Go

Learning

- https://go.dev/tour/welcome/1
  - https://tour.ardanlabs.com/tour/eng/list
  - https://x.com/func25/status/1959880006039146785
  - https://go.dev/doc/
  - https://go.dev/doc/effective_go
  - https://go.dev/doc/articles/wiki/
  - https://go.dev/doc/modules/layout
  - https://gobyexample.com/
  - https://stream-wiki.notion.site/Stream-Go-10-Week-Backend-Eng-Onboarding-625363c8c3684753b7f2b7d829bcd67a
  - https://go.dev/blog/testing-time
  - https://hackmysql.com/design/thinking-in-go/
  - https://hackmysql.com/golang/idiomatic-go/
  - https://hackmysql.com/golang/go-antipatterns/
  - https://hackmysql.com/golang/go-channel-red-flags/
  - https://hackmysql.com/golang/go-error-types/
  - https://hackmysql.com/golang/go-single-character-names/
  - https://www.bytesizego.com/books/anatomy-of-go
  - https://www.bytesizego.com/books/foundations-of-debugging
  - https://bitfieldconsulting.com/books/go-library
  - https://bitfieldconsulting.com/books/deeper
  - https://github.com/uber-go/guide/blob/master/style.md
  - https://pkg.go.dev/runtime/pprof
  - https://go.dev/doc/pgo
  - https://www.reddit.com/r/elixir/comments/13ub46o/comment/jm2zpez/
  - https://quii.gitbook.io/learn-go-with-tests/
  - https://bitfieldconsulting.com/books/go-library
  - https://go.dev/wiki/TestComments#assert-libraries
  - https://grafana.com/blog/2024/02/09/how-i-write-http-services-in-go-after-13-years/
  - https://www.mailgun.com/blog/it-and-engineering/golangs-superior-cache-solution-memcached-redis/
  - https://www.reddit.com/r/golang/comments/18ujt6g/new_at_go_start_here/

    - https://docs.google.com/document/d/1Zb9GCWPKeEJ4Dyn2TkT-O3wJ8AFc-IMxZzTugNCjr-8/edit
- https://rakyll.org/instruments/
- https://chris124567.github.io/2021-06-21-go-performance/
- https://kevin.burke.dev/kevin/fast-parallel-database-tests/
- https://www.crunchydata.com/blog/dont-mock-the-database-data-fixtures-are-parallel-safe-and-plenty-fast
- https://michael.stapelberg.ch/posts/2024-11-19-testing-with-go-and-postgresql-ephemeral-dbs/
    - https://github.com/peterldowns/pgtestdb
- https://michael.stapelberg.ch/posts/2024-10-22-debug-go-core-dumps-delve-export-bytes/
- https://earthly.dev/blog/property-based-testing/
- https://betterstack.com/community/guides/logging/logging-in-go/#using-third-party-logging-backends-with-slog
    - Use **Info("msg")**, **Error("msg")** for, non critical code paths.
    - Use **LogAttrs(ctx, slog.LevelDebug, "msg")** for speed and zero allocation logging!
- https://github.com/eranyanay/1m-go-websockets
    - https://speakerdeck.com/eranyanay/going-infinite-handling-1m-websockets-connections-in-go
- A pragmatic guide to Go module updates
- Honeycomb's OpenTelemetry Distribution for Go can create a link to a trace visualization in the Honeycomb UI for local traces
- https://opentelemetry.io/blog/2026/go-compile-time-instrumentation-v1/
- Dynamic crimes in golang
- https://brandur.org/fragments/go-no-common-nouns
- https://anto.pt/articles/go-http-responsewriter
- **$(go env GOPATH)/bin/staticcheck**
- https://skoredin.pro/blog/golang/cpu-cache-friendly-go
- https://benhoyt.com/writings/go-1brc/

