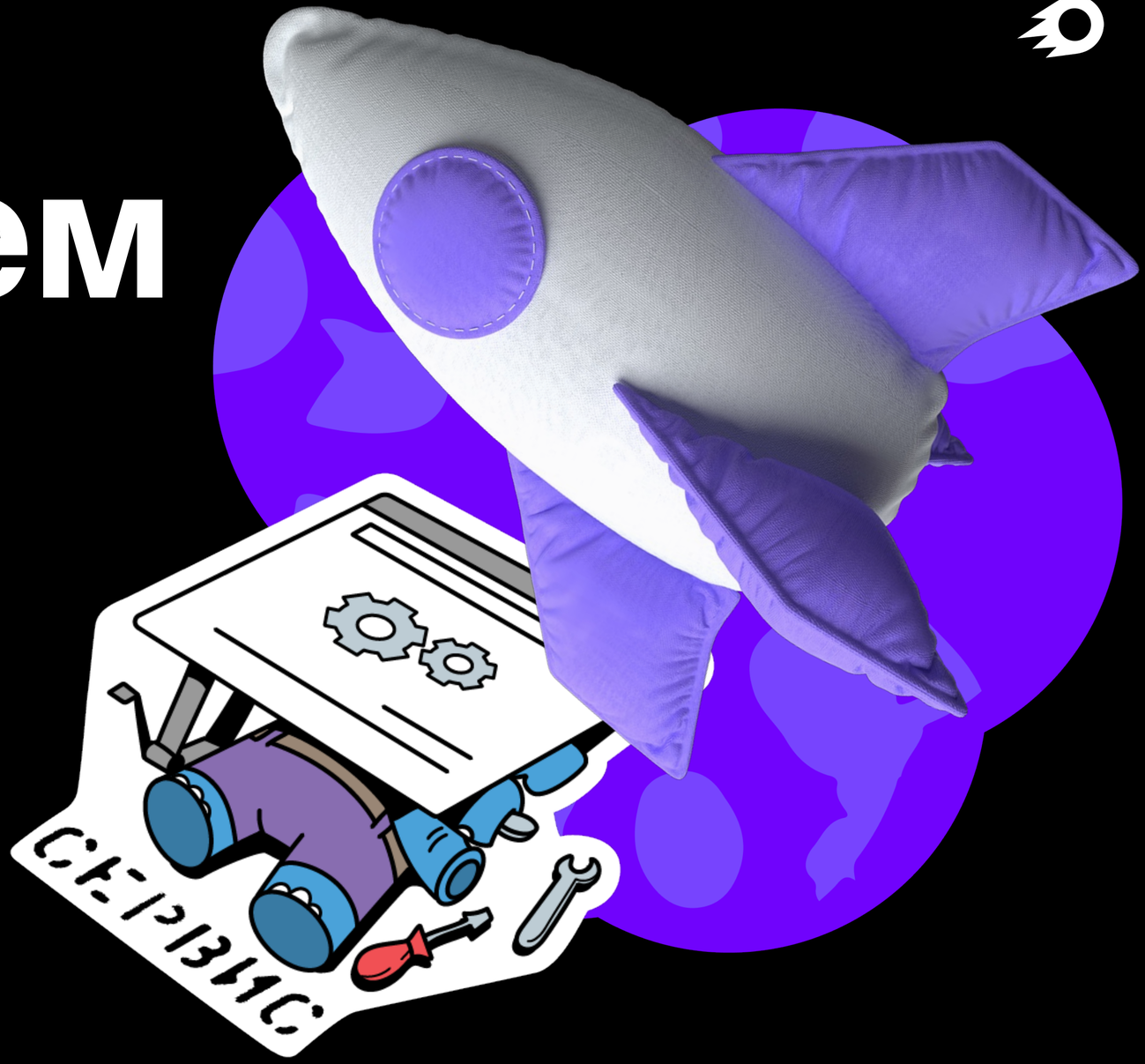
avito.tech

Санкт-Петербург 2024

Отлаживаем сервис на проде



Павел Агалецкий **Авито, ведущий инженер в платформе**

Не надо отлаживать на проде!

Содержание

1. Почему не надо отлаживать на проде
2. Но если все-таки надо, то…
    1. Логи
    2. Метрики
    3. Pprof
    4. Трейсинг
    5. Отладчик
3. Выводы

А, собственно, почему не надо?

Мало ли, что может пойти не так?

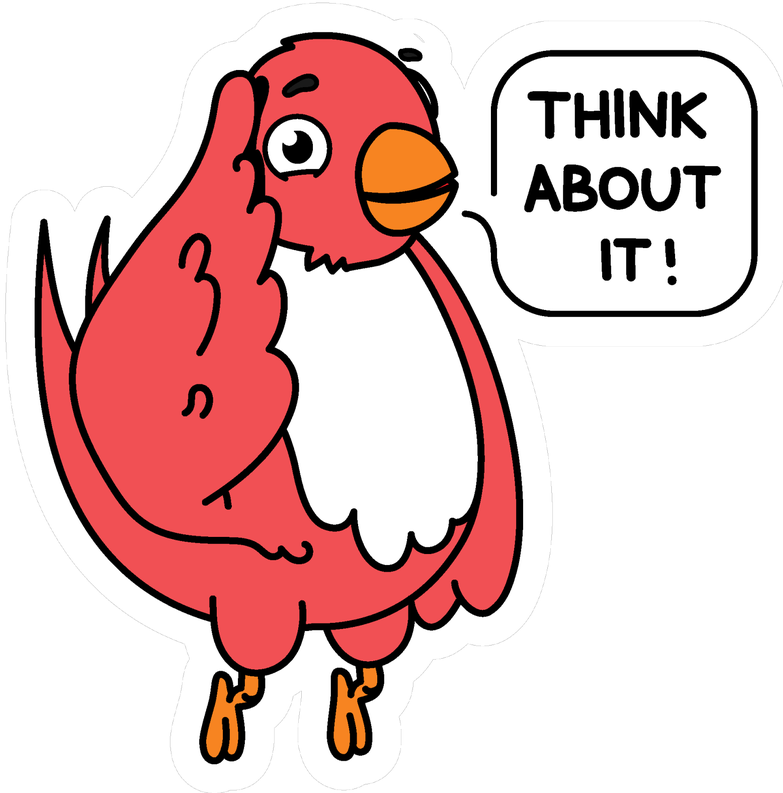
- Случайно дропнем базу?
- Выполним опасный запрос?
- Сломаем сервис еще сильнее?
- Просто нет доступов?

Давайте отладим локально?

И правда!

локально – безопаснее

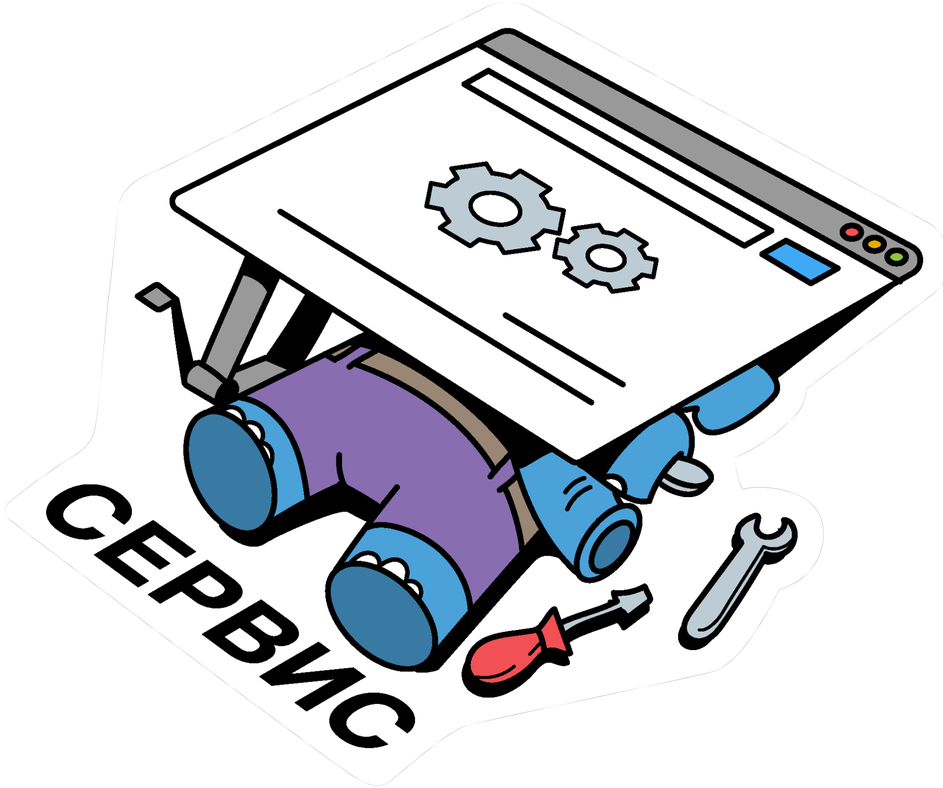
- База локальная, можно делать что угодно
- Можно выполнять любые действия
- Не страшно сломать
- **Всегда стоит выбирать именно локальную отладку!**



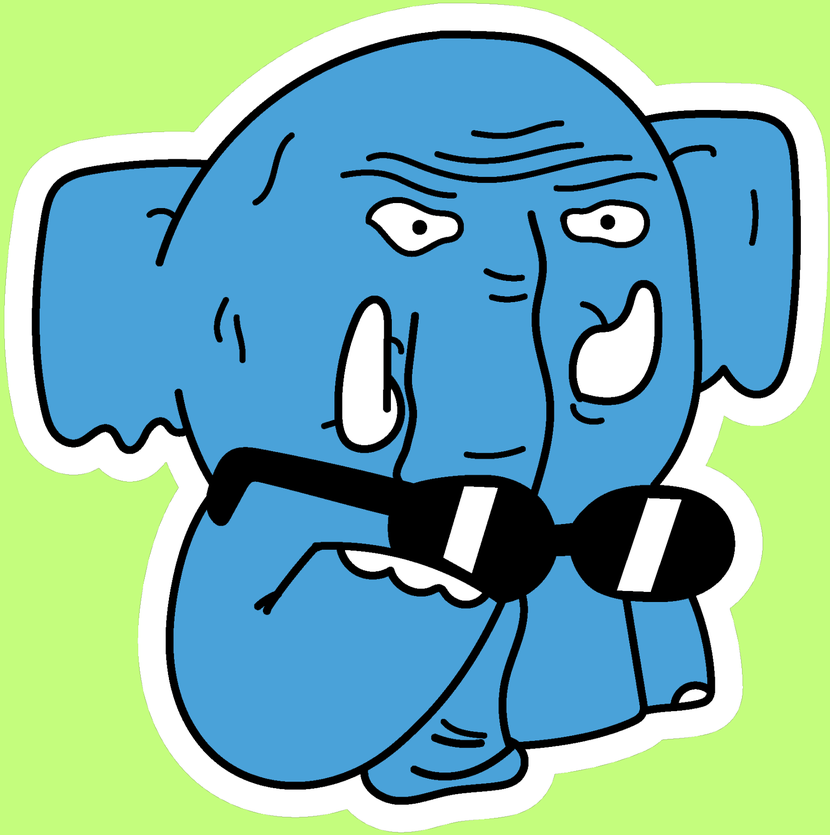
Но надо уметь понимать причину проблемы

тогда можно будет ее воспроизвести

- для этого нам надо как-то получить сведения о том, что именно в приложении идет не так



Посмотрим, что на проде...



