

From YAML Blues to Autocomplete: Schema-First Config for OpenTelemetry Collector Components

Jan Korona · Splunk by Cisco

About me

Observability Day
NORTH AMERICA



Staff Software Engineer at Splunk by Cisco

- **Lives** in Lublin, Poland
- **Works** on the Splunk Observability Cloud and the Splunk OpenTelemetry Collector
- **Member** of the OpenTelemetry community
- **Contributor to —** OpenTelemetry Collector, OpenTelemetry Collector Contrib, and OpenTelemetry Java Instrumentation
- **Driving** schema-based configuration for Collector components

Four acts

Observability Day
NORTH AMERICA

1 **The problem**
Why Collector config is costly to write, maintain, and document today

2 **What's been tried**
Approaches the community already attempted — and why each fell short

3 **Schema-first generation**
Flipping the direction: one YAML schema generates structs, JSON Schema, and docs

4 **Where we are & what's next**
Real schemas bootstrapped today, the migration path, and how to get involved

Observability Day
NORTH AMERICA

ACT 1

The problem

Why Collector config is costly to write, maintain, and document today

A quick primer: how Collector config works

Observability Day NORTH AMERICA

Every Collector config has two parts: what's configured, and how it's wired together.

```yaml
receivers:
  otlp/ingest:
    protocols:
      grpc: {}
processors:
  queue_batch: {}
exporters:
  debug: {}
service:
  pipelines:
    traces:
      receivers: [otlp/ingest]
      processors: [queue_batch]
      exporters: [debug]
```

1 **Component settings**

Configure an instance of a component type — a receiver, processor, or exporter.

2 **`service.pipelines`**

Enable those instances and connect them into a pipeline.

A real example: ICMP Check Receiver

Observability Day NORTH AMERICA

```go
type Config struct {
    ControllerConfig     scraperhelper.ControllerConfig `mapstructure:",squash"`
    MetricsBuilderConfig metadata.MetricsBuilderConfig  `mapstructure:",squash"`
    Targets              []PingTarget                   `mapstructure:"targets"`
}

type PingTarget struct {
    Host         string        `mapstructure:"host"`
    PingCount    int           `mapstructure:"ping_count,omitempty"`
    PingTimeout  time.Duration `mapstructure:"ping_timeout,omitempty"`
    PingInterval time.Duration `mapstructure:"ping_interval,omitempty"`
}

func (c *Config) Validate() error {
    var err error
    if len(c.Targets) == 0 {
        return multierr.Append(err, errMissingTarget)
    }
    return err
}

func createDefaultConfig() component.Config {
    return &Config{...}
}
```

Every one of these is hand-written, with no link back to the others:

- **The struct shape —** `Config`, `PingTarget` — has no mechanical link to documentation; a renamed field doesn't fail a build.
- **`Validate()` —** required-field logic for `targets` lives only in this Go method; nothing else in the repo knows that rule exists.
- **`createDefaultConfig()` —** again, only this Go function keeps the default values for config fields.

Maintenance cost

1

Documentation drift

Go config structs and README docs evolve independently and diverge over time — users configure against docs that no longer match the code.

2

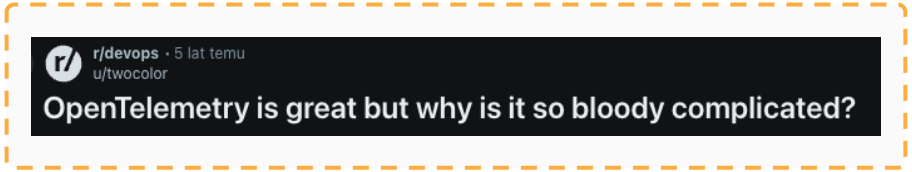
Maintenance overhead

No standardized pattern for defining a component's configuration — every author reinvents validation, defaults, and docs format from scratch.

