Avant-propos

Go est un langage qui se veut simple. Cette simplicité est réelle au niveau syntaxique : peu de mots clés, pas d'héritage complexe, pas de généricité débridée jusqu'à récemment. Pourtant, derrière cette façade épurée se cache un runtime complexe qui gère la concurrence, la mémoire et l'ordonnancement avec une efficacité que peu de langages atteignent.

1. Pourquoi cet ouvrage

Mon parcours professionnel m'a amené à travailler sur des systèmes où chaque microseconde compte. J'ai conçu des appliances de load balancing chez HAProxy Technologies, développé un WAF en C et des outils d'agrégation de données en Go chez OZON (dont j'ai été cofondateur et CTO). Ces expériences, complétées par des missions de troubleshooting pour diverses entreprises, m'ont appris à analyser le comportement de code sous contrainte.

Curieux par nature, j'ai été attiré par les défis de sécurité comme ceux de Root-Me, une plateforme francophone reconnue proposant des challenges CTF dans de nombreux domaines. Ces exercices m'ont poussé à comprendre non seulement ce que fait un programme, mais comment il le fait réellement. Cette démarche m'a naturellement conduit à explorer le code assembleur généré par Go.

Amateur de performances, j'ai voulu mesurer l'impact de certaines constructions du langage. Quel est le coût réel d'une fonction anonyme ? Comment les maps affectent-elles le code machine généré ? Ces questions m'ont amené à disséquer le compilateur et le runtime. J'ai rédigé quelques articles de blog pour partager ces découvertes.

L'idée d'un ouvrage complet s'est imposée progressivement. La documentation sur les aspects avancés de Go reste limitée, aussi bien en français qu'en anglais. Les ressources existantes couvrent bien les bases et l'utilisation quotidienne du langage, mais peu explorent les mécanismes internes avec la profondeur nécessaire pour prendre des décisions architecturales éclairées. Quelques projets en ligne abordent ces sujets, mais restent incomplets ou fragmentés. Ce livre vise à combler ce manque.

2. À qui s'adresse ce livre

Ce livre s'adresse aux développeurs qui ont dépassé le stade de l'apprentissage initial, mais aussi aux curieux qui veulent comprendre ce qui se passe sous le capot. Les lecteurs qui découvrent Go peuvent consulter le tutoriel officiel avant d'aborder cet ouvrage.

Le lecteur sait déjà écrire du Go. Il a compris les goroutines et les channels. Il utilise peut-être Go en production. Mais il se pose des questions plus profondes : comment le scheduler choisit-il quelle goroutine exécuter ? Pourquoi certaines variables échappent-elles à la stack ? Comment les maps gèrent-elles réellement les collisions ? Quand faut-il vraiment s'inquiéter des performances ?

L'objectif n'est pas de transformer le lecteur en développeur du compilateur Go, mais de lui donner les connaissances nécessaires pour prendre des décisions éclairées. Comprendre les rouages l'aidera à choisir les meilleures architectures en fonction de ses contraintes : mémoire disponible, exigences de performance, nombre de connexions simultanées, latence acceptable. Comprendre les mécanismes internes change la façon de coder. Le développeur cesse de suivre aveuglément des règles pour comprendre leurs raisons. Il distingue les optimisations qui comptent de celles qui sont inutiles.

3. Structure de l'ouvrage

Le livre explore progressivement les mécanismes internes de Go, du runtime aux types et structures de données, puis aux mécanismes avancés du langage et à la concurrence. La progression est pensée pour construire une compréhension cohérente.

Les deux premières parties (Fondements du runtime et Types et structures de données) sont largement indépendantes et peuvent être lues séparément. Elles apportent chacune une valeur immédiate sans nécessiter la lecture complète du livre. En revanche, les parties sur les mécanismes avancés du langage et la concurrence s'appuient sur les fondations posées précédemment. Comprendre le scheduler avant d'aborder la concurrence, ou connaître la représentation des types avant d'étudier les optimisations, permet de saisir les implications de ses choix de conception.

L'accent est mis sur la théorie et la compréhension des mécanismes plutôt que sur les recettes pratiques. Les exemples de code servent à illustrer les concepts, pas à fournir des solutions clés en main.