Используем логирование

Виды логов

- plain text
- structured logs

Plain text logs

```go
package main

import "log"

func main() {
    log.Print( v...: "something happened")
}
```

```
2024/09/04 12:16:10 something happened
```

Structured logs

- Имеют какую-то структуру внутри
- Есть много библиотек для работы с ними: zap, zerolog, logrus, …
- В go 1.21 появился стандартный пакет: slog

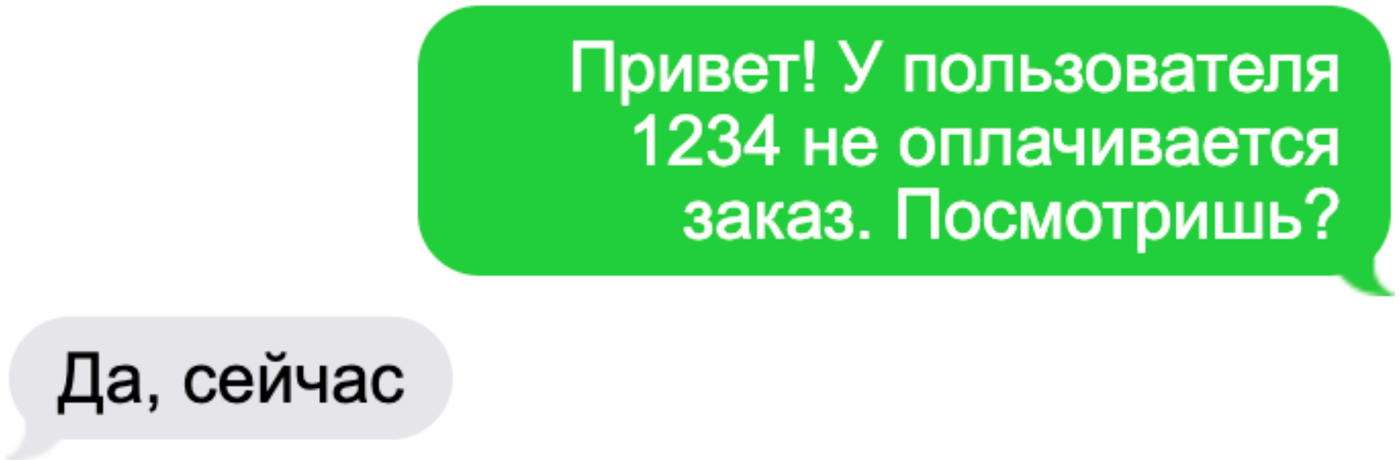
log/slog

```go
package main

import (
    "log/slog"
    "os"
)

func main() {
    slog.SetDefault(slog.New(slog.NewJSONHandler(os.Stdout, opts: nil)))

    slog.Warn(
        msg: "can not update user",
        slog.Int( key: "user_id",  value: 1234),
        slog.String( key: "error",  value: "wrong email"),
    )
}
```

log/slog

```go
package main

import (
    "log/slog"
    "os"
)

func main() {
    slog.SetDefault(slog.New(slog.NewJSONHandler(os.Stdout, opts: nil)))

    slog.Warn(
        msg: "can not update user",
        slog.Int( key: "user_id", value: 1234),
        slog.String( key: "error", value: "wrong email"),
    )
}
```

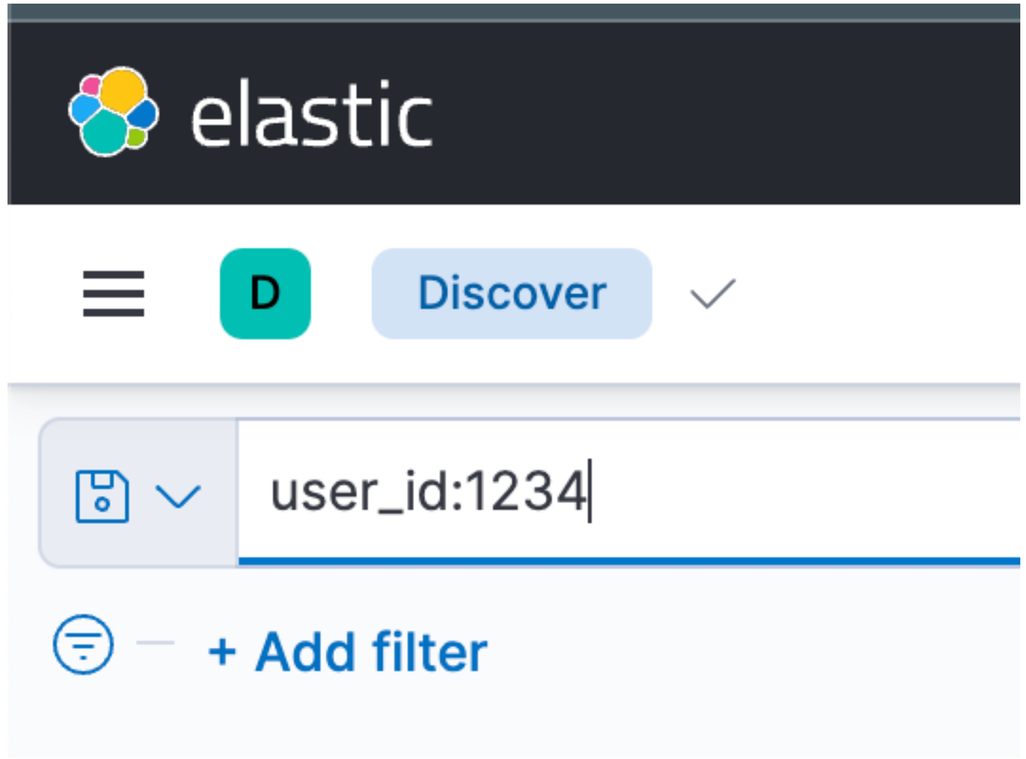
Чем полезны такие логи?



Чем полезны такие логи?

Привет! У пользователя 1234 не оплачивается заказ. Посмотришь?

Да, сейчас



Но надо логи писать. А в каком случае?

Например, в случае ошибок

Например, в случае ошибок

```go
err := u.createOrder(ctx, userID)
if err != nil {
    return fmt.Errorf( format: "user %d: can not create order: %w", userID, err)
}
```

Но надо логи писать. А в каком случае?

```go
err := u.createOrder(ctx, userID)
if err != nil {
    slog.Error( msg: "can not create order",
        slog.Int( key: "user_id", userID),
        slog.String( key: "error", err.Error()),
    )

    return nil
}
```

Но не стоит дублировать лог и возврат ошибки

```go
err := u.createOrder(ctx, userID)
if err != nil {
    slog.Error( msg: "can not create order",
        slog.Int( key: "user_id", userID),
        slog.String( key: "error", err.Error()),
    )

    return fmt.Errorf( format: "user %d: can not create order: %w", userID, err)
}
```

Если просто вернуть ошибку, она будет plain text

```go
err := u.createOrder(ctx, userID)
if err != nil {
    return fmt.Errorf( format: "user %d: can not create order: %w", userID, err)
}
```

Если просто вернуть ошибку, она будет plain text

```
err = u.Do()
if err != nil {
    slog.Error( msg: "operation error", slog.String( key: "error", err.Error()))
}
```

Стоит добавить к ошибке детали

```go
type ErrWithDetails struct {
    Err     error
    Details []any
}

func (e *ErrWithDetails) Error() string { return e.Err.Error() }

func WithDetails(err error, details ...any) error {
    return &ErrWithDetails{
        Err:     err,
        Details: details,
    }
}
```

Стоит добавить к ошибке детали

```
type ErrWithDetails struct {
    Err     error
    Details
}

func (e *Er

func WithDe
    return
        Err
        Det
    }
}
```

```
err := u.createOrder(ctx, userID)
if err != nil {
    return WithDetails(
        fmt.Errorf( format: "can not create order: %w", err),
        slog.Int( key: "user_id", userID),
    )
}
```

Потом эти детали можно логировать

```go
func AttrsFromErr(err error) []any {
    var errDetails *ErrWithDetails
    if !errors.As(err, &errDetails) {
        return []any{slog.String( key: "error", err.Error())}
    }

    attrs := make([]any, 0, len(errDetails.Details)+1)
    attrs = append(attrs, slog.String( key: "error", errDetails.Error()))
    attrs = append(attrs, errDetails.Details...)

    return attrs
}
```

Потом эти детали можно логировать

```go
func AttrsFromErr(err error) []any {
    var errDetails *ErrWithDetails
    if !errors.As(err, &errDetails) {
        return []any{slog.String( key: "error", e
    }

    attrs := make([]any, 0, len(errDetails.Details)+1)
    attrs = append(attrs, slog.String( key: "error", errDetails.Error()))
    attrs = append(attrs, errDetails.Details...)

    return attrs
}
```

```go
err := u.Do()
if err ≠ nil {
    slog.Error( msg: "operation error", AttrsFromErr(err)...)
}
```

Потом эти детали можно логировать

```go
func AttrsFromErr(err error) []any {
    var errDetails *ErrWithDetails
    if !errors.As(err, &errDetails) {
        return []any{slog.String( key: "error", e
    }

    attrs := make(
    attrs = append
    attrs = append

    return attrs
}
```

```go
err := u.Do()
if err != nil {
    slog.Error( msg: "operation error", AttrsFromErr(err)...)
}
```

```
{
  "time": "2024-09-04T16:25:13.191877+03:00",
  "level": "ERROR",
  "msg": "operation error",
  "error": "can not create order: can not create order: inner problem",
  "user_id": 1234
}
```

Иногда логировать ошибки – недостаточно

```
slog.Info( msg: "init request")

// some logic

slog.Info( msg: "start operation")
err := u.Do()
if err != nil {
    slog.Error( msg: "operation error", AttrsFromErr(err)...)
}

slog.Info( msg: "end operation")
```

