23 Feb 2023

Observability Testing

or who watches the watchers

Golang Meetup, Amsterdam

Anatoly Rugalev

Who are you, again?

Toly Rugalev

Software Nerd.

- Go: 6 years
- Total: 11 years

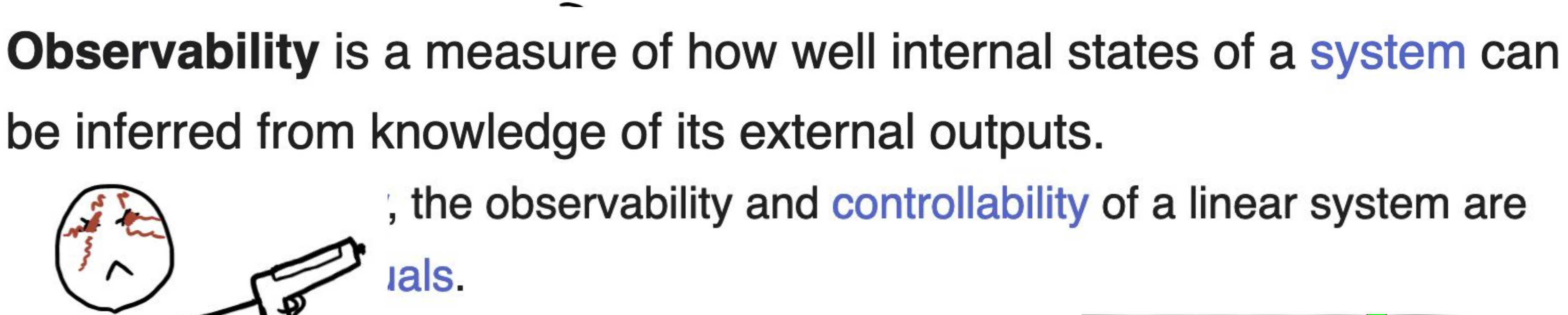
Passion: **Developer Experience**

Position: SE @ MessageBird

https://github.com/AnatolyRugalev

Today's Topic is

Refresher: Observability

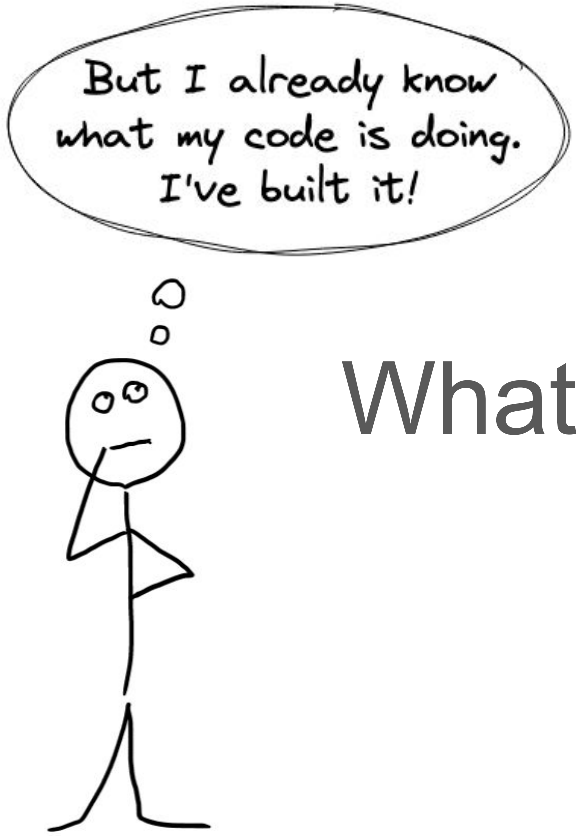


Observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs.

, the observability and controllability of a linear system are

ials.

Observability for Humans

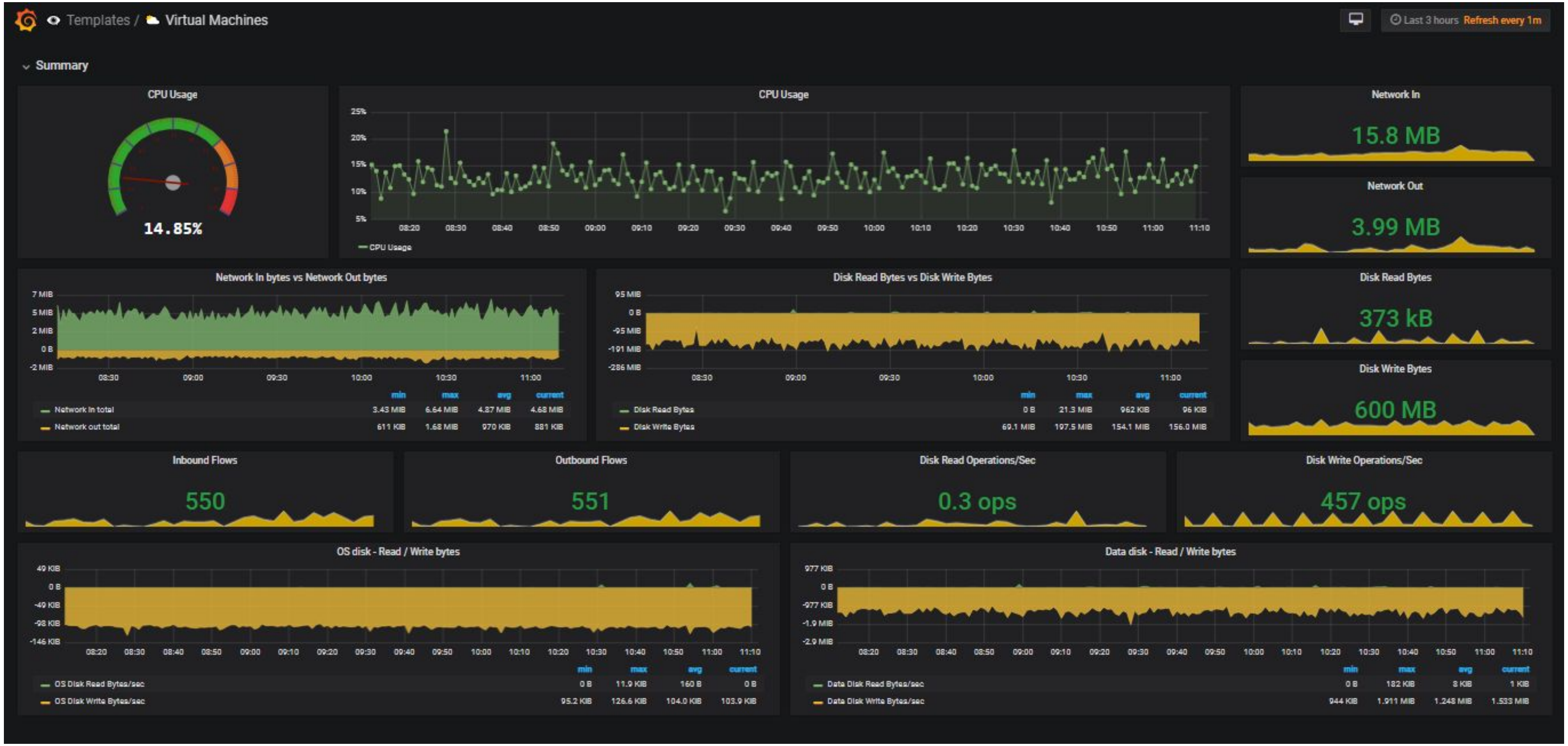


What is your code doing?