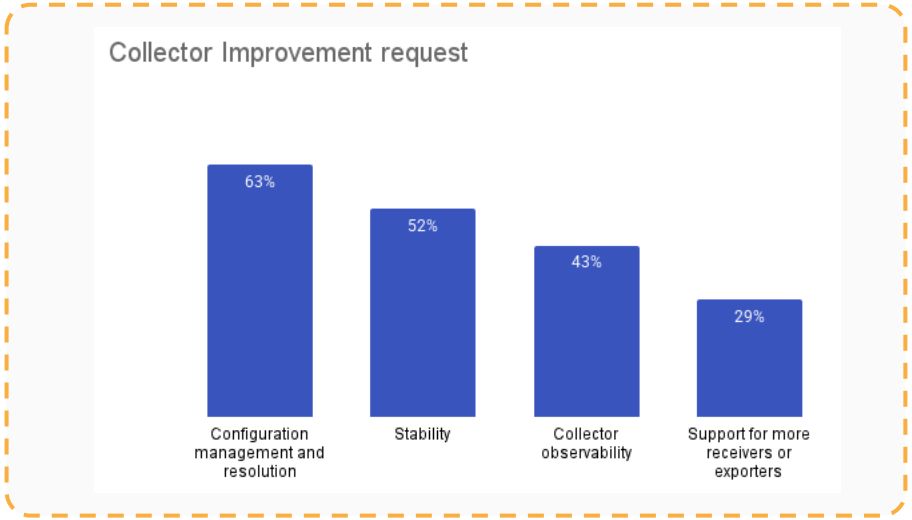
3

No validation or autocomplete

Without a JSON Schema, config editors can't flag typos or missing fields before you run the Collector and watch it fail at startup.







Observability Day
NORTH AMERICA

ACT 2

What's been tried

Approaches the community already attempted — and why each fell short

Attempts

1

ConfigSchema API

Dynamic analysis of Go code, pulling in every dependency to resolve types. Too complex, too incomplete to maintain.

RETIRED

2

AST-parsing tools

AST-parses an existing hand-written Go struct into a starter JSON Schema.

STILL IN USE

The pivot: generate FROM schema, not TO schema

Observability Day NORTH AMERICA

OLD DIRECTION

Go structs

(source of truth)

↓ **AST parsing (`schemagen`)**
↓ **JSON Schema**

Lossy: validation & defaults that aren't in the Go code are lost.

SCHEMA-FIRST DIRECTION

YAML schema

(source of truth)

generates:

- Go code
- JSON Schema
- Docs

Template-based generation, not analysis — predictable and reversible.

Observability Day
NORTH AMERICA

ACT 3

Schema-first generation

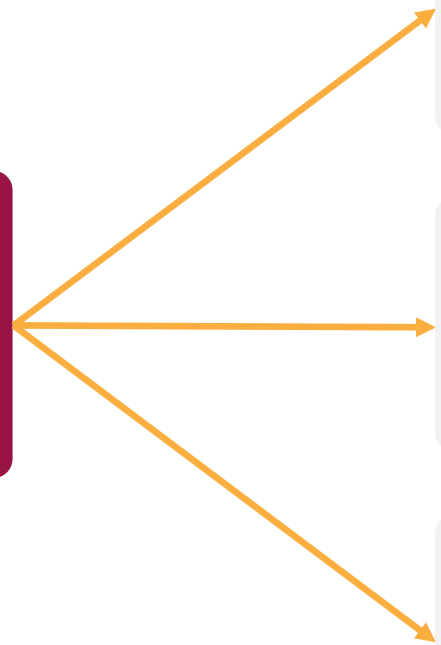
Flipping the direction: one YAML schema generates structs, JSON Schema, and docs

One schema, three generated outputs

Observability Day NORTH AMERICA

`metadata.yaml`

`config:` section



`generated_config.go`

Structs, `mapstructure` tags, `Validate()`, `createDefaultConfig()` — plus `generated_config_test.go`

`config.schema.json`

Full JSON Schema of the component's configuration

`README.md`

Synced documentation, never hand-edited again

Why YAML for the source of truth?

Observability Day
NORTH AMERICA

1

Human-readable

Easier for component developers to author and maintain than JSON.