Иногда логировать ошибки – недостаточно

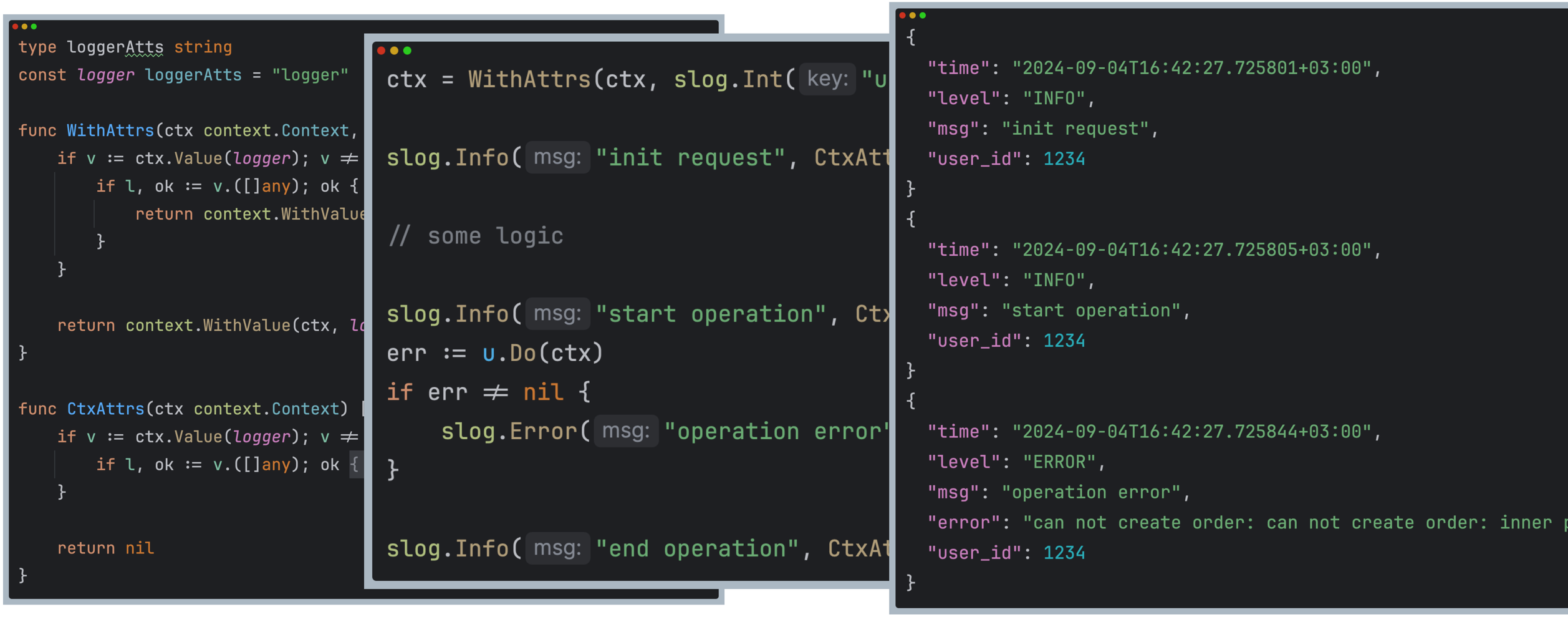
```go
type loggerAtts string
const logger loggerAtts = "logger"

func WithAttrs(ctx context.Context, fields ...any) context.Context  {
    if v := ctx.Value(logger); v != nil {
        if l, ok := v.([]any); ok {
            return context.WithValue(ctx, logger, append(l, fields...))
        }
    }

    return context.WithValue(ctx, logger,fields)
}

func CtxAttrs(ctx context.Context) []any {
    if v := ctx.Value(logger); v != nil {
        if l, ok := v.([]any); ok { return l }
    }

    return nil
}
```

Иногда логировать ошибки – недостаточно

```
type loggerAtts string
const logger loggerAtts = "logger"

func WithAttrs(ctx context.Context,
    if v := ctx.Value(logger); v ≠
        if l, ok := v.([]any); ok {
            return context.WithValue
        }
    }

    return context.WithValue(ctx, lo
}

func CtxAttrs(ctx context.Context) [
    if v := ctx.Value(logger); v ≠
        if l, ok := v.([]any); ok {
    }

    return nil
}
```

```
ctx = WithAttrs(ctx, slog.Int( key: "user_id", userID))

slog.Info( msg: "init request", CtxAttrs(ctx)...)

// some logic

slog.Info( msg: "start operation", CtxAttrs(ctx)...)
err := u.Do(ctx)
if err ≠ nil {
    slog.Error( msg: "operation error", AttrsFromErr(err)...)
}

slog.Info( msg: "end operation", CtxAttrs(ctx)...)
```

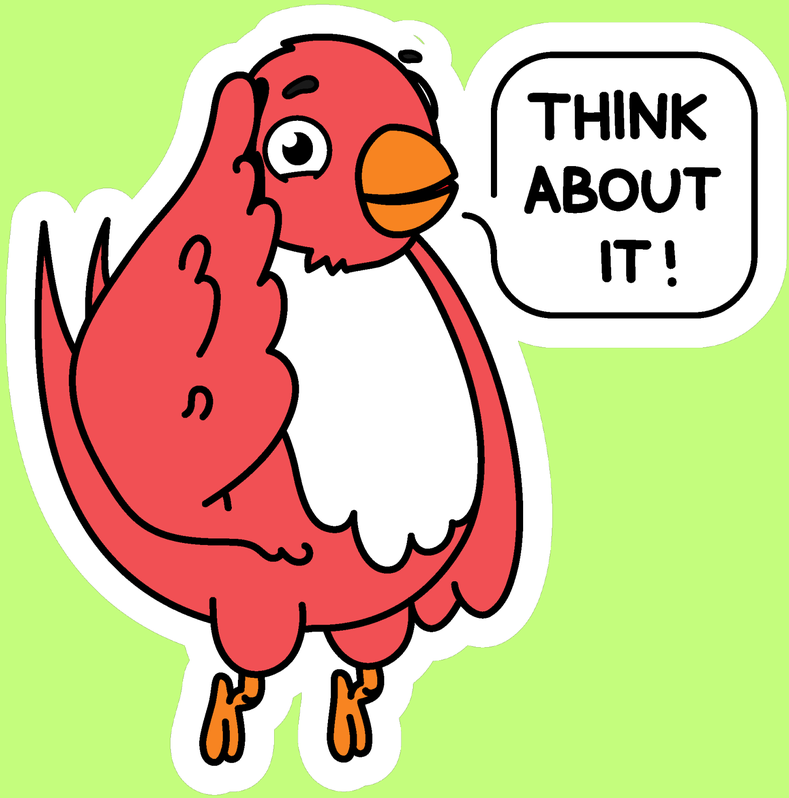
Иногда логировать ошибки – недостаточно



Итого

- Логируем ошибки с деталями
- Стремимся передавать контекст логирования ниже по стеку
- Используем разные уровни логирования

Не забываем про метрики



Зачем нужны метрики

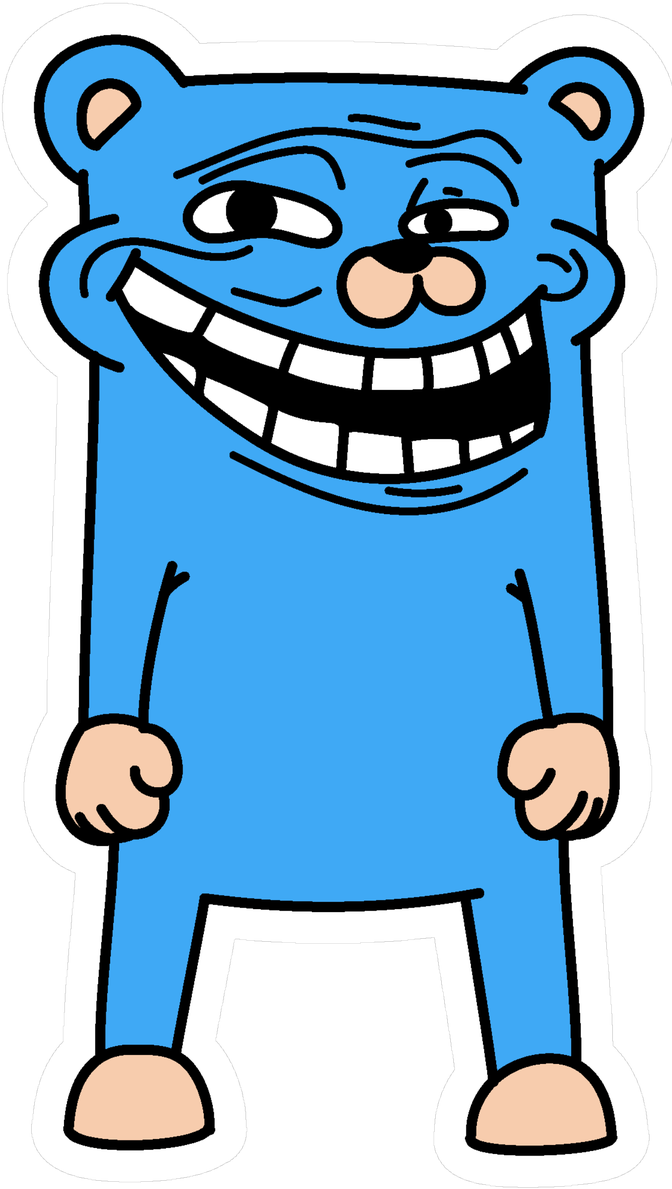
- Контроль работы сервиса на протяжении времени
- Алерты на их основе

Как выглядят метрики

- request_latency: 10
- orders_created: 5267

Библиотеки

- В Go нет библиотек для метрик в stdlib



Библиотеки

- В Go нет библиотек для метрик в stdlib
- Но есть сторонние:
  - StatsD
  - Prometheus

Рассмотрим пример с prometheus

```go
func main() {
    go func() {
        http.Handle("/metrics", promhttp.Handler())
        http.ListenAndServe( addr: ":2112",  handler: nil)
    }()

    application()
}
```

Рассмотрим пример с prometheus

```go
func main() {
    go func() {
        http.Handle(
        http.ListenA
    }()

    application()
}
```

```go
var (  1 usage
    operationError = promauto.NewCounter(prometheus.CounterOpts{
        Name: "operation_error",
        Help: "Total number of operations error",
    })
)
```

Рассмотрим пример с prometheus

```go
func main() {
    go func() {
        http.Handle(
        http.List
    }()

    application()
}
```

```go
var (  1 usage
    operationError = promauto.NewCounter(prometheus.CounterOpts{
        Name: "operation_error",
        Help: "Total number of operations error",
    })
```

```go
err := u.Do(ctx)
if err != nil {
    operationError.Inc()
    slog.Error( msg: "operation error", AttrsFromErr(err)...)
}
```

Рассмотрим пример с prometheus

У метрик тоже может быть контекст

```
var (  2 usages
    operationError = promauto.NewCounterVec(prometheus.CounterOpts{
        Name: "operation_error",
        Help: "Total number of operations error",
    }, []string{
        "user_id",
    })
)
```

У метрик тоже может быть контекст

```
var (  2 usages
    operationError = promauto.NewCounterVec(prometheus.CounterOpts{
        Name: "operation_error",
        Help: "Total number of operations error",
    }, []string{
        "user_id",
    })
)
```

```
err := u.Do(ctx)
if err != nil {
    operationError.With(prometheus.Labels{"user_id": fmt.Sprint(userID)}).Inc()
    slog.Error( msg: "operation error", AttrsFromErr(err)...)
}
```

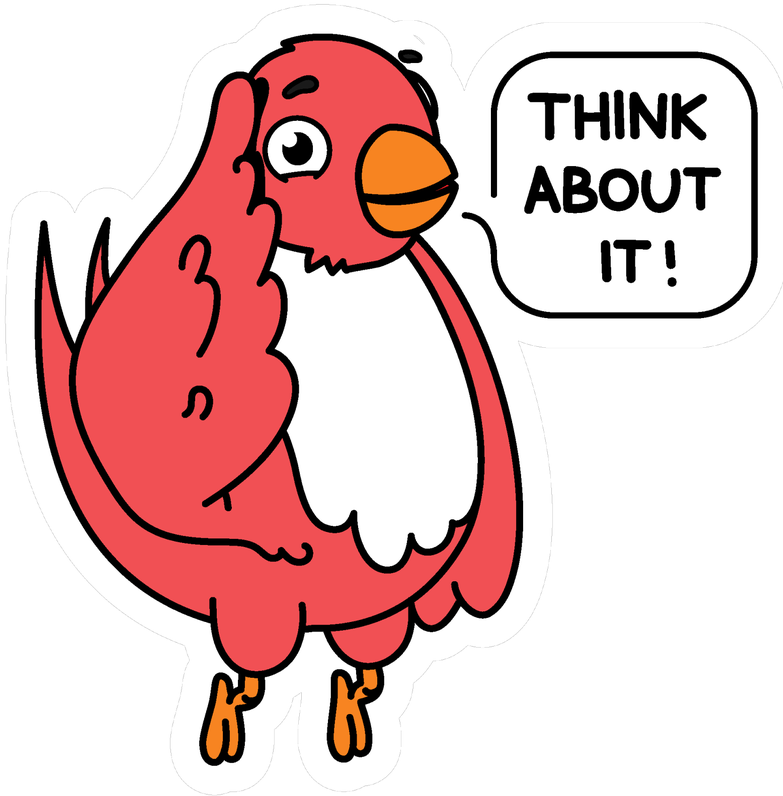
Но следует избегать большой кардинальности метрик