Observability for Humans

What is your code **ACTUALLY** doing?

Refresher: Instruments


Templates / Virtual Machines
Last 3 hours Refresh every 1m
Summary
CPU Usage
14.85%
CPU Usage
CPU Usage
Network In
15.8 MB
Network Out
3.99 MB
Network In bytes vs Network Out bytes
Disk Read Bytes vs Disk Write Bytes
Disk Read Bytes
373 kB
Disk Write Bytes
600 MB
Inbound Flows
550
Outbound Flows
551
Disk Read Operations/Sec
0.3 ops
Disk Write Operations/Sec
457 ops
OS disk - Read / Write bytes
Data disk - Read / Write bytes

	min	max	avg	current
Network In total	3.43 MiB	6.64 MiB	4.87 MiB	4.68 MiB
Network out total	611 KiB	1.68 MiB	970 KiB	881 KiB

	min	max	avg	current
Disk Read Bytes	0 B	21.3 MiB	962 KiB	96 KiB
Disk Write Bytes	69.1 MiB	197.5 MiB	154.1 MiB	156.0 MiB

	min	max	avg	current
OS Disk Read Bytes/sec	0 B	11.9 KiB	160 B	0 B
OS Disk Write Bytes/sec	95.2 KiB	126.6 KiB	104.0 KiB	103.9 KiB

	min	max	avg	current
Data Disk Read Bytes/sec	0 B	182 KiB	8 KiB	1 KiB
Data Disk Write Bytes/sec	944 KiB	1.911 MiB	1.248 MiB	1.533 MiB

Ready?

Observability Testing

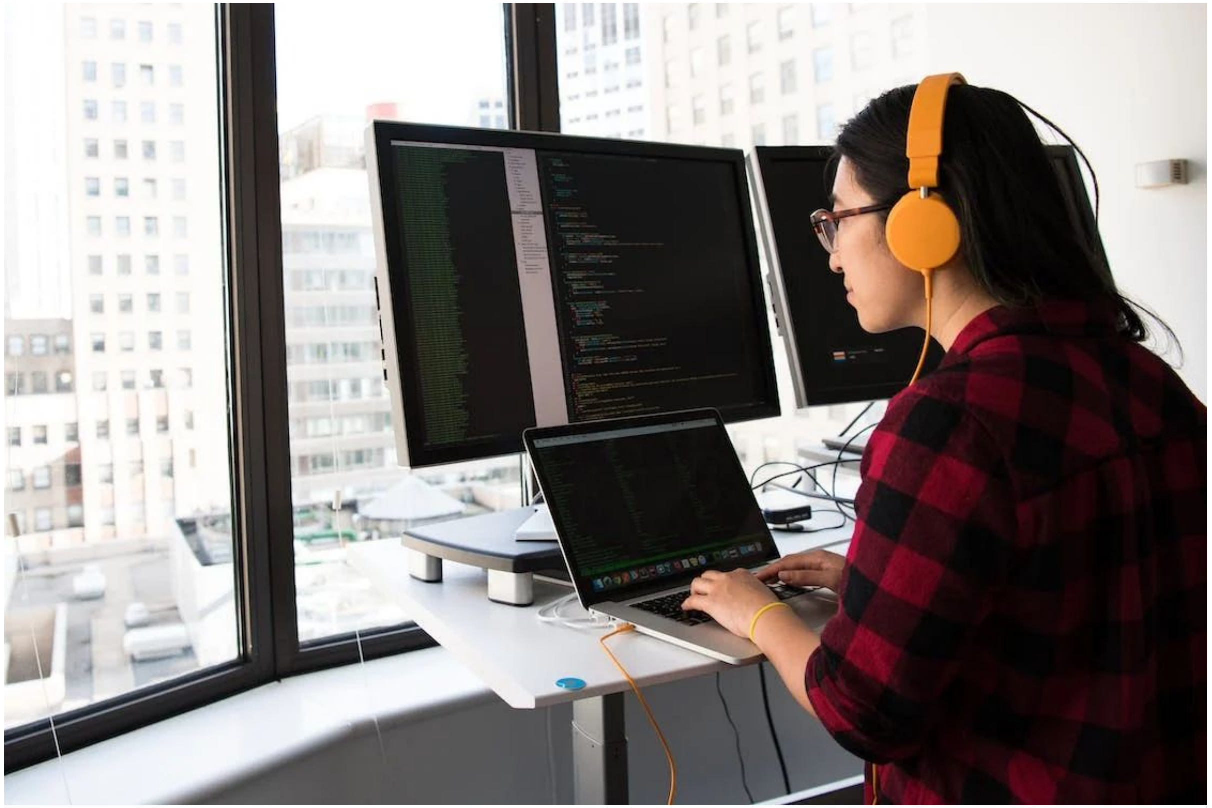
Story Time

Story Time

Story Time

Story Time

Story Time



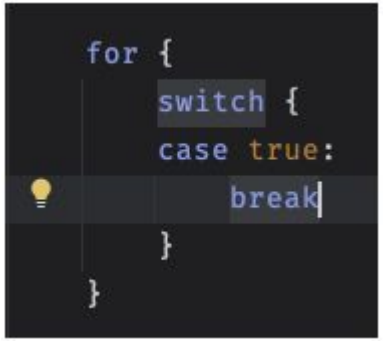
Story Time

Story Time

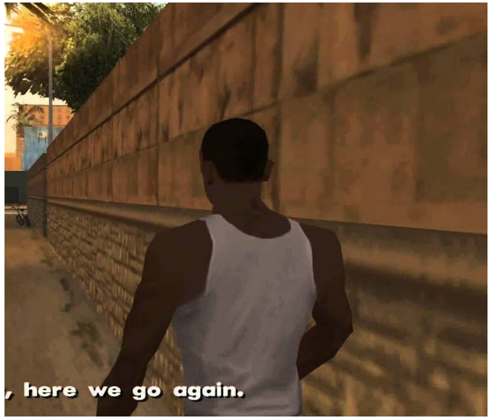
Story Time

Five Whys

1. Why did production go down?
   - Because our database got overloaded
2. Why was it overloaded?
   - Because one of the jobs stuck in an infinite loop for $\boxed{\text{two weeks}}$
3. Why did it stuck?
   - Human error: \`break\` was breaking from \`switch\` instead of \`for\`



```
for {
    switch {
    case true:
        break
    }
}
```

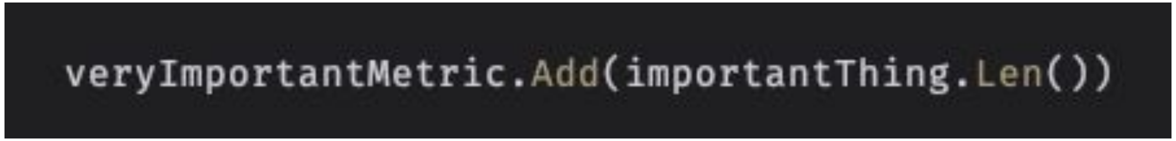


Going Deeper

1. Why didn’t we notice the issue earlier?
    - Because the alert didn’t fire
2. Why didn’t it fire?
    - Because the metric it relies on reported incorrect data
3. Why?
    - HUMAN ERROR
4. Why humans make mistakes?
    - … what if…
    - … I told you …
    - …
    -

Verdict?