Tricks / Best practices

Flags first config

config.go:

```
package config

import (
   "flag"
   "log"
   "os"

   "github.com/peterbourgon/ff/v3"
)

type Config struct {
   Port   string
   DbHost string
   DbUser string
   DbPass string
}

func LoadConfig() *Config {
   // Define the flags
   fs := flag.NewFlagSet("mysvc", flag.ExitOnError)
   var (
      port   = fs.String("port", "8080", "Server
port (ENV: PORT)")
      dbHost = fs.String("db_host", "localhost",
"Database host (ENV: DB_HOST)")
      dbUser = fs.String("db_user", "", "Database
user (ENV: DB_USER)")
      dbPass = fs.String("db_pass", "", "Database
password (ENV: DB_PASS)")
   )

   // Parse flags with support for environment
variables
   ff.Parse(fs, os.Args[1:],
ff.WithEnvVarPrefix(""))

   // Return the populated configuration
   return &Config{
      Port:   *port,
      DbHost: *dbHost,
```

```
        DbUser: *dbUser,
        DbPass: *dbPass,
    }
}
```

main.go:

```
package main

import (
    "log"
    "mysvc/config"
)

func main() {
    // Load configuration
    cfg := config.LoadConfig()

    // Output configuration for demonstration
(replace this with your app logic)
    log.Printf("Server starting on port %s",
cfg.Port)
    log.Printf("Database host: %s", cfg.DbHost)
    log.Printf("Database user: %s", cfg.DbUser)
}
```

Memory/GC optimizations

- https://go.dev/doc/gc-guide
  - https://tip.golang.org/doc/gc-guide#GOGC
  - https://tip.golang.org/doc/gc-guide#Memory_limit
  - https://www.perplexity.ai/search/gogc-100-vs-gogc-10000-nez51MAOTtmxwgthFgI0fg

```
GOGC=100 # default, consider increasing to 1000,
10000, or setting it to off
GOMEMLIMIT=2750MiB
```

Consider https://blog.twitch.tv/en/2019/04/10/go-memory-ballast-how-i-learnt-to-stop-worrying-and-love-the-heap/ as well.

Debugging GC

```
GODEBUG=gctrace=1
```

Formatting / Linting

- **go fmt**
  - https://github.com/segmentio/golines
  - https://pkg.go.dev/golang.org/x/tools/cmd/goimports
  - https://github.com/mvdan/gofumpt
  - depguard: Go linter that checks if package imports are in a list of acceptable packages
  - https://revive.run/
  - wrapcheck: check that errors from external packages are wrapped
  - Debugging Go compiler performance in a large codebase / Keeping build times in check
    - actiongraph: Go build actiongraph debug visualiser
    - https://ui.perfetto.dev/#!/viewer
    - depguard: Go linter that checks if package imports are in a list of acceptable packages

Code layout

https://gist.github.com/gmcabrita/7f2d91545855571ecedb48aa1423a9f7

Auto reload app

Can use https://github.com/watchexec/watchexec or https://github.com/cosmtrek/air instead.

```
while true
do
  go install ./cmd/mysvc && mysvc &
  fswatch -1 .
  pkill mysvc && wait
done
```

sqlc

- https://play.sqlc.dev/
  - https://brandur.org/sqlc
  - https://brandur.org/fragments/sqlc-2024

    - https://docs.sqlc.dev/en/stable/howto/named_parameters.html#nullable-parameters
    - https://brandur.org/two-phase-render
    - https://brandur.org/fragments/go-data-fixtures
    - https://brandur.org/fragments/pgx-v5-sqlc-upgrade
    - Formatting: https://www.reddit.com/r/golang/s/wm4DmKXxUJ
        - https://www.sqlfluff.com/
        - https://github.com/sql-formatter-org/sql-formatter
        - https://github.com/darold/pgFormatter
        - deno formatter
    - https://github.com/aitva/sqlc-exp/tree/ebb074bc0538c668d9243befa9c99a9fa4c7a88a/bulk
        - https://docs.sqlc.dev/en/stable/howto/insert.html

    ```
    INSERT INTO table (col_a, col_b) VALUES
    (unnest(@a_array::text[]),
    unnest(@b_array::text[]));
    ```

- https://github.com/riverqueue/river/blob/896fd0b93ba4c334e3d7e1d35d1b03688e5d3830/internal/dbsqlc/sqlc.yaml
    - https://github.com/riverqueue/river/blob/896fd0b93ba4c334e3d7e1d35d1b03688e5d3830/Makefile#L6-L9
    - https://codeberg.org/pfad.fr/fahrtkosten/src/branch/main/internal/migrations
    - https://codeberg.org/pfad.fr/fahrtkosten/src/branch/main/internal/traveller/traveller.sql
    - https://codeberg.org/pfad.fr/fahrtkosten/src/branch/main/sqlc.yaml
    - https://github.com/amacneil/dbmate
    - https://github.com/pressly/goose