- Кардинальность – количество комбинаций параметров

```
err := u.Do(ctx)
if err != nil {
    operationError.With(prometheus.Labels{"user_id": fmt.Sprint(userID)}).Inc()
    slog.Error( msg: "operation error", AttrsFromErr(err)...)
}
```

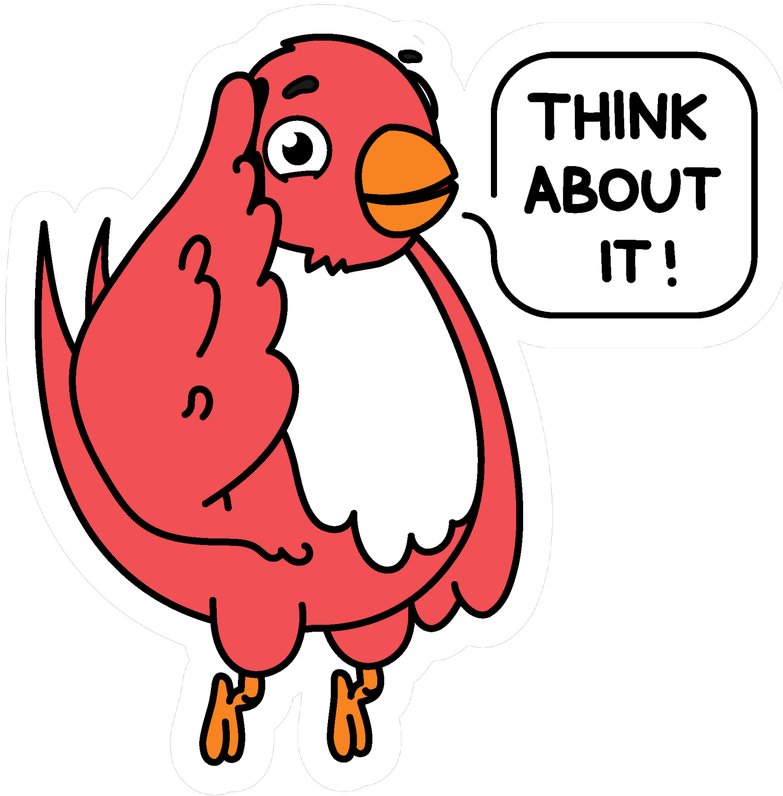
Но следует избегать большой кардинальности метрик

- Кардинальность – количество комбинаций параметров
- Большое количество разных параметров порождает много отдельных серий в системах метрик



Но следует избегать большой кардинальности метрик

- Кардинальность – количество комбинаций параметров
- Большое количество разных параметров порождает много отдельных серий в системах метрик
- В метки стоит выносить значения, у которых мало вариаций (например, окружение сервиса, версию приложения)

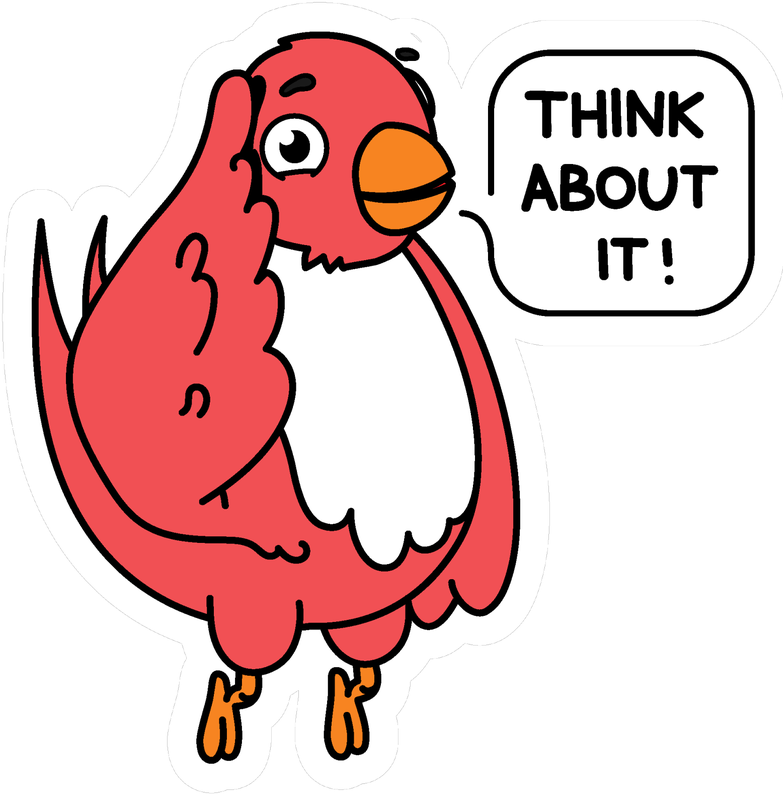


Что покрывать метриками

- Количество ошибок
- Тайминги операций
- Счетчики сущностей

Метрики ничто без визуализации

- Доски в Grafana
- Алерты на основе метрик



Профилируем приложение

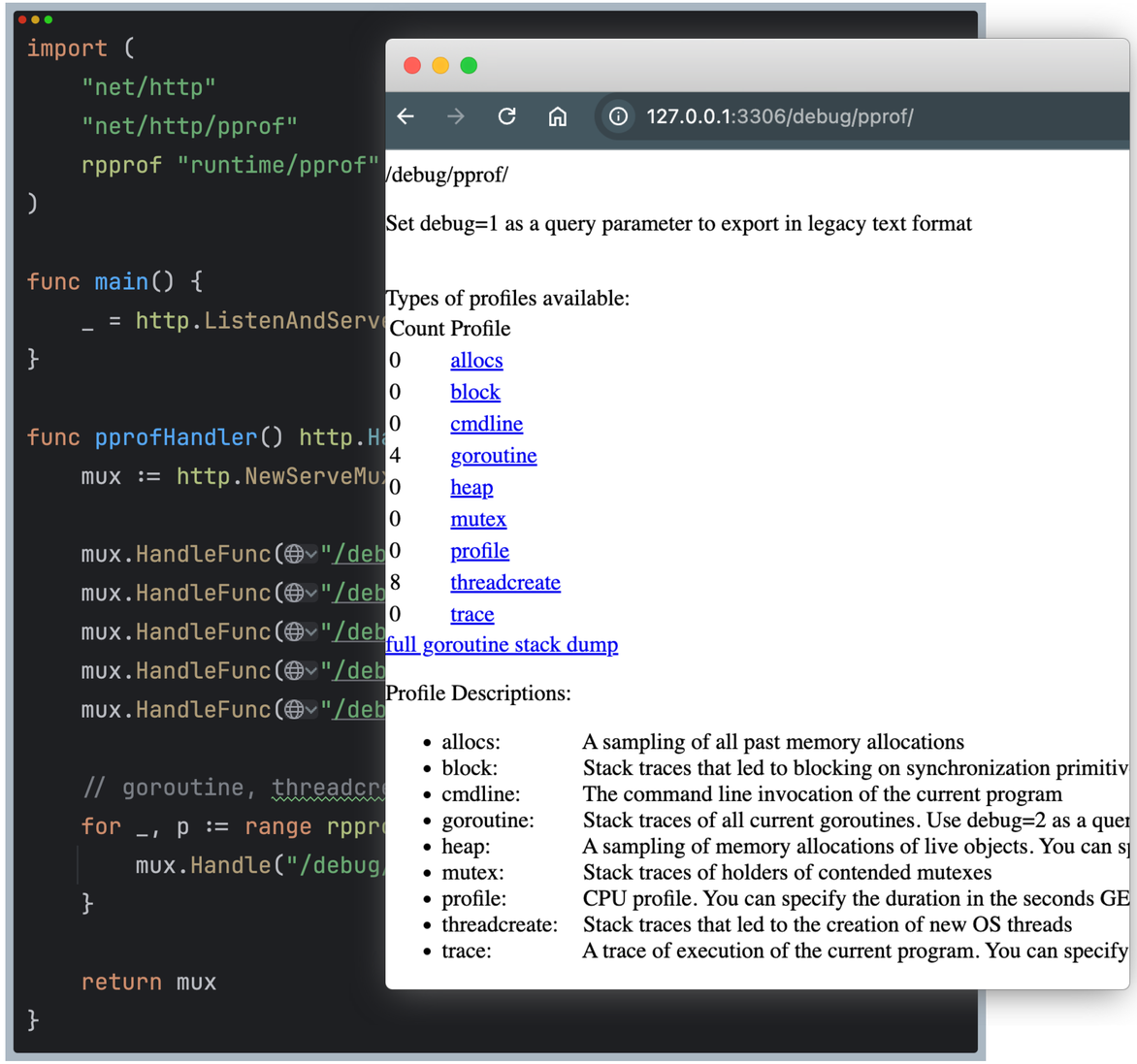
pprof

- В Go есть стандартный профилировщик
- Который надо включить отдельно!