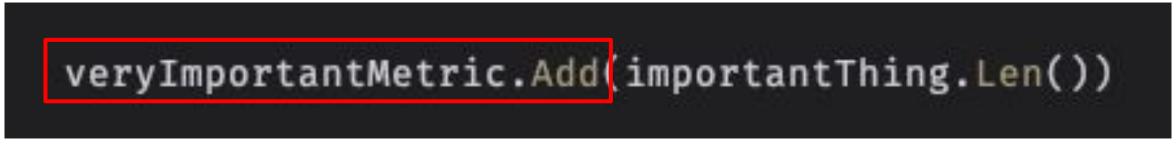
Humans suck



```
veryImportantMetric.Add(importantThing.Len())
```

What if…

… we tested what's REALLY important to us



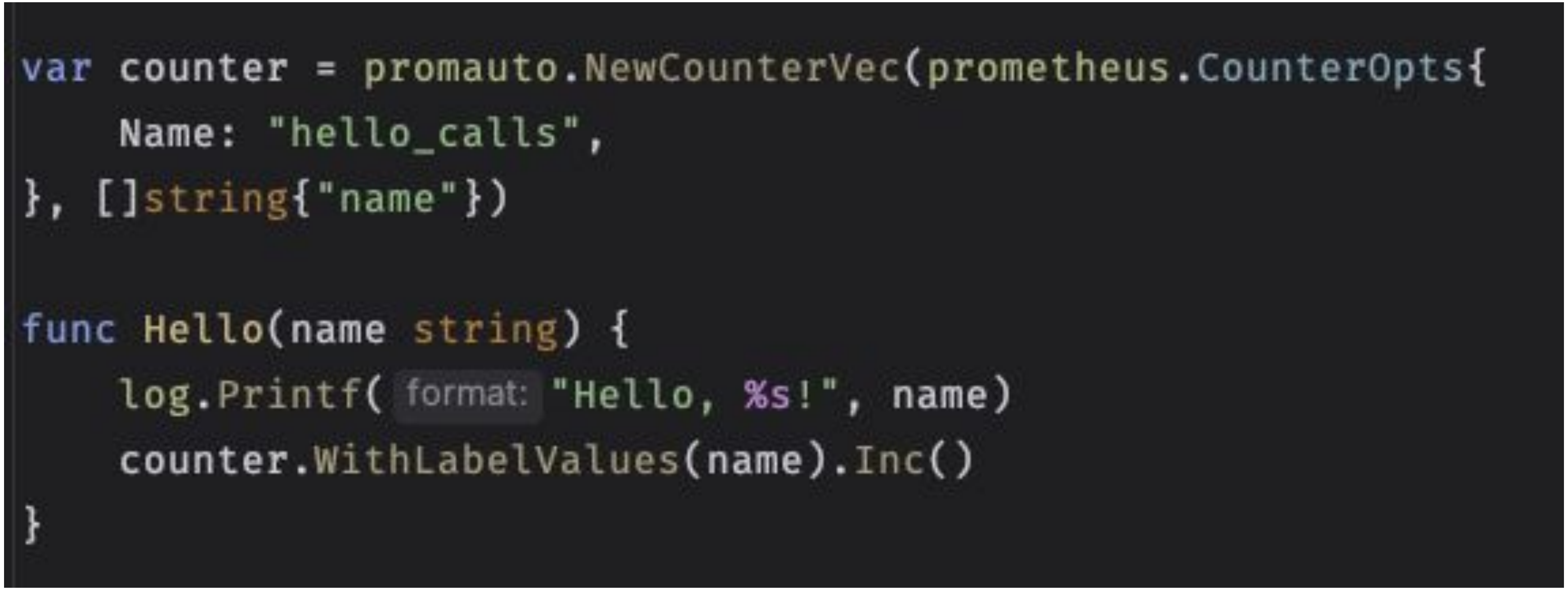
```
veryImportantMetric.Add(importantThing.Len())
```

Is it possible?

Good news: it is possible

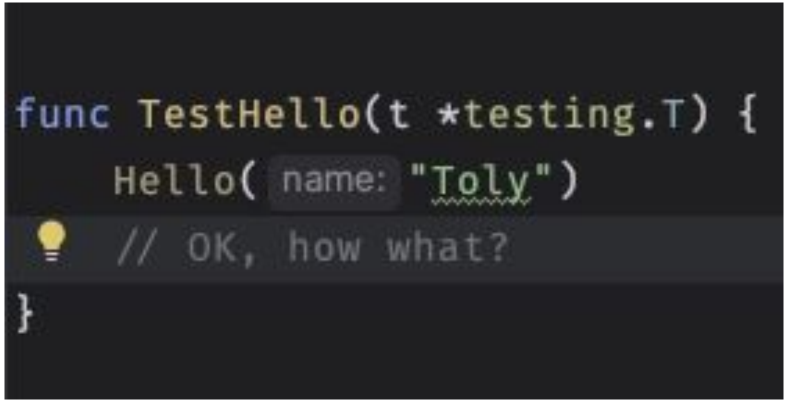
Bad news: you’ll see…

Test Subject



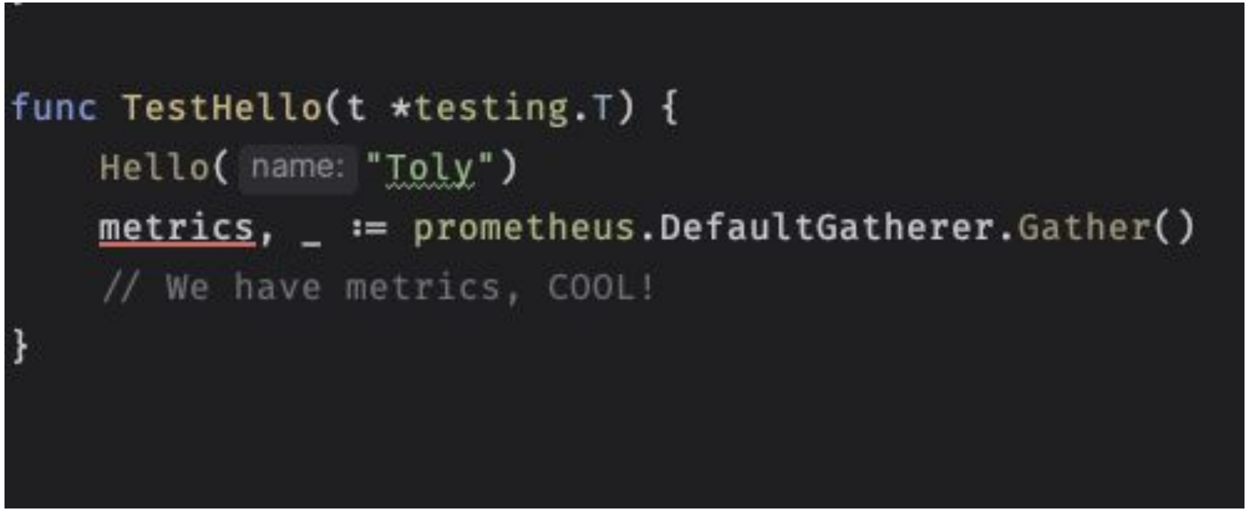
```go
var counter = promauto.NewCounterVec(prometheus.CounterOpts{
    Name: "hello_calls",
}, []string{"name"})

func Hello(name string) {
    log.Printf( format: "Hello, %s!", name)
    counter.WithLabelValues(name).Inc()
}
```