Defer and errdefer

Defer: From Basic To Traps

From: https://news.ycombinator.com/item?id=28095488

```
// must use named returns so you can observe the
returned error
func yourfunc() (thing thingtype, err error) {
  thing := constructIt()
```

```
  defer func(){
    err = errors.Join(err, f.Close())
  }()

  // use thing to do a few things that could error
  err = thing.setup()
  if err != nil {
    return nil, err
  }
  other, err := thing.other()
  if err != nil {
    return // naked returns are fine too
  }
  // etc

  return
}
```

See also:

- Examples in Sourcegraph's codebase
  - lainio/err2: Automatic and modern error handling package for Go

Interface guard

https://github.com/uber-go/guide/blob/master/style.md#verify-interface-compliance

```
type Handler struct {
   // ...
}

var _ http.Handler = (*Handler)(nil)
```

Errors as exported constants

Export errors as contant structs to avoid Hyrum's law.

https://pkg.go.dev/net/http#pkg-constants

```
type MyError struct {
```

```
    ErrorString string
}

var (
    ErrNotSupported = &ProtocolError{"feature not
supported"}
    ErrUnexpectedEndState =
&ProtocolError{"unexpected end state"}
)
```

Testing request cancellation

https://brandur.org/fragments/testing-request-cancellation

Password [re]hashing

- https://brandur.org/fragments/password-hashing#rehashing
    - https://brandur.org/fragments/password-hash-nesting

Capture goroutine stack dump (dumps all goroutines)

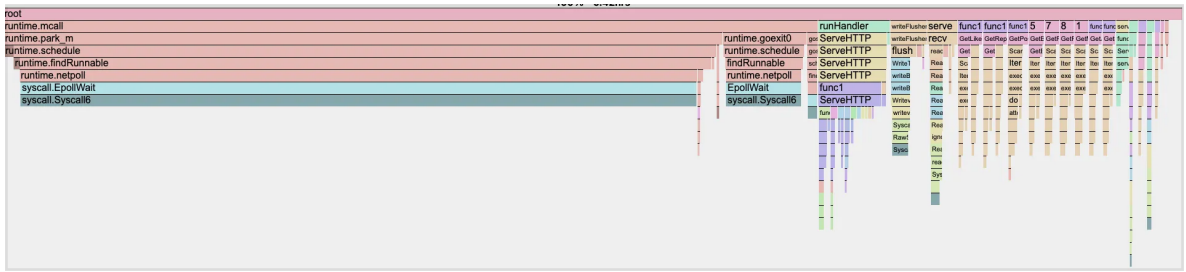
```
func captureGoroutineDump() string {
    buf := make([]byte, 1 << 20) // 1 MiB
    for {
        n := runtime.Stack(buf, true)
        if n < len(buf) {
            return string(buf[:n])
        }

        buf = make([]byte, 2 * len(buf))
    }
}
```

Caveats

Netpoll socket buffering limits (Before <1.25)

Unfortunately, Netpoll only buffers at most 128 sockets in a single EPoll call, meaning we're stuck making several EPoll calls in order to fetch all the sockets becoming available. In a CPU profile, this manifests as nearly 65% of our CPU time being spent

on syscall.EpollWait in runtime.netpoll!



To resolve this issue, the solution is quite apparent: we need to run a larger number of Go runtimes per host and reduce their individual network I/O workloads to something the Go runtime can manage.

Thankfully, in our case, this was as easy as spinning up 8 application containers per host on different ports (skipping Docker NAT) and pointing our Typescript Frontend API at the additional addresses to route its requests.

After implementing this change, we saw a 4x increase in performance.

From a previous maximum throughput of ~1.3M Scylla queries per second across 3 containers on 3 hosts, we see a new maximum of ~2.8M Scylla queries per second (for now) across 24 containers on 3 hosts.

From a previous maximum throughput of ~90K requests served per second to the AppView Frontend, we saw a jump to ~185k requests served per second.

Our p50 and p99 latencies dropped by more than 50% during load tests, and at the same time the CPU utilization on each Dataplane host saw a reduction from 80% across all cores to under 40% across all cores.

This find finally allows us to get the full performance out of our baremetal hardware and will let us scale out much more effectively in the future.