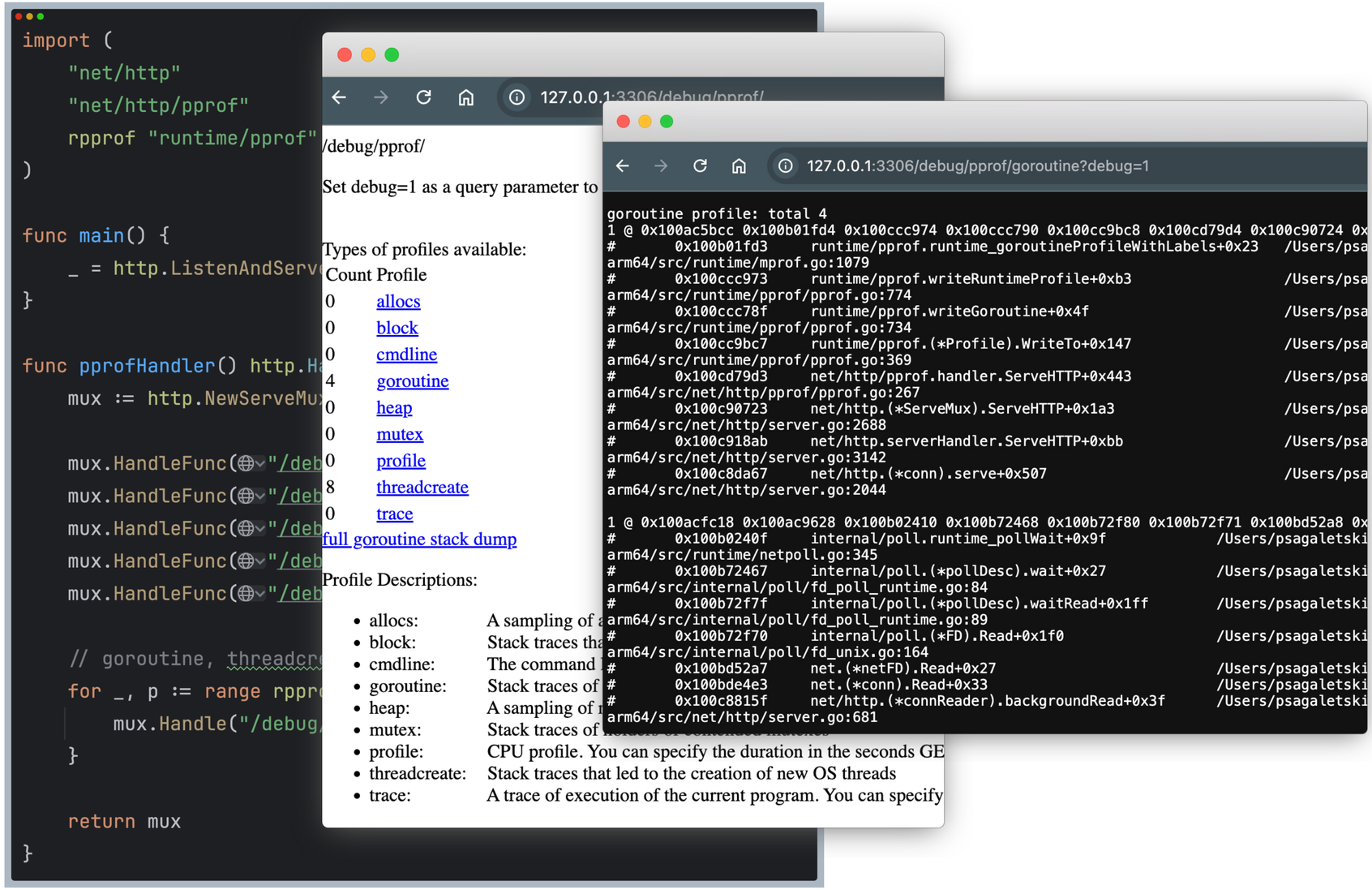
pprof

```go
import (
    "net/http"
    "net/http/pprof"
    rpprof "runtime/pprof"
)

func main() {
    _ = http.ListenAndServe( addr: ":3306", pprofHandler())
}

func pprofHandler() http.Handler {  1 usage
    mux := http.NewServeMux()

    mux.HandleFunc("/debug/pprof/", pprof.Index)
    mux.HandleFunc("/debug/pprof/cmdline", pprof.Cmdline)
    mux.HandleFunc("/debug/pprof/profile", pprof.Profile)
    mux.HandleFunc("/debug/pprof/symbol", pprof.Symbol)
    mux.HandleFunc("/debug/pprof/trace", pprof.Trace)

    // goroutine, threadcreate, heap, allocs, block, mutex
    for _, p := range rpprof.Profiles() {
        mux.Handle("/debug/pprof/"+p.Name(), pprof.Handler(p.Name()))
    }

    return mux
}
```

pprof


import (
"net/http"
"net/http/pprof"
rpprof "runtime/pprof"
)
func main() {
_ = http.ListenAndServ
}
func pprofHandler() http.H
mux := http.NewServeMu
mux.HandleFunc("/deb
mux.HandleFunc("/deb
mux.HandleFunc("/deb
mux.HandleFunc("/deb
mux.HandleFunc("/deb
// goroutine, threadcr
for _, p := range rppr
mux.Handle("/debug
}
return mux
}
127.0.0.1:3306/debug/pprof/
/debug/pprof/
Set debug=1 as a query parameter to export in legacy text format
Types of profiles available:
Count Profile
0 allocs
0 block
0 cmdline
4 goroutine
0 heap
0 mutex
0 profile
8 threadcreate
0 trace
full goroutine stack dump
Profile Descriptions:
allocs: A sampling of all past memory allocations
block: Stack traces that led to blocking on synchronization primitiv
cmdline: The command line invocation of the current program
goroutine: Stack traces of all current goroutines. Use debug=2 as a quer
heap: A sampling of memory allocations of live objects. You can s
mutex: Stack traces of holders of contended mutexes
profile: CPU profile. You can specify the duration in the seconds GE
threadcreate: Stack traces that led to the creation of new OS threads
trace: A trace of execution of the current program. You can specify

pprof


import (
"net/http"
"net/http/pprof"
rpprof "runtime/pprof"
)
func main() {
_ = http.ListenAndServ
}
func pprofHandler() http.H
mux := http.NewServeMu
mux.HandleFunc("/deb
mux.HandleFunc("/deb
mux.HandleFunc("/deb
mux.HandleFunc("/deb
mux.HandleFunc("/deb
// goroutine, threadcr
for _, p := range rppr
mux.Handle("/debug
}
return mux
}
127.0.0.1:3306/debug/pprof/
/debug/pprof/
Set debug=1 as a query parameter to
Types of profiles available:
Count Profile
0 allocs
0 block
0 cmdline
4 goroutine
0 heap
0 mutex
0 profile
8 threadcreate
0 trace
full goroutine stack dump
Profile Descriptions:
allocs: A sampling of
block: Stack traces tha
cmdline: The command
goroutine: Stack traces of
heap: A sampling of
mutex: Stack traces of
profile: CPU profile. You can specify the duration in the seconds GE
threadcreate: Stack traces that led to the creation of new OS threads
trace: A trace of execution of the current program. You can specify
127.0.0.1:3306/debug/pprof/goroutine?debug=1
goroutine profile: total 4
1 @ 0x100ac5bcc 0x100b01fd4 0x100ccc974 0x100ccc790 0x100cc9bc8 0x100cd79d4 0x100c90724 0x
0x100b01fd3 runtime/pprof.runtime_goroutineProfileWithLabels+0x23 /Users/psa
arm64/src/runtime/mprof.go:1079
0x100ccc973 runtime/pprof.writeRuntimeProfile+0xb3 /Users/psa
arm64/src/runtime/pprof/pprof.go:774
0x100ccc78f runtime/pprof.writeGoroutine+0x4f /Users/psa
arm64/src/runtime/pprof/pprof.go:734
0x100cc9bc7 runtime/pprof.(*Profile).WriteTo+0x147 /Users/psa
arm64/src/runtime/pprof/pprof.go:369
0x100cd79d3 net/http/pprof.handler.ServeHTTP+0x443 /Users/psa
arm64/src/net/http/pprof/pprof.go:267
0x100c90723 net/http.(*ServeMux).ServeHTTP+0x1a3 /Users/psa
arm64/src/net/http/server.go:2688
0x100c918ab net/http.serverHandler.ServeHTTP+0xbb /Users/psa
arm64/src/net/http/server.go:3142
0x100c8da67 net/http.(*conn).serve+0x507 /Users/psa
arm64/src/net/http/server.go:2044
1 @ 0x100acfc18 0x100ac9628 0x100b02410 0x100b72468 0x100b72f80 0x100b72f71 0x100bd52a8 0x
0x100b0240f internal/poll.runtime_pollWait+0x9f /Users/psagaletski
arm64/src/runtime/netpoll.go:345
0x100b72467 internal/poll.(*pollDesc).wait+0x27 /Users/psagaletski
arm64/src/internal/poll/fd_poll_runtime.go:84
0x100b72f7f internal/poll.(*pollDesc).waitRead+0x1ff /Users/psagaletski
arm64/src/internal/poll/fd_poll_runtime.go:89
0x100b72f70 internal/poll.(*FD).Read+0x1f0 /Users/psagaletski
arm64/src/internal/poll/fd_unix.go:164
0x100bd52a7 net.(*netFD).Read+0x27 /Users/psagaletski
0x100bde4e3 net.(*conn).Read+0x33 /Users/psagaletski
0x100c8815f net/http.(*connReader).backgroundRead+0x3f /Users/psagaletski
arm64/src/net/http/server.go:681

pprof

```
import (
    "net/http"
    "net/http/pprof"
    rpprof "runtime/pprof"
)

func main() {
    _ = http.ListenAndServe( addr: ":3306", pprofHandler())
}
```

```
go tool pprof -http=:8183 \
  "http://127.0.0.1:3306/debug/pprof/profile?seconds=30"
```

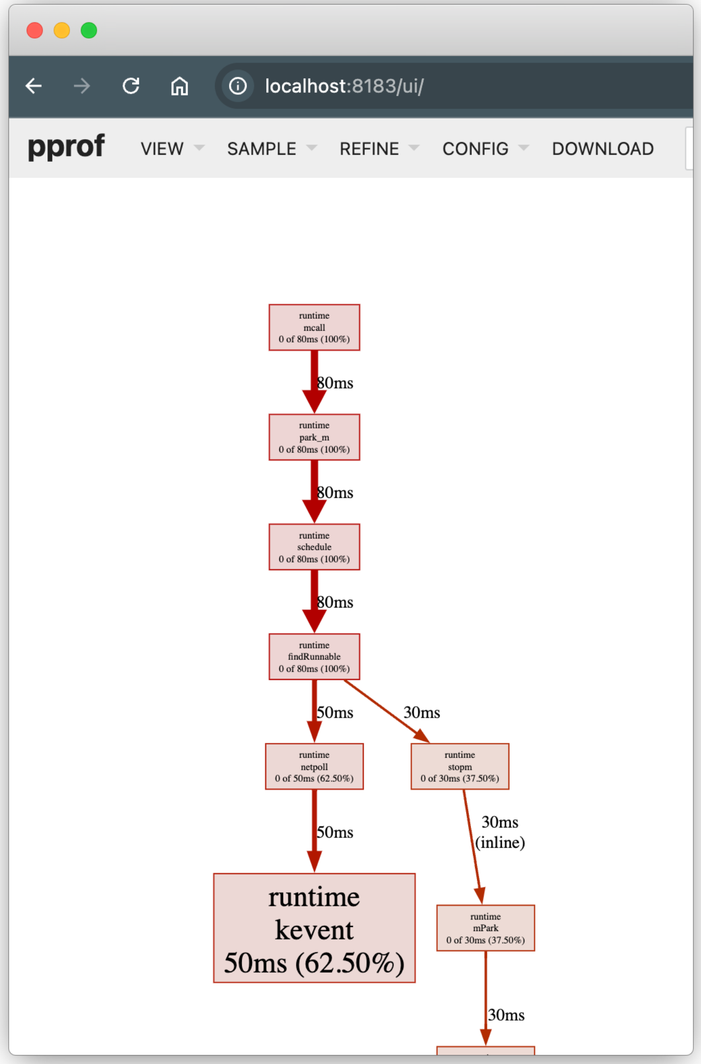
```
    mux.HandleFunc("/debug/pprof/profile", pprof.Profile)
    mux.HandleFunc("/debug/pprof/symbol", pprof.Symbol)
    mux.HandleFunc("/debug/pprof/trace", pprof.Trace)

    // goroutine, threadcreate, heap, allocs, block, mutex
    for _, p := range rpprof.Profiles() {
        mux.Handle("/debug/pprof/"+p.Name(), pprof.Handler(p.Name()))
    }

    return mux
}
```

pprof

```go
import (
    "net/http"
    "net/http/pprof"
    rpprof "runtime/pprof"
)

func main() {
    _ = http.ListenAndServe( addr: ":3306", pprofHandler())
}
```

```
go tool pprof -http=:8183 \
  "http://127.0.0.1:3306/debug/pprof/profile?seconds=30"
```

```go
    mux.HandleFunc("/debug/pprof/profile", pprof.Profile)
    mux.HandleFunc("/debug/pprof/symbol", pprof.Symbol)
    mux.HandleFunc("/debug/pprof/trace", pprof.Trace)

    // goroutine, threadcreate, heap, allocs, block, mutex
    for _, p := range rpprof.Profiles() {
        mux.Handle("/debug/pprof/"+p.Name(), pprof.Handler(p.Name()))
    }

    return mux
}
```



В чем польза?