Les annexes contiennent du code de démonstration et des benchmarks qui mettent en évidence les éléments expliqués dans les chapitres. Ces démonstrations permettent de valider et de mesurer concrètement les comportements décrits. D'autres annexes détaillent certains algorithmes rencontrés à l'intérieur du runtime, comme les structures de tas binaires ou le sondage quadratique dans les tables de hachage.

La lecture des annexes n'est pas requise pour profiter de l'ouvrage. Elles servent d'approfondissement pour les lecteurs qui souhaitent aller plus loin sur certains points spécifiques.

4. Note sur la terminologie

Ce livre utilise de nombreux termes anglais sans les traduire. Ce choix n'est pas un rejet de la langue française, mais une nécessité pratique. La documentation officielle de Go, le code source du runtime et du compilateur, ainsi que la quasi-totalité des ressources techniques sont en anglais. Traduire *scheduler* par "ordonnanceur" ou *runtime* par "environnement d'exécution" créerait une barrière supplémentaire entre le lecteur et les sources primaires. Pour ne pas nuire à la compréhension, chaque terme anglais employé est défini dans le glossaire en fin d'ouvrage, où son équivalent français est indiqué lorsqu'il en existe un. À l'inverse, certains termes techniques passés dans l'usage courant en français sont conservés dans notre langue, comme le compilateur ou le pointeur plutôt que *compiler* ou *pointer*.

5. Approche méthodologique

Ce livre privilégie les sources officielles. Les affirmations sur le comportement du runtime ou du compilateur s'appuient sur la documentation de `go.dev` ou directement sur le code source du dépôt `github.com/golang/go`. Quand c'est pertinent, des démonstrations techniques et des benchmarks viennent valider les explications.

Le ton adopté est délibérément factuel et technique. Pas d'enthousiasme artificiel, pas de promesses marketing. Go a des forces et des limitations. Les deux méritent d'être comprises. Certains choix de conception sont discutables, d'autres sont des compromis assumés. L'objectif est de donner au lecteur les éléments pour forger sa propre opinion.

6. Prérequis

Le lecteur doit avoir une pratique régulière de Go. Il connaît la syntaxe, les types de base, les structures de contrôle. Il a déjà utilisé les goroutines et les channels dans des programmes réels. Une familiarité avec les concepts systèmes de base (processus, threads, mémoire virtuelle) est utile mais pas indispensable.

Avoir quelques notions d'assembleur peut faciliter la lecture de certains passages, notamment lors de l'examen des optimisations du compilateur ou de certaines implémentations bas niveau. Cependant, ce n'est pas bloquant pour comprendre les concepts. Les explications restent accessibles sans connaissance approfondie de l'assembleur.

Ce livre n'est pas :

- un tutoriel d'introduction à Go ;
- une référence exhaustive du langage ;

- un catalogue de solutions clés en main ;
- un guide des bonnes pratiques générales.

C'est une exploration technique des mécanismes qui font fonctionner Go. Une fois ces mécanismes compris, les bonnes pratiques découlent naturellement de la compréhension plutôt que de l'application mécanique de règles.

7. Versions et compatibilité

Les explications se basent principalement sur Go 1.26 et versions ultérieures. Le runtime Go évolue, mais ses principes fondamentaux restent stables. Les différences significatives entre versions sont signalées quand elles sont pertinentes.

Les benchmarks ont été réalisés sur deux configurations. Les premiers travaux utilisaient un Intel Core i7 Quad-Core à 2,3 GHz sous macOS. Les tests ultérieurs ont été effectués sur un Apple M4 Pro sous macOS ARM.

Certains comportements sont spécifiques à certaines architectures. Pour faciliter l'interprétation, l'architecture sera précisée quand nécessaire. En l'absence d'indication, les tests ont été effectués sur x86. Les tests sont très souvent des comparaisons : ce qui compte n'est pas le résultat absolu mais la différence entre les deux mesures, ce qui rend les conclusions transposables d'une architecture à l'autre.

Bonne lecture. J'espère que ce livre vous aidera à mieux comprendre Go et à prendre des décisions éclairées dans vos projets.