Let's write a test…



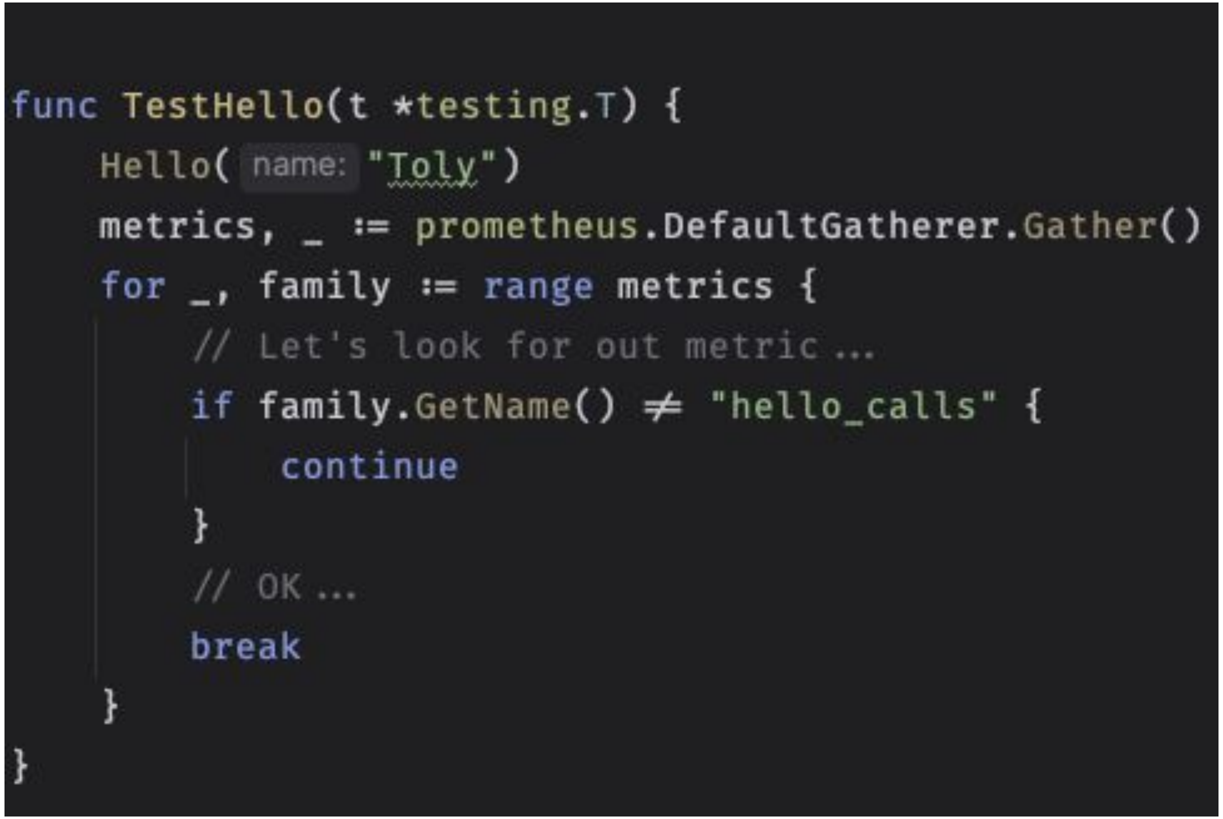
```
func TestHello(t *testing.T) {
    Hello( name: "Toly")
    // OK, how what?
}
```

Promising!



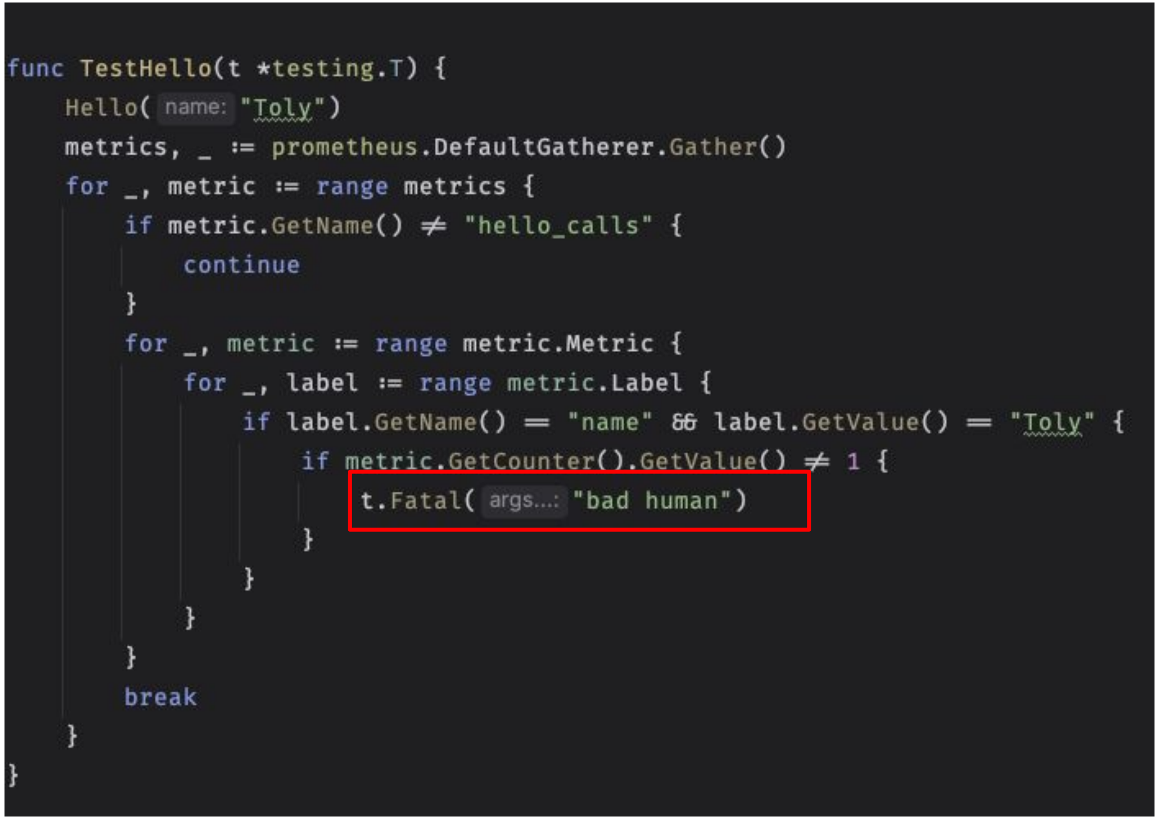
```
func TestHello(t *testing.T) {
    Hello( name: "Toly")
    metrics, _ := prometheus.DefaultGatherer.Gather()
    // We have metrics, COOL!
}
```