This was (allegedly) fixed on Go 1.25 https://github.com/golang/go/issues/65064

State machines

- Use your database to power state machines

Packages

- bradenaw/backpressure: Prioritized semaphore/rate-limiter/adaptive throttle for load management
  - Compression
    - https://github.com/andybalholm/brotli
    - https://github.com/klauspost/compress
  - https://github.com/Southclaws/fault
  - https://github.com/goforj/godump
  - https://github.com/vimeo/go-retry
    - See also: https://commaok.xyz/post/simple-backoff/
  - https://github.com/enrichman/httpgrace
  - https://github.com/oklog/run
  - sqlite
    - https://github.com/cvilsmeier/go-sqlite-bench
    - https://github.com/zombiezen/go-sqlite (not database/sql compatible)
      - https://modernc.org/sqlite (database/sql compatible)
  - tef/rwmap
    - sync.Map uses a read-only copy and a locked dirty map to implement a concurrent map type in Go
    - RWMap uses a RWMutex instead, but also uses a dirty map to batch up changes
  - tef/sink: A lock-free map for Go 1.24
  - https://pkg.go.dev/golang.org/x/sync/singleflight
    - https://github.com/tarndt/shardedsingleflight
    - https://github.com/go-chi/stampede
  - https://connectrpc.com/docs/go
    - https://github.com/sudorandom/protoc-gen-connect-openapi
    - https://buf.build/studio
  - rill: Go toolkit for clean, composable, channel-based concurrency
  - conc: Better structured concurrency for go
  - betteralign: Make your Go programs use less memory

(maybe)
- durabletask-go: a lightweight, embeddable engine for writing durable, fault-tolerant business logic (orchestrations) as ordinary code
- go-workflows: Embedded durable workflows for Golang similar to DTFx/Cadence/Temporal
- resonate: a dead simple programming model for the cloud
- hatchet: A distributed, fault-tolerant task queue
- river: Fast and reliable background jobs in Go
  - https://riverqueue.com/
  - https://github.com/gmcabrita/goproxy-riverqueue
  - https://github.com/almottier/rivertui
  - https://github.com/riverqueue/riverui
- Utils:
  - https://github.com/riverqueue/river/blob/master/rivershared/util/sliceutil/slice_util.go
  - https://github.com/riverqueue/river/blob/master/rivershared/util/maputil/map_util.go
  - https://github.com/riverqueue/river/blob/master/rivershared/util/ptrutil/ptr_util.go
  - https://github.com/riverqueue/river/blob/master/rivershared/util/randutil/rand_util.go
  - https://github.com/riverqueue/river/blob/master/rivershared/util/serviceutil/service_util.go
  - https://github.com/riverqueue/river/blob/master/rivershared/util/slogutil/slog_util.go
  - https://github.com/riverqueue/river/blob/master/rivershared/util/timeutil/time_util.go
  - https://github.com/riverqueue/river/blob/master/rivershared/util/valutil/val_util.go
- quamina: A fast pattern-matching library in Go
- uber-go/automaxprocs: Automatically set GOMAXPROCS to match Linux container CPU quota
- KimMachineGun/automemlimit: Automatically set GOMEMLIMIT to match Linux cgroups(7) memory limit
- rs/zerolog: Zero Allocation JSON Logger
- bitfield/script: Making it easy to write shell-like scripts in Go
  - https://bitfieldconsulting.com/posts/scripting
- VictoriaMetrics/fastcache: fast thread-safe inmemory cache for big number of entries in Go