- Можно увидеть какие горутины в системе и что они делают
- Посмотреть flamegraph нашего приложения и понять на что тратится время
- Проанализировать причины замедления отдельных частей программы

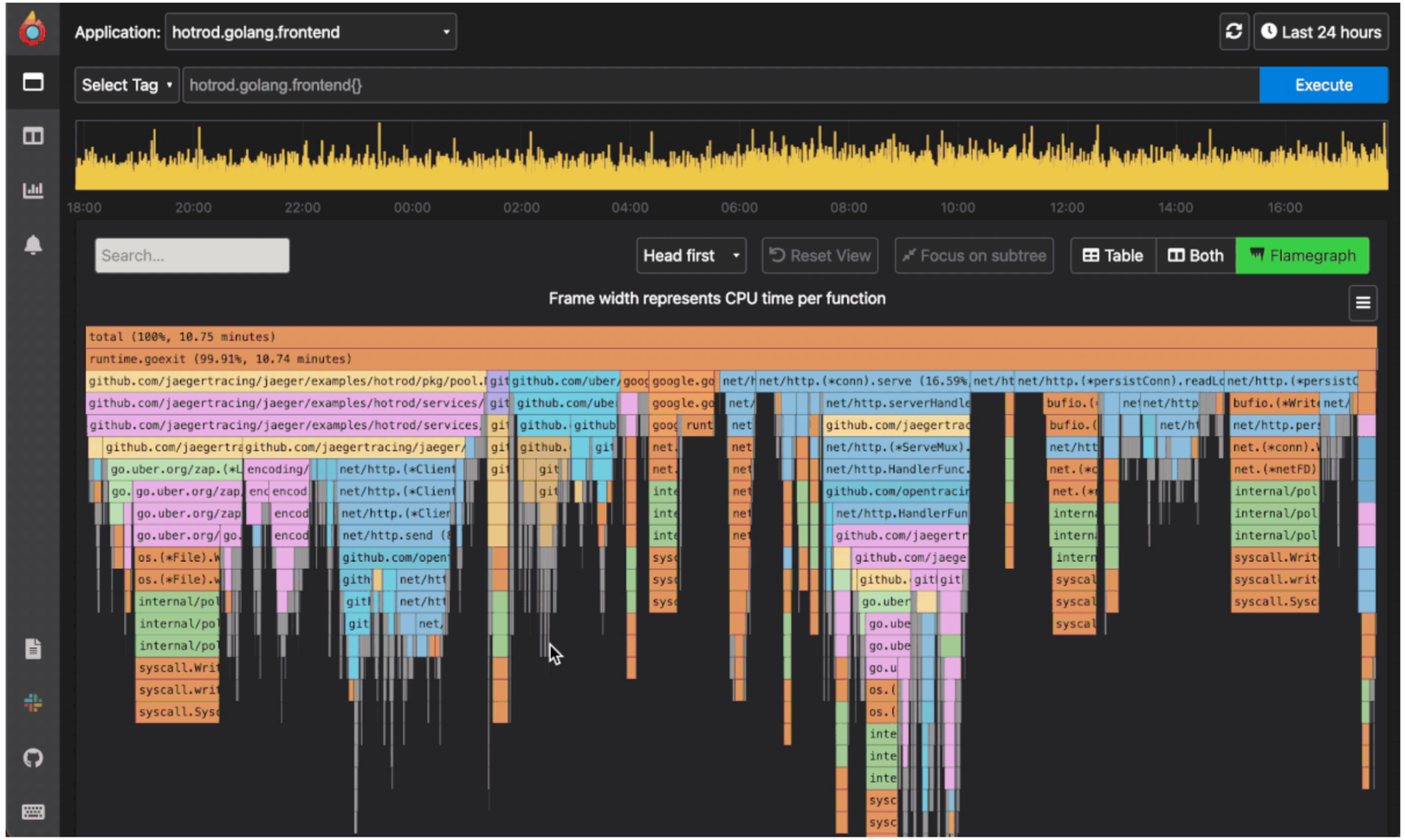
В чем польза?

- Можно увидеть какие горутины в системе и что они делают
- Посмотреть flamegraph нашего приложения и понять на что тратится время
- Проанализировать причины замедления отдельных частей программы

Continuous profile

- Когда профайл снимается постоянно
- Можно видеть его исторические данные

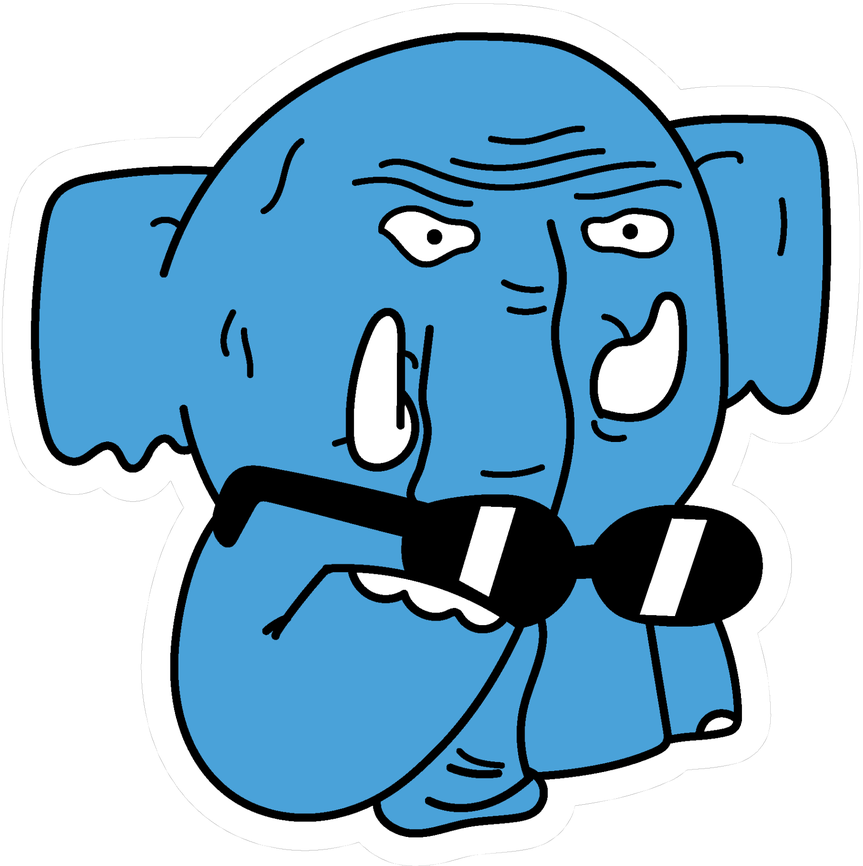
Continuous profile (например, pyroscope)