Alright, getting there…

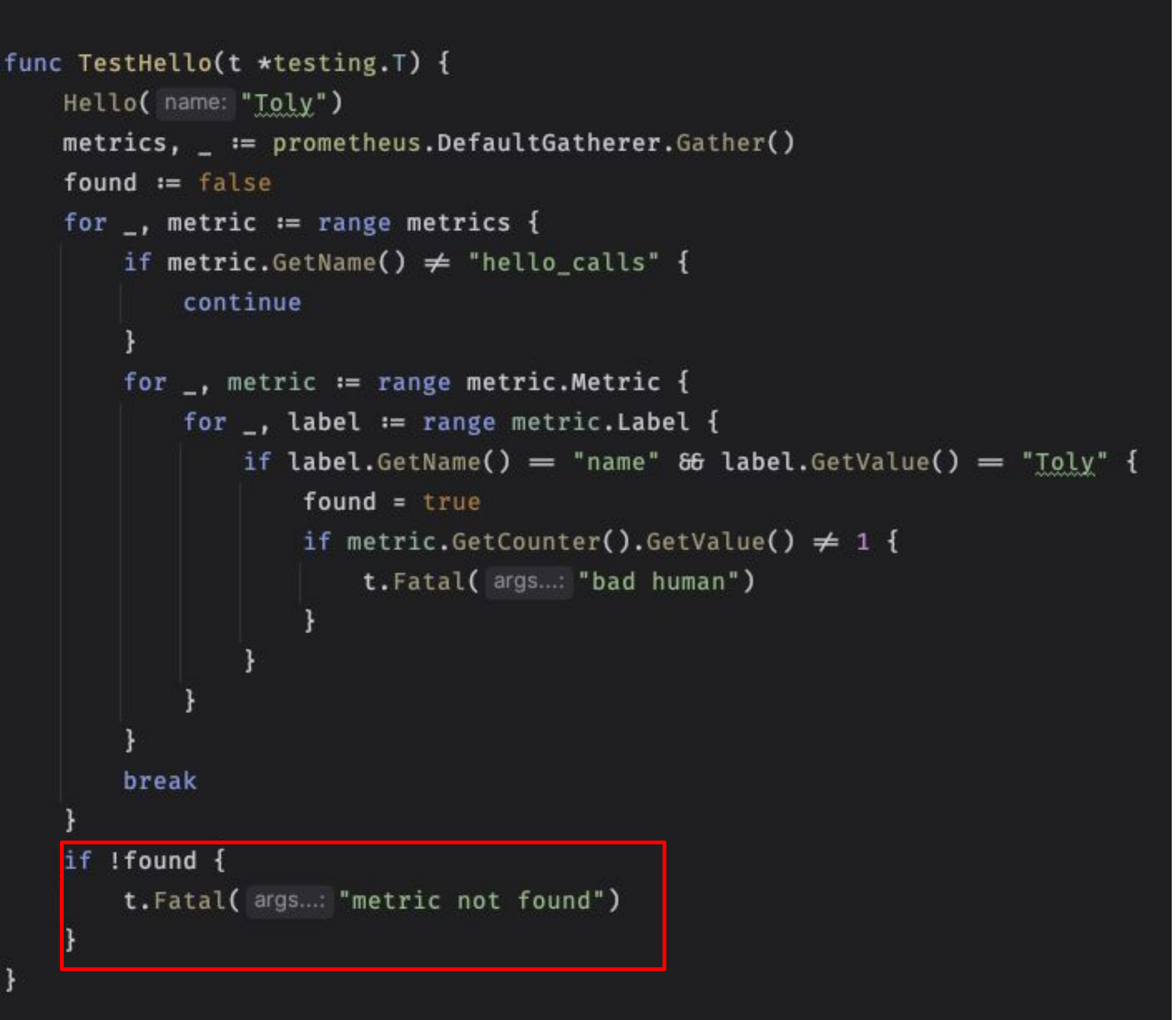


```go
func TestHello(t *testing.T) {
	Hello( name: "Toly")
	metrics, _ := prometheus.DefaultGatherer.Gather()
	for _, family := range metrics {
		// Let's look for out metric ...
		if family.GetName() ≠ "hello_calls" {
			continue
		}
		// OK ...
		break
	}
}
```

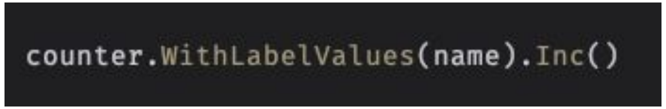
Done! Or wait…



```go
func TestHello(t *testing.T) {
    Hello( name: "Toly")
    metrics, _ := prometheus.DefaultGatherer.Gather()
    for _, metric := range metrics {
        if metric.GetName() != "hello_calls" {
            continue
        }
        for _, metric := range metric.Metric {
            for _, label := range metric.Label {
                if label.GetName() == "name" && label.GetValue() == "Toly" {
                    if metric.GetCounter().GetValue() != 1 {
                        t.Fatal( args...: "bad human")
                    }
                }
            }
        }
        break
    }
}
```

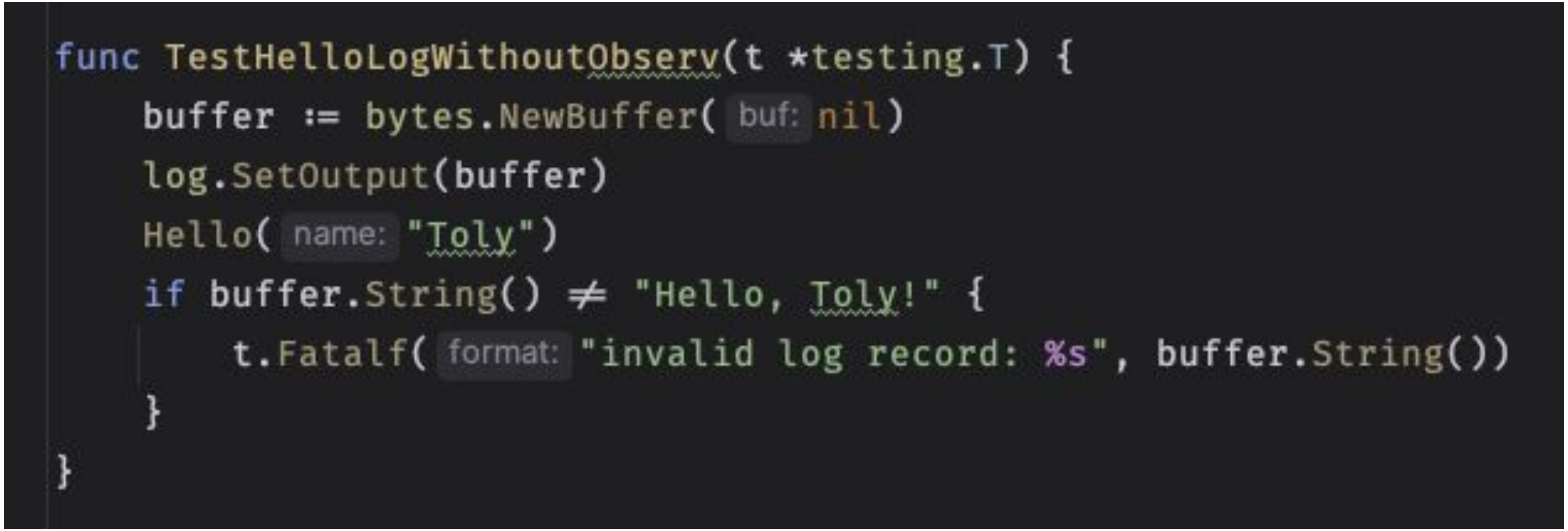


```go
func TestHello(t *testing.T) {
    Hello( name: "Toly")
    metrics, _ := prometheus.DefaultGatherer.Gather()
    found := false
    for _, metric := range metrics {
        if metric.GetName() ≠ "hello_calls" {
            continue
        }
        for _, metric := range metric.Metric {
            for _, label := range metric.Label {
                if label.GetName() == "name" && label.GetValue() == "Toly" {
                    found = true
                    if metric.GetCounter().GetValue() ≠ 1 {
                        t.Fatal( args...: "bad human")
                    }
                }
            }
        }
        break
    }
    if !found {
        t.Fatal( args...: "metric not found")
    }
}
```