2

Fits existing infrastructure

A natural extension of the `metadata.yaml` that `mdatagen` already uses to generate metrics builder configs.

3

Extensible

Custom fields capture domain-specific configuration and provide escape hatches for fields that still need custom implementation, validation, or default-value logic.

Anatomy of schema


```yaml
config:
  properties:
    controller_config:
      $ref: /scraper/scraperhelper.controller_config
      embed: true
    metrics_builder_config:
      $ref: ./internal/metadata.metrics_builder_config
      embed: true
    targets:
      type: slice
      description: The list of targets to be pinged.
      values:
        properties:
          host:
            type: string
            description: The hostname or IP address to ping.
          ping_count:
            type: int
            default: 3
            description: Number of pings to send per interval.
          ping_timeout:
            type: duration
            default: 1s
            description: Timeout for a single ping.
          ping_interval:
            type: duration
            default: 5s
            description: Interval between ping batches.
required: [targets]
```

Types

Simple and complex types like maps or slices are supported. Alias types such as `duration` or `opaque_string` map straight onto Go types and config packages.

Explicit defaults

Define default values explicitly to generate `createDefaultConfig()` — also reflected in the generated documentation.

Validation rules

A set of built-in validation rules is already supported and used to generate the `Validate()` method.

Descriptions

Engine uses `description` annotations to build documentation tables in `README`.

Schema composition

Reference a shared type by full package path, or a sibling schema in the same repo.

```yaml
controller_config:
  $ref: go.opentelemetry.io/collector/scraper/scraperhelper.controller_config
  embed: true
metrics_builder_config:
  $ref: ./internal/metadata.metrics_builder_config
  embed: true
```

Use mapstructure `squash` and Go type embedding in your configs, with `embed: true`.

Go escape hatch

Observability Day NORTH AMERICA

Need a hand-written validator? Declare it, and the generator calls your code.

Custom validator

```
endpoint:
  type: string
  go_struct:
    custom_validator:
      name: validateEndpoint
```

Pointers & optionals

```
endpoint:
  type: string
  go_struct:
    pointer: true
    optional: true
```

Custom types

```
endpoint:
  type: string
  go_struct:
    type: com.github/somelib
      .custom_type
```

Go-specific syntax elements — pointers, optionals, and arbitrary custom types — are all supported through the same `go_struct` escape hatch.

Migration philosophy

Compatibility is non-negotiable

1 **What users see**

The YAML configuration file format MUST remain 100% backward compatible. A migrated component accepts the exact same config it did before — this is a build-time and internal-API change, not a user-facing one.

2 **What contributors may change**

Generated code can break Go API compatibility in favor of consistent naming across all components — if an existing struct doesn't follow the generator's conventions, long-term maintainability wins over preserving today's exact Go symbol names.

This asymmetry is deliberate: end users should never notice a migration happened. Component authors and downstream Go importers might.

Observability Day
NORTH AMERICA

ACT 4

Where we are & what's next

Real schemas bootstrapped today, the migration path, and how to get involved

Status today

MOSTLY GENERATED

Shared core packages

Config types like `confighttp`, `configgrpc`, `configtls`, and others already ship a `generated_config.go` from their own `metadata.yaml` `exported_configs` — real production code, not a sample.

NEXT STEP

Full components

Receivers, processors, exporters, and extensions still hand-write `config.go` today.

Why this matters for the ecosystem

Observability Day
NORTH AMERICA

1

Component authors

One place to define config — type, validation, and defaults — instead of four.

2

Maintainers

A migration path for core and contrib that keeps the end-user config format compatible throughout.

3

End users

JSON Schema means real-time validation and autocomplete in any editor that supports it.

Reduced documentation drift, a consistent developer experience, and config tooling that works the way users already expect it to.

Observability Day
NORTH AMERICA

Get involved

Thank you!

- Read the RFC and guide in `open-telemetry/opentelemetry-collector`
- Try `mdatagen` on a component that hasn't migrated yet
- Come find me after the talk, or open an issue upstream



From YAML Blues to Autocomplete: Schema-First Config for OpenTelemetry Collector Components