:zap: zap

Blazing fast, structured, leveled logging in Go.

Installation

`go get -u go.uber.org/zap`

Note that zap only supports the two most recent minor versions of Go.

Quick Start

In contexts where performance is nice, but not critical, use the `SugaredLogger`. It's 4-10x faster than other structured logging packages and includes both structured and `printf`-style APIs.

```
logger, _ := zap.NewProduction()
defer logger.Sync() // flushes buffer, if any
sugar := logger.Sugar()
sugar.Infow("failed to fetch URL",
  // Structured context as loosely typed key-value pairs.
  "url", url,
  "attempt", 3,
  "backoff", time.Second,
)
sugar.Infof("Failed to fetch URL: %s", url)
```

When performance and type safety are critical, use the `Logger`. It's even faster than the `SugaredLogger` and allocates far less, but it only supports structured logging.

```
logger, _ := zap.NewProduction()
defer logger.Sync()
logger.Info("failed to fetch URL",
  // Structured context as strongly typed Field values.
  zap.String("url", url),
  zap.Int("attempt", 3),
  zap.Duration("backoff", time.Second),
)
```

See the documentation and FAQ for more details.

Performance

For applications that log in the hot path, reflection-based serialization and string formatting are prohibitively expensive — they're CPU-intensive and make many small allocations. Put differently, using `encoding/json` and `fmt.Fprintf` to log tons of `interface{}` s makes your application slow.

Zap takes a different approach. It includes a reflection-free, zero-allocation JSON encoder, and the base `Logger` strives to avoid serialization overhead and allocations wherever possible. By building the high-level

`SugaredLogger` on that foundation, zap lets users *choose* when they need to count every allocation and when they'd prefer a more familiar, loosely typed API.

As measured by its own benchmarking suite, not only is zap more performant than comparable structured logging packages — it's also faster than the standard library. Like all benchmarks, take these with a grain of salt.[1]

Log a message and 10 fields:

Package	Time	Time % to zap	Objects Allocated
:zap: zap	862 ns/op	+0%	5 allocs/op
:zap: zap (sugared)	1250 ns/op	+45%	11 allocs/op
zerolog	4021 ns/op	+366%	76 allocs/op
go-kit	4542 ns/op	+427%	105 allocs/op
apex/log	26785 ns/op	+3007%	115 allocs/op
logrus	29501 ns/op	+3322%	125 allocs/op
log15	29906 ns/op	+3369%	122 allocs/op

Log a message with a logger that already has 10 fields of context:

Package	Time	Time % to zap	Objects Allocated
:zap: zap	126 ns/op	+0%	0 allocs/op
:zap: zap (sugared)	187 ns/op	+48%	2 allocs/op
zerolog	88 ns/op	-30%	0 allocs/op
go-kit	5087 ns/op	+3937%	103 allocs/op
log15	18548 ns/op	+14621%	73 allocs/op
apex/log	26012 ns/op	+20544%	104 allocs/op
logrus	27236 ns/op	+21516%	113 allocs/op

Log a static string, without any context or `printf` -style templating:

Package	Time	Time % to zap	Objects Allocated
:zap: zap	118 ns/op	+0%	0 allocs/op
:zap: zap (sugared)	191 ns/op	+62%	2 allocs/op
zerolog	93 ns/op	-21%	0 allocs/op
go-kit	280 ns/op	+137%	11 allocs/op
standard library	499 ns/op	+323%	2 allocs/op
apex/log	1990 ns/op	+1586%	10 allocs/op

logrus	3129 ns/op	+2552%	24 allocs/op
log15	3887 ns/op	+3194%	23 allocs/op

Development Status: Stable

All APIs are finalized, and no breaking changes will be made in the 1.x series of releases. Users of semver-aware dependency management systems should pin zap to `^1`.

Contributing

We encourage and support an active, healthy community of contributors — including you! Details are in the contribution guide and the code of conduct. The zap maintainers keep an eye on issues and pull requests, but you can also report any negative conduct to oss-conduct@uber.com. That email list is a private, safe space; even the zap maintainers don't have access, so don't hesitate to hold us to a high standard.

---

[1] In particular, keep in mind that we may be benchmarking against slightly older versions of other packages. Versions are pinned in the benchmarks/go.mod file. ↩