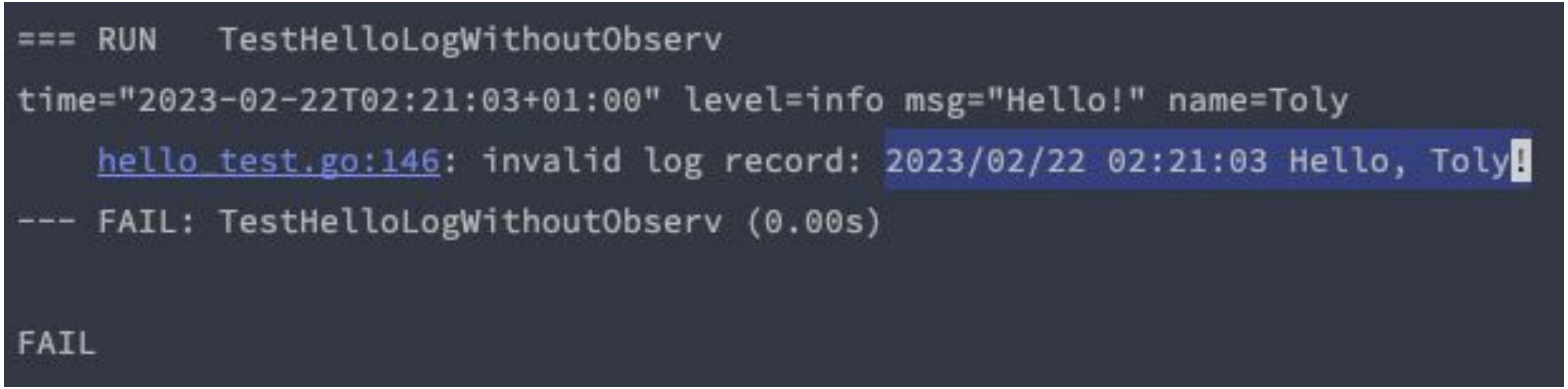
```go
counter.WithLabelValues(name).Inc()
```

Logger?



```
func TestHelloLogWithoutObserv(t *testing.T) {
    buffer := bytes.NewBuffer( buf: nil)
    log.SetOutput(buffer)
    Hello( name: "Toly")
    if buffer.String() ≠ "Hello, Toly!" {
        t.Fatalf( format: "invalid log record: %s", buffer.String())
    }
}
```

Whoops



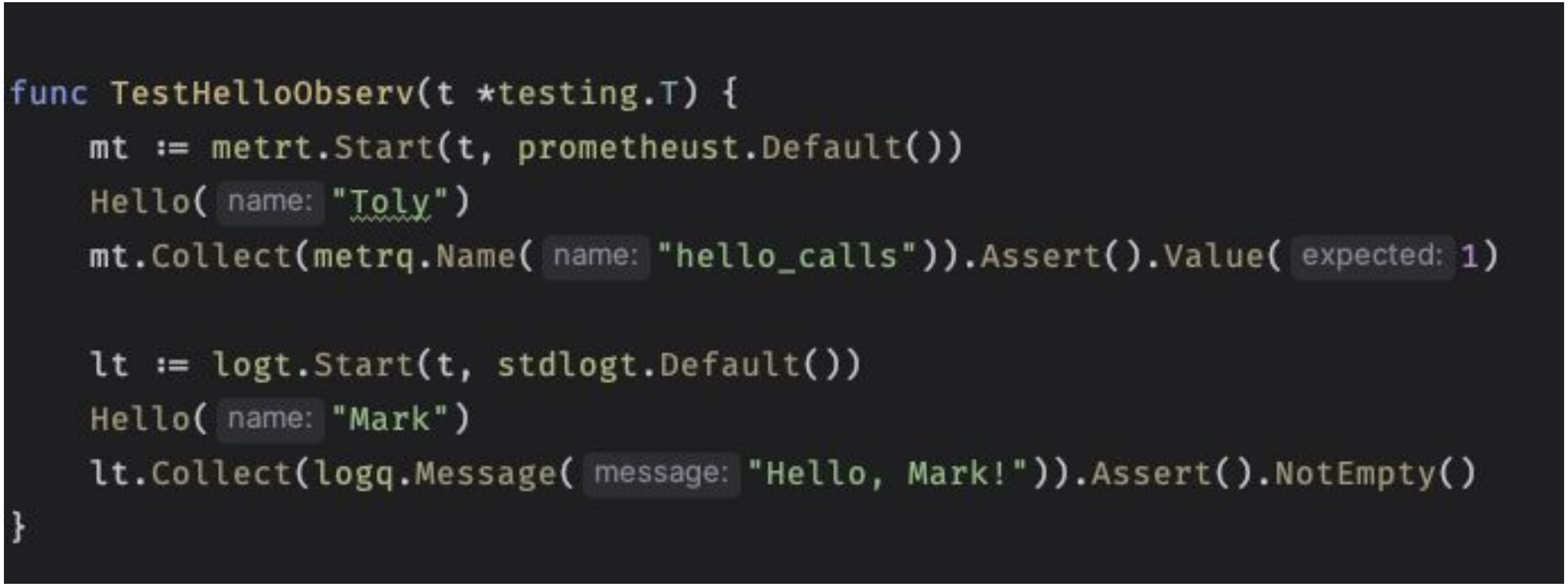
```
=== RUN   TestHelloLogWithoutObserv
time="2023-02-22T02:21:03+01:00" level=info msg="Hello!" name=Toly
    hello_test.go:146: invalid log record: 2023/02/22 02:21:03 Hello, Toly!
--- FAIL: TestHelloLogWithoutObserv (0.00s)


FAIL
```

