
- Cancellation signals
- Context values (quand et comment)

Gestion Avancée des Erreurs

- `errors.Is` et `errors.As` (Go 1.13+)
- Error wrapping avec `fmt.Errorf("%w")`
- Custom error types
- Bibliothèque `pkg/errors`

Performance et Optimisation

- Profiling CPU : `pprof`
- Memory profiling et heap analysis
- Escape analysis
- Benchmarking avancé
- Optimisation de slices et maps
- String interning
- `sync.Pool` pour réutilisation d'objets

🏗️ Niveau 4 : Architecture et Patterns (3-4 semaines)

Design Patterns en Go

- Creational : Factory, Builder, Singleton
- Structural : Adapter, Decorator, Proxy
- Behavioral : Strategy, Observer, Command
- Functional options pattern
- Repository pattern
- Service layer pattern

Architecture d'Applications

- Clean Architecture / Hexagonal Architecture
- Dependency Injection
- Configuration management (Viper, env)
- Logging structuré (zap, zerolog)
- Graceful shutdown
- Health checks et readiness probes

Microservices

- gRPC et Protocol Buffers
- Message queuing (RabbitMQ, Kafka avec Sarama)
- Service discovery (Consul, etcd)
- Circuit breaker (gobreaker)
- Distributed tracing (OpenTelemetry)
- API Gateway patterns

CLI Development

- `cobra` pour CLI complexes
- `urfave/cli` alternative
- Flags et subcommands
- Configuration files
- Progress bars et UI terminal

🔒 Niveau 5 : Production et DevOps (3-4 semaines)

Sécurité

- Input validation et sanitization
- SQL injection prevention
- XSS et CSRF protection
- Secure headers
- Rate limiting
- Encryption (crypto package)
- TLS/SSL configuration
- Secrets management

Containerisation et Déploiement

- Dockerfile multi-stage pour Go
- Optimisation de la taille d'image
- Docker Compose pour dev
- Kubernetes :
  - Deployments, Services, ConfigMaps
  - Health checks (liveness/readiness)
  - Resource limits
  - Helm charts
- CI/CD avec GitHub Actions, GitLab CI

Observabilité

- Metrics avec Prometheus
- Logging centralisé (ELK, Loki)
- Distributed tracing (Jaeger, Zipkin)
- Alerting (Alertmanager)
- Dashboards (Grafana)

Go Modules Avancé

- Semantic versioning
- Go workspace mode (Go 1.18+)
- Private modules
- Vendor directory
- Module proxy configuration

🧠 Niveau 6 : Expert (Ongoing)

Internals de Go

- Memory model de Go
- Garbage collector (fonctionnement, tuning)
- Scheduler de goroutines (GPM model)
- Stack vs heap allocation
- Assembly Go (comprendre, pas forcément écrire)
- Compiler toolchain

Concurrence Avancée

- Lock-free programming
- Atomic operations (`sync/atomic`)
- Memory ordering
- Patterns avancés (errgroup, semaphore)
- Context propagation complexe

Generics (Go 1.18+)

- Type parameters
- Type constraints
- Generic functions et types
- Standard library generics (maps, slices)
- Quand utiliser/éviter les generics

Reflection et Meta-programming

- `reflect` package
- Type introspection
- Dynamic function calls
- Limitations et performance

Code Generation

- `go generate`
- `stringer`, `mockgen`
- Protobuf et gRPC codegen
- Custom code generators

Compilation et Build

- Build tags et conditional compilation
- Cross-compilation (GOOS, GOARCH)
- CGO et intégration C/C++
- Link flags et optimisations
- Plugin system

Contribution Open Source

- Lire et comprendre le code Go standard
- Contribuer à des projets Go populaires
- Design d'API idiomatiques
- Documentation (godoc, exemples)
- Code review best practices

📚 Ressources Clés

Livres

- **"The Go Programming Language"** (Donovan & Kernighan) - la référence
- **"Concurrency in Go"** (Katherine Cox-Buday)
- **"Learning Go"** (Jon Bodner) - moderne, Go 1.18+
- **"100 Go Mistakes and How to Avoid Them"** (Teiva Harsanyi)

Documentation Officielle

- Tour of Go - interactif
- Effective Go
- Go Blog
- Go by Example

Pratique

- Exercism Go Track
- LeetCode pour algorithmes
- HackerRank Go
- Projets perso : CLI tools, APIs REST, microservices

Communauté

- r/golang
- Gophers Slack
- GopherCon talks (YouTube)
- Go Time podcast

🎯 Projets Pratiques par Niveau

Débutant

- CLI calculator
- Todo list en CLI
- File organizer
- URL shortener (en mémoire)

Intermédiaire

- REST API avec base de données
- Web scraper concurrent
- Chat server avec WebSockets
- Authentication service (JWT)

Avancé

- Distributed cache (type Redis simple)
- API Gateway
- Message broker basique
- Monitoring dashboard

Expert

- Container runtime simplifié
- Database engine basique
- Load balancer avec health checks
- Contribution à un projet Go majeur

⏱️ Estimation Globale

- **Bases solides** : 2-3 mois (pratique quotidienne)
- **Production-ready** : 6-9 mois
- **Expert** : 2+ ans d'expérience continue

Note : Avec votre background en C, Java et Python, vous progresserez plus vite, surtout sur la concurrence (expérience C) et l'architecture (Java).