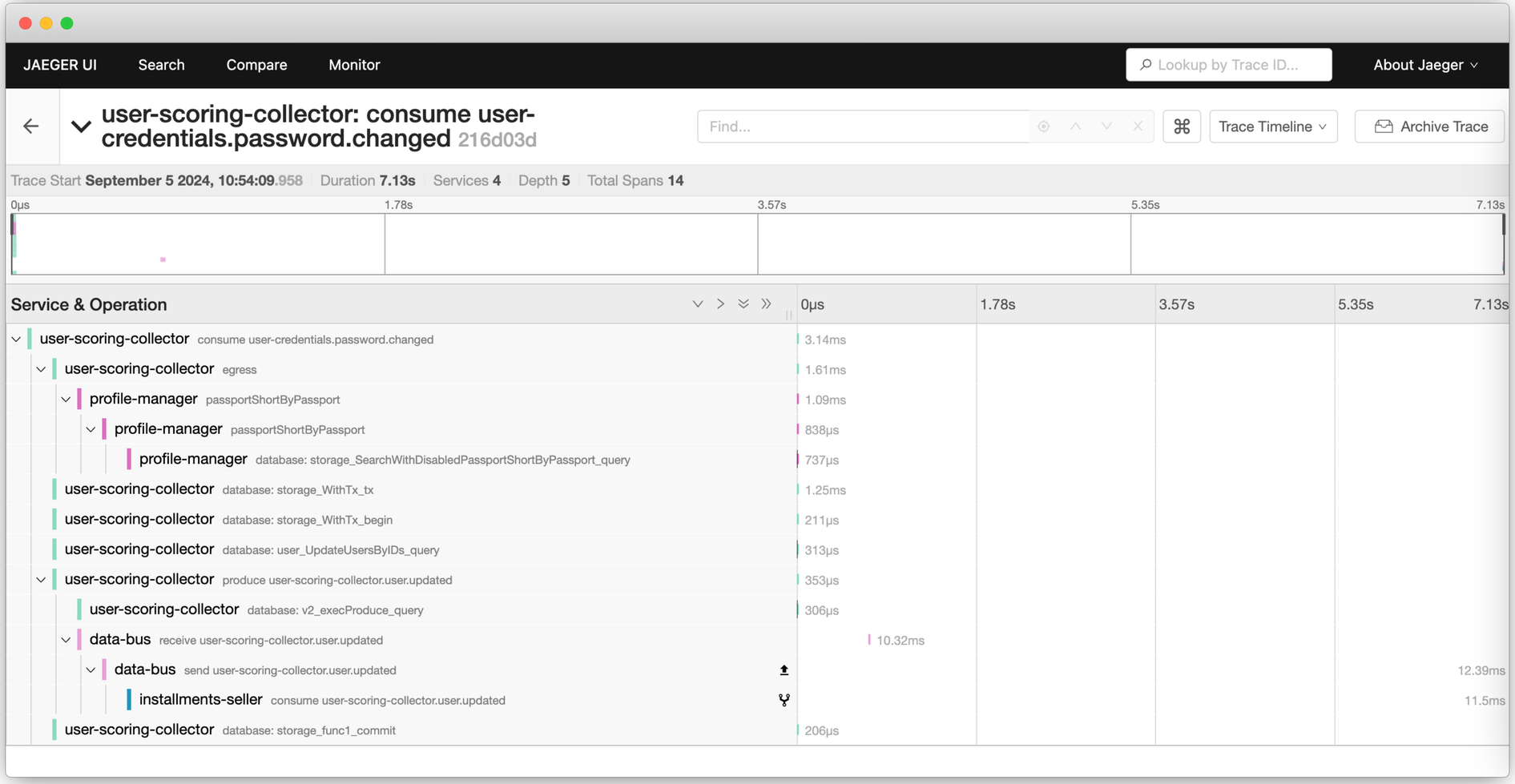
Попробуем трейсинг

Иногда не ясно, в каком приложении проблема

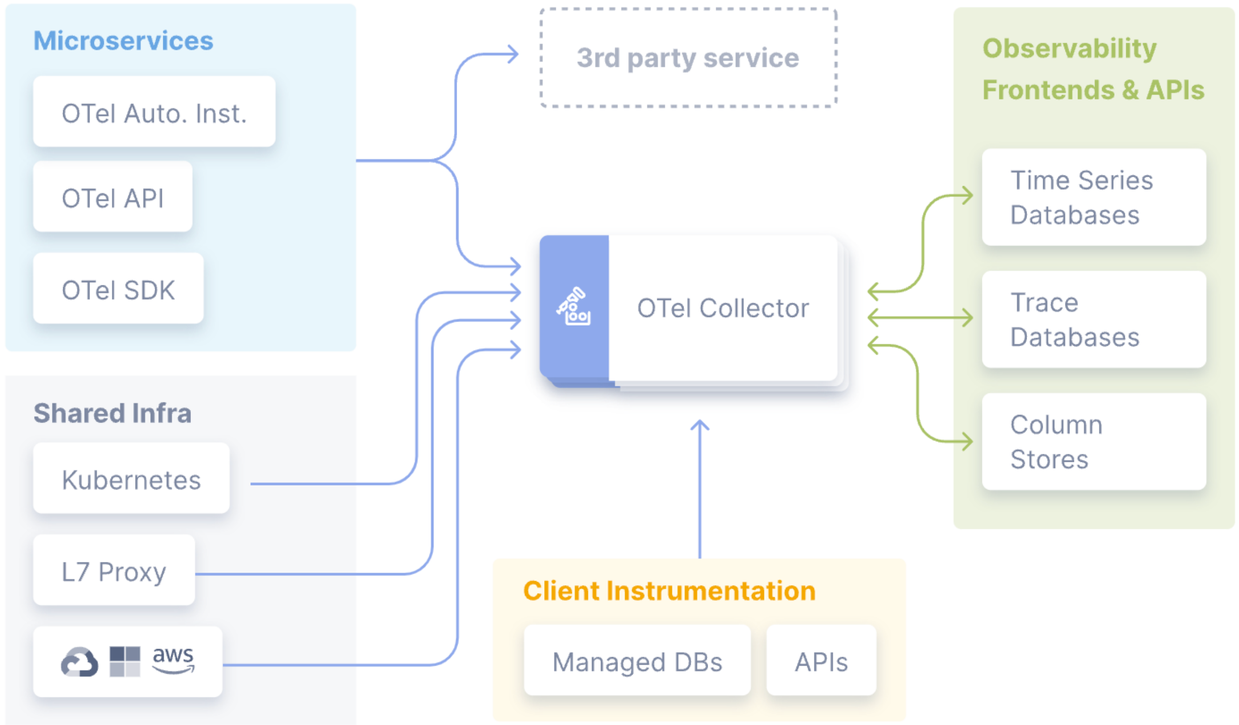
- Если сервис один, то это может быть и не надо
- Но когда сервисов несколько, бывает не ясно, где проблема



Трейсинг


JAEGER UI
Search
Compare
Monitor
Lookup by Trace ID...
About Jaeger
user-scoring-collector: consume user-credentials.password.changed 216d03d
Find...
Trace Timeline
Archive Trace
Trace Start September 5 2024, 10:54:09.958 Duration 7.13s Services 4 Depth 5 Total Spans 14
0µs
1.78s
3.57s
5.35s
7.13s
Service & Operation
user-scoring-collector consume user-credentials.password.changed 3.14ms
user-scoring-collector egress 1.61ms
profile-manager passportShortByPassport 1.09ms
profile-manager passportShortByPassport 838µs
profile-manager database: storage_SearchWithDisabledPassportShortByPassport_query 737µs
user-scoring-collector database: storage_WithTx_tx 1.25ms
user-scoring-collector database: storage_WithTx_begin 211µs
user-scoring-collector database: user_UpdateUsersByIDs_query 313µs
user-scoring-collector produce user-scoring-collector.user.updated 353µs
user-scoring-collector database: v2_execProduce_query 306µs
data-bus receive user-scoring-collector.user.updated 10.32ms
data-bus send user-scoring-collector.user.updated 12.39ms
installments-seller consume user-scoring-collector.user.updated 11.5ms
user-scoring-collector database: storage_func1_commit 206µs

Для его работы нужна инфраструктура



Один из вариантов – opentelemetry

Инициализация немного сложная...

```
    "go.opentelemetry.io/otel/exporters/stdout/stdoutlog"
    "go.opentelemetry.io/otel/exporters/stdout/stdoutmetric"
    "go.opentelemetry.io/otel/exporters/stdout/stdouttrace"
    "go.opentelemetry.io/otel/log/global"
    "go.opentelemetry.io/otel/propagation"
    "go.opentelemetry.io/otel/sdk/log"
    "go.opentelemetry.io/otel/sdk/metric"
    "go.opentelemetry.io/otel/sdk/trace"
)

// setupOTelSDK bootstraps the OpenTelemetry pipeline.
// If it does not return an error, make sure to call shutdown for proper cleanup.
func setupOTelSDK(ctx context.Context) (shutdown func(context.Context) error, err error) {
    var shutdownFuncs []func(context.Context) error

    // shutdown calls cleanup functions registered via shutdownFuncs.
    // The errors from the calls are joined.
    // Each registered cleanup will be invoked once.
    shutdown = func(ctx context.Context) error {
        var err error
        for _, fn := range shutdownFuncs {
            err = errors.Join(err, fn(ctx))
        }
        shutdownFuncs = nil
        return err
    }

    // handleErr calls shutdown for cleanup and makes sure that all errors are returned.
    handleErr := func(inErr error) {
        err = errors.Join(inErr, shutdown(ctx))
    }

    // Set up propagator.
    prop := newPropagator()
    otel.SetTextMapPropagator(prop)

    // Set up trace provider.
    tracerProvider, err := newTraceProvider()
    if err != nil {
        handleErr(err)
        return
    }
    shutdownFuncs = append(shutdownFuncs, tracerProvider.Shutdown)
    otel.SetTracerProvider(tracerProvider)

    // Set up meter provider.
    meterProvider, err := newMeterProvider()
    if err != nil {
        handleErr(err)
        return
    }
    shutdownFuncs = append(shutdownFuncs, meterProvider.Shutdown)
```

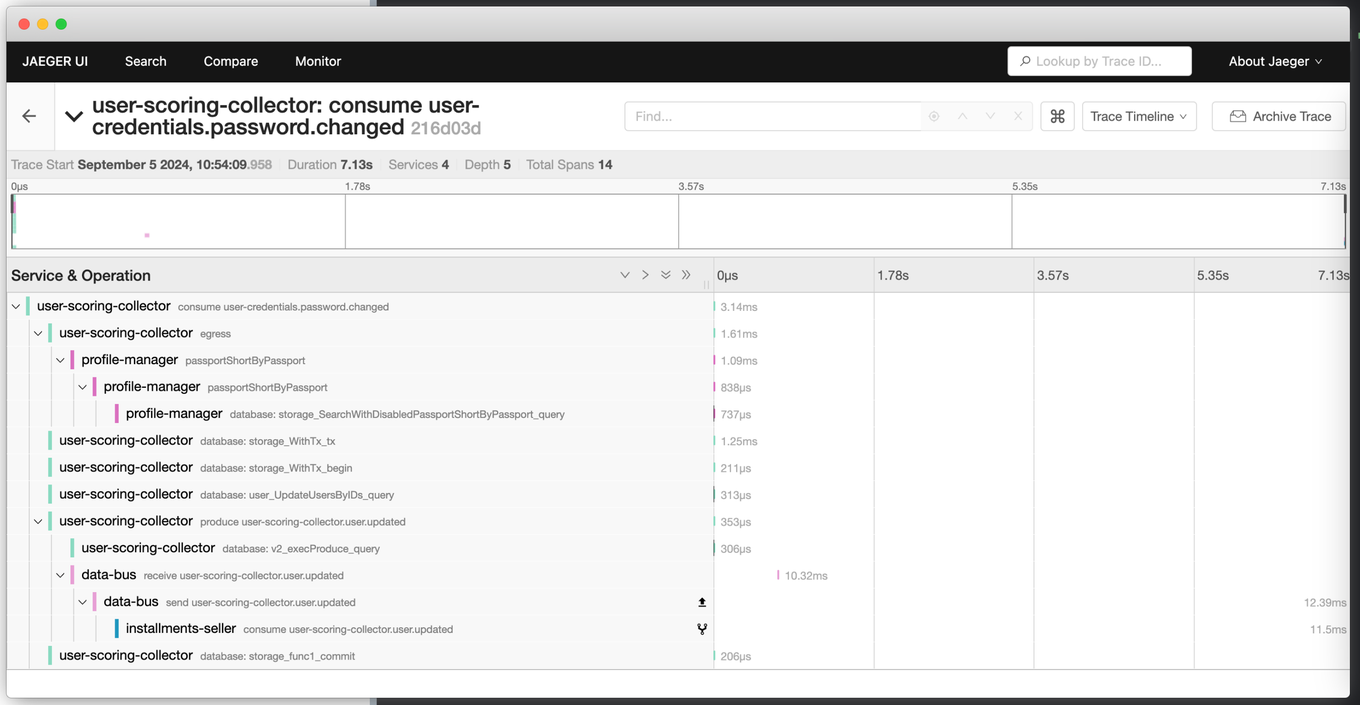
И видеть, что происходит между сервисами и внутри них

```go
func main() {
    // initialize OpenTelemetry SDK ...

    http.Handle("/", otelhttp.NewHandler(http.HandlerFunc(httpHandler), operation: "/"))
}

const name = "myapp"  1 usage

var (  2 usages
    tracer = otel.Tracer(name)  2 usages
)

func httpHandler(w http.ResponseWriter, r *http.Request) {  1 usage
    ctx, span := tracer.Start(r.Context(), spanName: "httpHandler")
    defer span.End()

    // do something with context
    otherMethod(ctx)
}

func otherMethod(ctx context.Context) {  1 usage
    _, span := tracer.Start(ctx, spanName: "otherMethod")
    defer span.End()

    // some logic
}
```

```go
    "go.opentelemetry.io/otel/exporters/stdout/stdoutlog"
    "go.opentelemetry.io/otel/exporters/stdout/stdoutmetric"
    "go.opentelemetry.io/otel/exporters/stdout/stdouttrace"
    "go.opentelemetry.io/otel/log/global"
    "go.opentelemetry.io/otel/propagation"
    "go.opentelemetry.io/otel/sdk/log"
    "go.opentelemetry.io/otel/sdk/metric"
    "go.opentelemetry.io/otel/sdk/trace"
)

// setupOTelSDK bootstraps the OpenTelemetry pipeline.
// If it does not return an error, make sure to call shutdown for proper cleanup.
func setupOTelSDK(ctx context.Context) (shutdown func(context.Context) error, err error) {
    var shutdownFuncs []func(context.Context) error

    // shutdown calls cleanup functions registered via shutdownFuncs.
    // The errors from the calls are joined.
    // Each registered cleanup will be invoked once.
    shutdown = func(ctx context.Context) error {
        var err error
        for _, fn := range shutdownFuncs {
            err = errors.Join(err, fn(ctx))
        }
        shutdownFuncs = nil
        return err
    }

    // handleErr calls shutdown for cleanup and makes sure that all errors are returned.
    handleErr := func(inErr error) {
        err = errors.Join(inErr, shutdown(ctx))
    }

    // Set up propagator.
    prop := newPropagator()
    otel.SetTextMapPropagator(prop)

    // Set up trace provider.
    tracerProvider, err := newTraceProvider()
    if err != nil {
        handleErr(err)
        return
    }
    shutdownFuncs = append(shutdownFuncs, tracerProvider.Shutdown)
    otel.SetTracerProvider(tracerProvider)

    // Set up meter provider.
    meterProvider, err := newMeterProvider()
    if err != nil {
        handleErr(err)
        return
    }
    shutdownFuncs = append(shutdownFuncs, meterProvider.Shutdown)
```

И видеть, что происходит между сервисами и внутри них

```go
func main() {
    // initialize OpenTelemetry SDK ...
    "/"))
    // some logic
}
```