There should be

a

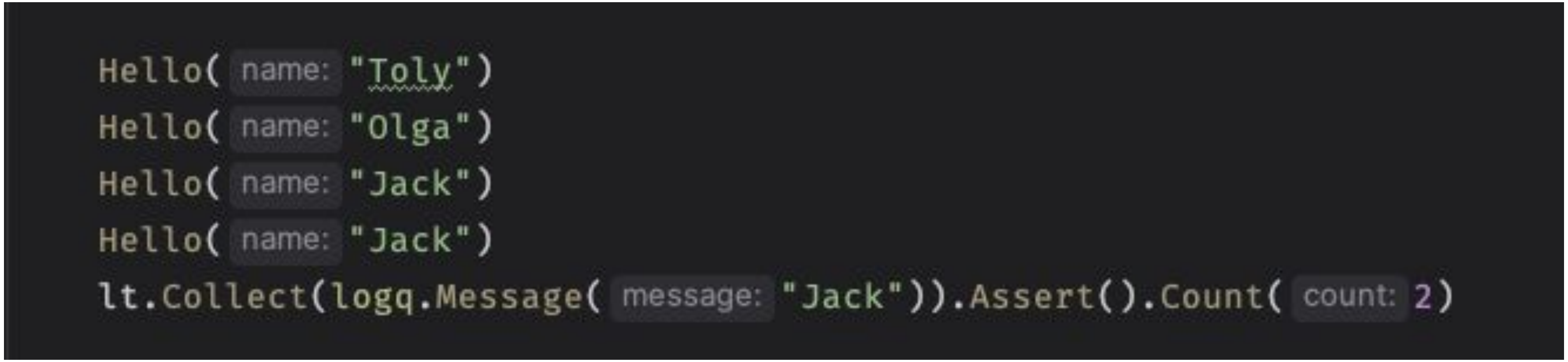
better way

Meet: `observ`

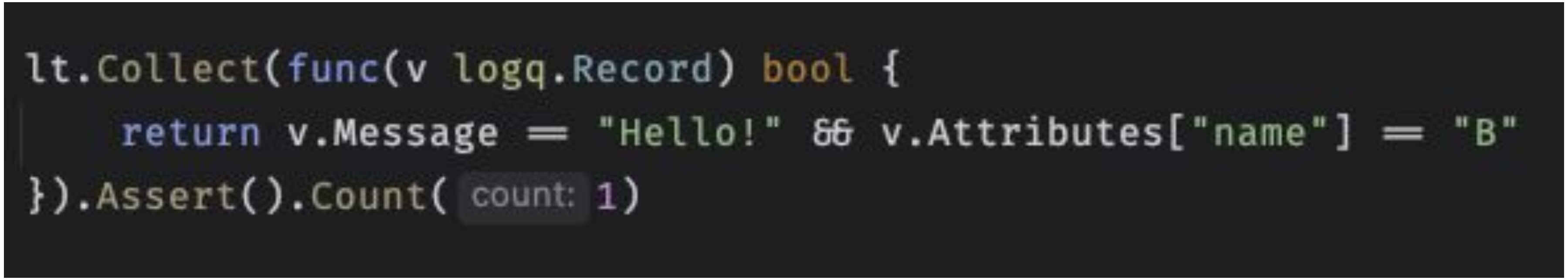


```
func TestHelloObserv(t *testing.T) {
    mt := metrt.Start(t, prometheust.Default())
    Hello( name: "Toly")
    mt.Collect(metrq.Name( name: "hello_calls")).Assert().Value( expected: 1)

    lt := logt.Start(t, stdlogt.Default())
    Hello( name: "Mark")
    lt.Collect(logq.Message( message: "Hello, Mark!")).Assert().NotEmpty()
}
```

observ: filtering

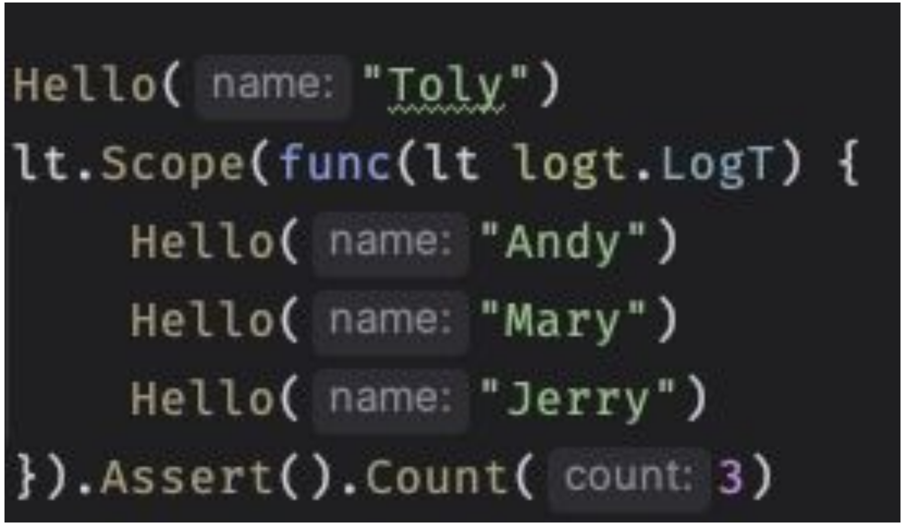


```
Hello( name: "Toly")
Hello( name: "Olga")
Hello( name: "Jack")
Hello( name: "Jack")
lt.Collect(logq.Message( message: "Jack")).Assert().Count( count: 2)
```

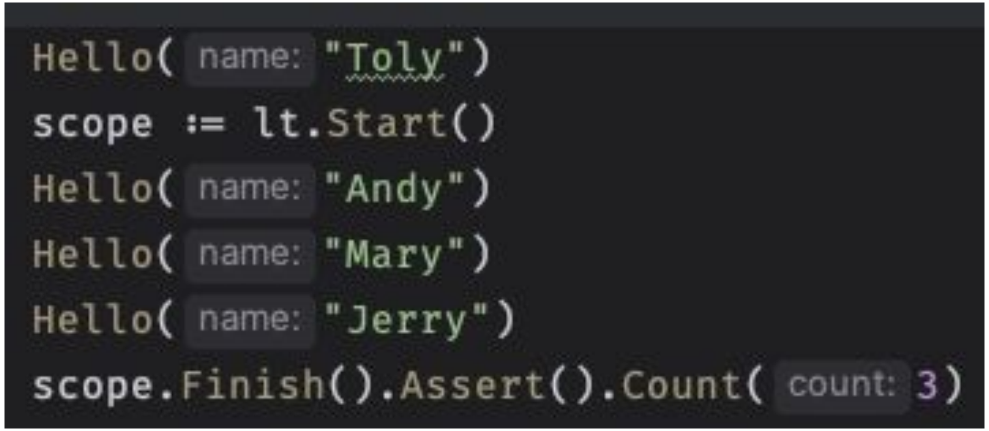


```
lt.Collect(func(v logq.Record) bool {
    return v.Message == "Hello!" && v.Attributes["name"] == "B"
}).Assert().Count( count: 1)
```