- mailgun/groupcache: Clone of golang/groupcache with TTL and Item Removal support
  - https://www.mailgun.com/blog/it-and-engineering/golangs-superior-cache-solution-memcached-redis/
- rogpeppe/gohack: Make temporary edits to your Go module dependencies
- gnet: high-performance, lightweight, non-blocking, event-driven networking framework
- cloudflare/tableflip: Graceful process restarts in Go
- https://github.com/go-chi/chi/
  - https://go-chi.io/
  - https://github.com/go-chi/stampede
- rest: Web services with OpenAPI and JSON Schema done quick in Go
- oapi-codegen: Generate Go client and server boilerplate from OpenAPI 3 specifications
- tsnet: Package tsnet provides Tailscale as a library
- https://github.com/a-h/templ
  - https://templ.guide/
  - https://templ.guide/server-side-rendering/htmx/
- datastar: A real-time hypermedia framework
  - https://data-star.dev/guide/getting_started
  - hot reload setup
  - https://github.com/CoreyCole/datastarui
- https://github.com/go-playground/validator
- https://github.com/go-playground/mold
- https://github.com/gorilla/sessions
- https://github.com/jub0bs/cors
- https://github.com/ardanlabs/conf
- https://github.com/kelseyhightower/envconfig
- https://github.com/apple/pkl-go
- https://github.com/cue-lang/cue
  - https://cuelang.org/
- uber-go/goleak: Goroutine leak detector
- https://github.com/failsafe-go/failsafe-go
  - https://failsafe-go.dev/
- https://github.com/samber/lo
- https://github.com/samber/mo
- https://github.com/samber/do
- go-delve/delve: Delve is a debugger for the Go programming

language
    - aarzilli/gdlv: GUI frontend for Delve
- https://github.com/crawshaw/jsonfile
- https://github.com/rsc/rf
- https://github.com/rsc/omap
- https://github.com/rsc/top
- https://github.com/rsc/ordered
- https://github.com/google/go-cmp/
- https://github.com/uptrace/bun
- https://github.com/go-jet/jet
- https://github.com/Masterminds/squirrel
- https://github.com/jackc/pgx
- https://github.com/sqlc-dev/sqlc
- https://github.com/amacneil/dbmate
- https://github.com/pressly/goose
- gocrest: Hamcrest-like matchers for Go
- https://github.com/stretchr/testify
    - https://github.com/stretchr/testify/assert
- negrel/assert: Zero cost debug assertions for Go
- https://code.pfad.fr/check/
    - https://log.pfad.fr/2024/reinventing-the-assertion-wheel/
- https://github.com/matryer/moq
- https://github.com/peterldowns/pgtestdb
- https://github.com/testcontainers/testcontainers-go
    - https://golang.testcontainers.org/quickstart/
    - https://golang.testcontainers.org/features/garbage_collector/
    - https://chatgpt.com/share/59b193ca-7860-4237-a902-9de95b1fb8d0
        - "how to use testcontainers with multiple test suites requiring a postgres database running, can/should they use the same container? i am using go."
        - "how could we improve upon this setup? for example, instead of having a single container we could have multiple, for example one for each cpu core in my machine. would that be a good idea?"
- https://github.com/DATA-DOG/go-sqlmock
- https://github.com/dominikh/gotraceui
    - https://gotraceui.dev/
- https://github.com/cloudwego/netpoll

- https://github.com/lesismal/go-websocket-benchmark
  - https://github.com/antlabs/greatws
  - https://github.com/lesismal/nbio
  - https://github.com/gobwas/ws
  - https://github.com/gorilla/websocket
  - https://pkg.go.dev/golang.org/x/net/websocket
- https://pkg.go.dev/golang.org/x/oauth2
- https://pkg.go.dev/github.com/golang-jwt/jwt/v5
- https://pkg.go.dev/gopkg.in/yaml.v3
- https://github.com/benbjohnson/hashfs
- https://github.com/segmentio/golines
- https://pkg.go.dev/golang.org/x/tools/cmd/goimports
- https://github.com/mvdan/gofumpt
- https://github.com/gkampitakis/go-snaps
- https://github.com/guregu/dynamo
- https://github.com/uber-go/zap
- https://github.com/frankban/quicktest
- https://github.com/go-quicktest/qt
- rapid: a modern Go property-based testing library
- https://charm.sh/libs/
  - https://github.com/charmbracelet/bubbletea
  - https://github.com/charmbracelet/huh
  - https://github.com/charmbracelet/lipgloss
- https://github.com/mitchellh/mapstructure
- https://github.com/open-telemetry/opentelemetry-go
- https://github.com/prometheus/client_golang
- https://github.com/anishathalye/porcupine
- https://github.com/tulir/whatsmeow
- https://github.com/mdp/qrterminal
- goja: ECMAScript/JavaScript engine in pure Go
- go-size-analyzer: Analyze the size of Go binaries
- ccgo: translate cc AST to Go source code
- batchedrand: Efficient batched random shuffle implementation (Brackett-Rozinsky and Lemire, 2025)