observ: scoping

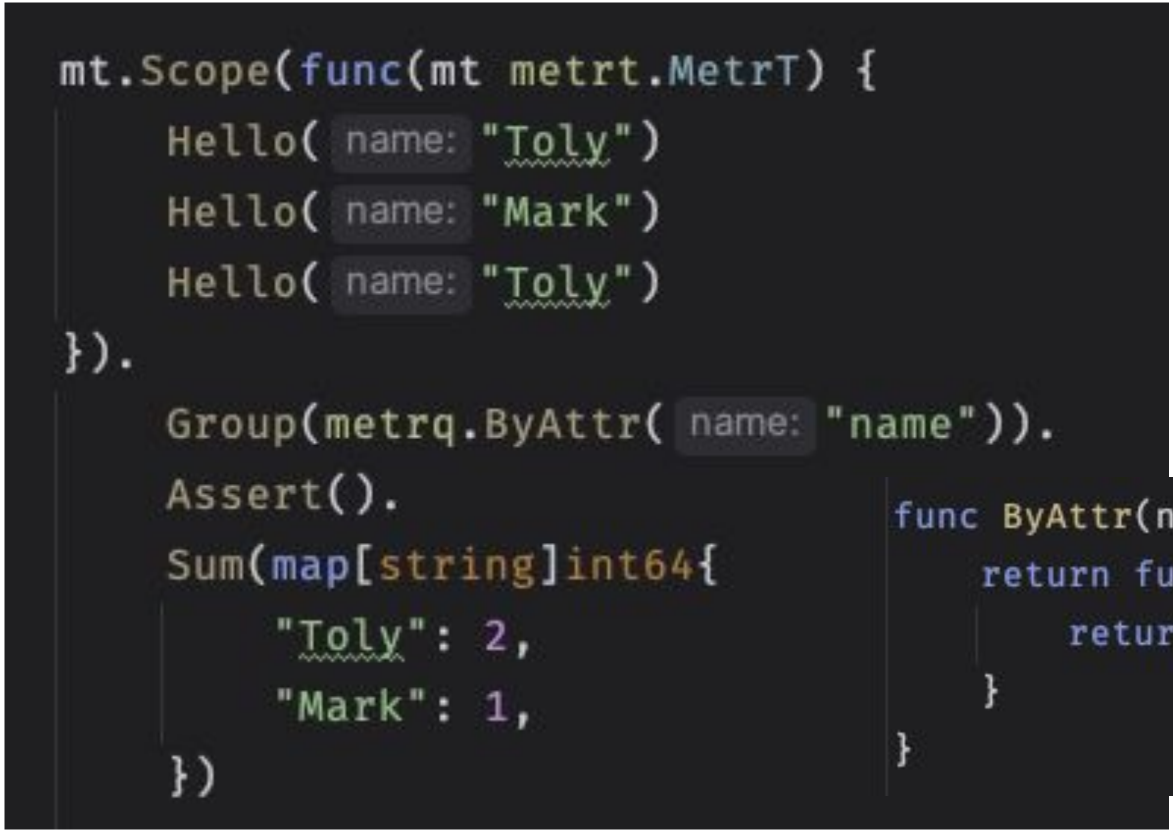


```
Hello( name: "Toly")
lt.Scope(func(lt logt.LogT) {
    Hello( name: "Andy")
    Hello( name: "Mary")
    Hello( name: "Jerry")
}).Assert().Count( count: 3)
```

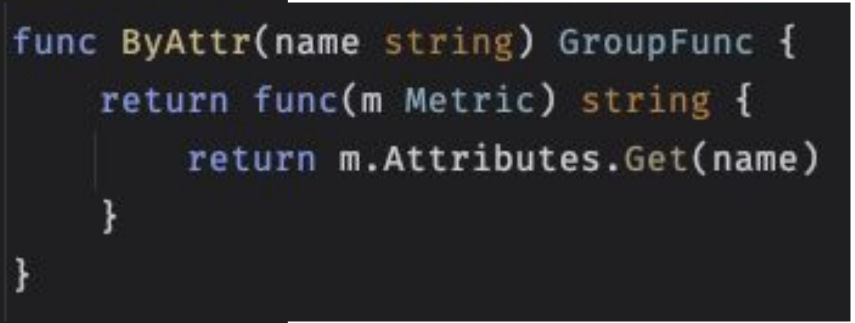


```
Hello( name: "Toly")
scope := lt.Start()
Hello( name: "Andy")
Hello( name: "Mary")
Hello( name: "Jerry")
scope.Finish().Assert().Count( count: 3)
```

observ: grouping

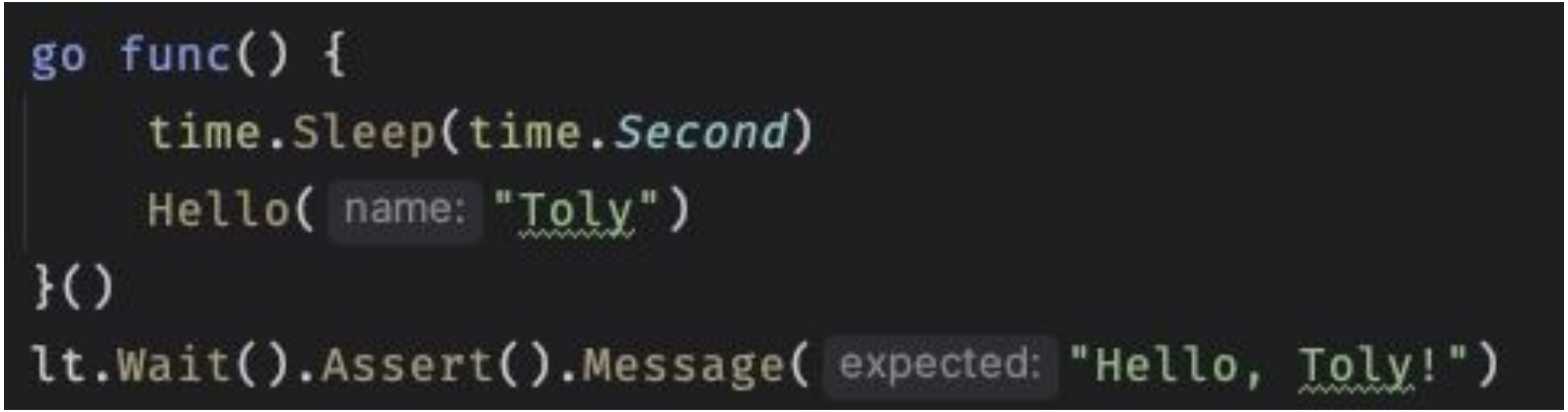


```go
mt.Scope(func(mt metrt.MetrT) {
	Hello( name: "Toly")
	Hello( name: "Mark")
	Hello( name: "Toly")
}).
	Group(metrq.ByAttr( name: "name")).
	Assert().
	Sum(map[string]int64{
		"Toly": 2,
		"Mark": 1,
	})
```



```go
func ByAttr(name string) GroupFunc {
	return func(m Metric) string {
		return m.Attributes.Get(name)
	}
}
```

observ: waiting for an event



```go
go func() {
    time.Sleep(time.Second)
    Hello( name: "Toly")
}()
lt.Wait().Assert().Message( expected: "Hello, Toly!")
```

observ: will my logger work?

- Metrics
  - Prometheus
  - OpenTelemetry
- Logs
  - log (Go stdlib)
  - Logrus

… and more is coming!

observ: potential applications

- Reduce mocks!
    - Rely on logs an counters to check if something has been executed
- Easy goroutine testing
    - Logs are you events!
- Structured logging guarantees
    - Make sure app generates logs that can be consumed by collectors
- Say “no” to regressions
    - Secure important logs and metrics with tests
- Testing infrastructure-critical components
    - Kubernetes controllers, anyone?

observ: give it a try!

Your code already generates a lot of data.

Take the full advantage of it with observ.

go-observ.io