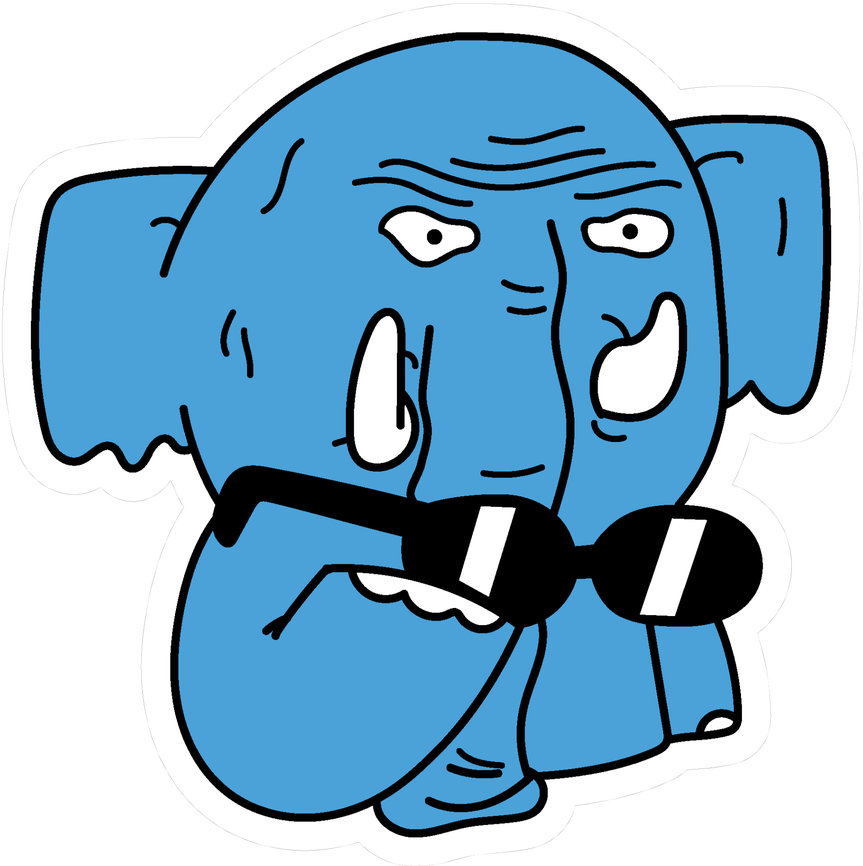



```go
    "go.opentelemetry.io/otel/exporters/stdout/stdoutlog"
    "go.opentelemetry.io/otel/exporters/stdout/stdoutmetric"
    "go.opentelemetry.io/otel/exporters/stdout/stdouttrace"
    "go.opentelemetry.io/otel/log/global"
    "go.opentelemetry.io/otel/propagation"
    "go.opentelemetry.io/otel/sdk/log"
    "go.opentelemetry.io/otel/sdk/metric"
    "go.opentelemetry.io/otel/sdk/trace"
)

// setupOTelSDK bootstraps the OpenTelemetry pipeline.
// If it does not return an error, make sure to call shutdown for proper cleanup.
func setupOTelSDK(ctx context.Context) (shutdown func(context.Context) error, err error) {
    var shutdownFuncs []func(context.Context) error

    // shutdown calls cleanup functions registered via shutdownFuncs.
    // The errors from the calls are joined.
    // Each registered cleanup will be invoked once.
    shutdown = func(ctx context.Context) error {
        var err error
        for _, fn := range shutdownFuncs {
            err = errors.Join(err, fn(ctx))
        }
        shutdownFuncs = nil
        return err
    }

    // handleErr calls shutdown for cleanup and makes sure that all errors are returned.
    handleErr := func(inErr error) {
        err = errors.Join(inErr, shutdown(ctx))
    }

    // Set up propagator.
    prop := newPropagator()
    otel.SetTextMapPropagator(prop)

    // Set up trace provider.
    tracerProvider, err := newTraceProvider()
    if err != nil {
        handleErr(err)
        return
    }
    shutdownFuncs = append(shutdownFuncs, tracerProvider.Shutdown)
    otel.SetTracerProvider(tracerProvider)

    // Set up meter provider.
    meterProvider, err := newMeterProvider()
    if err != nil {
        handleErr(err)
        return
    }
    shutdownFuncs = append(shutdownFuncs, meterProvider.Shutdown)
```

Подключаемся отладчиком

Disclaimer: это опасно

- Можно сломать приложение в prod
- Нужны доступы
- Но иногда помогает



Первым делом, надо поставить отладчик dlv

```
go install github.com/go-delve/delve/cmd/dlv@latest
```

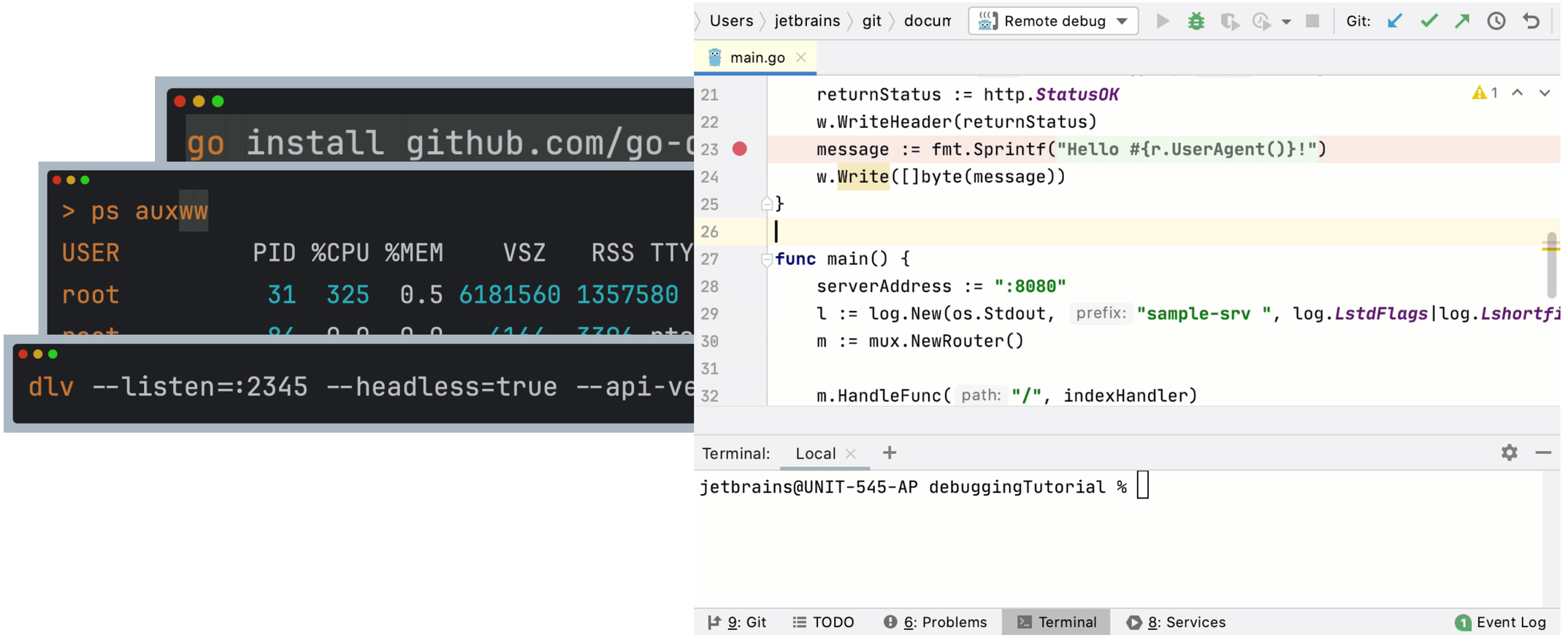
Первым делом, надо поставить отладчик dlv

```
go install github.com/go-delve/delve/cmd/dlv@latest
```

```
> ps auxww
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          31  325  0.5 6181560 1357580 ?      Sl   Sep04 2482:28 service-entrypoint
root          84  0.0  0.0   4164  3396 pts/0    Ss   11:00   0:00 bash
root          91  0.0  0.0   6760  3020 pts/0    R+   11:00   0:00 ps auxwww
```

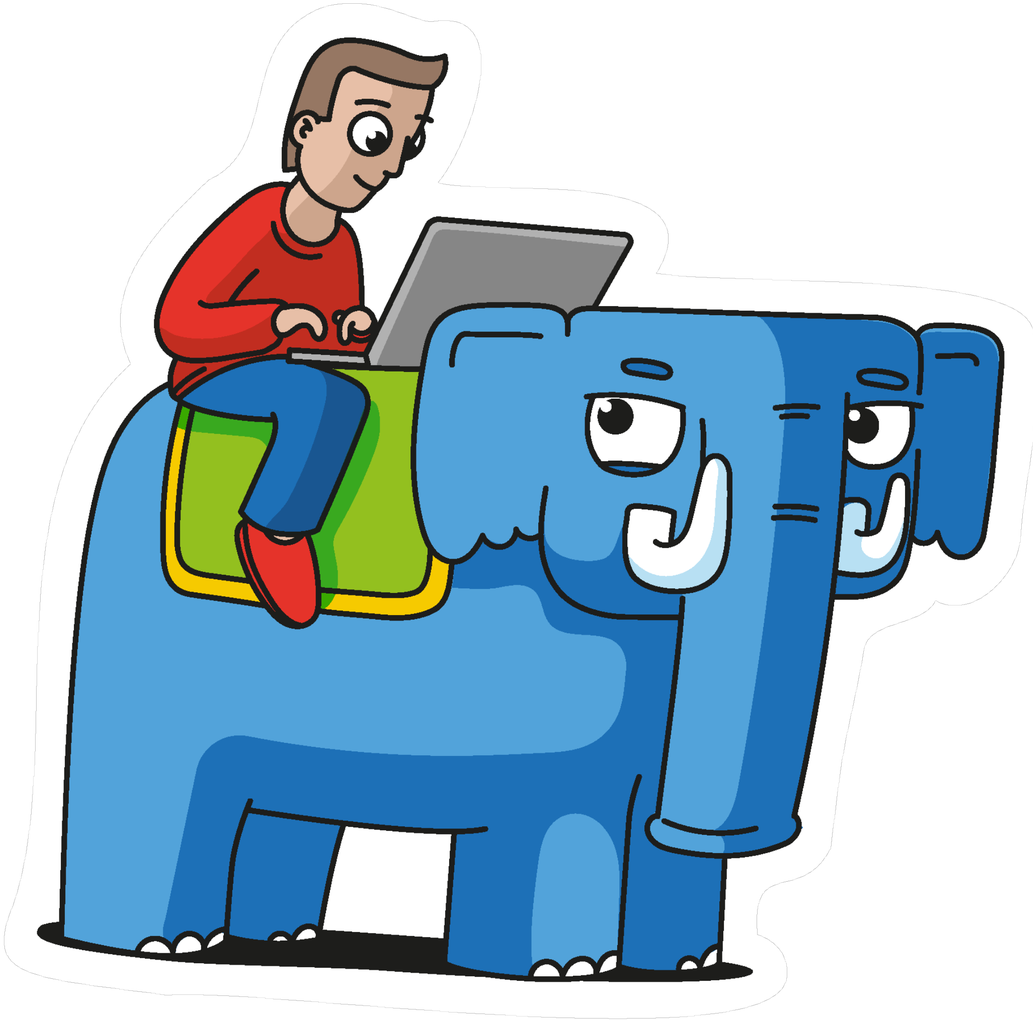
Первым делом, надо поставить отладчик dlv

Первым делом, надо поставить отладчик dlv



Требования к приложению

- Нужно иметь его исходный код
- Желательно собрать приложения без удаления отладочной информации



В итоге

Как отлаживать на проде?

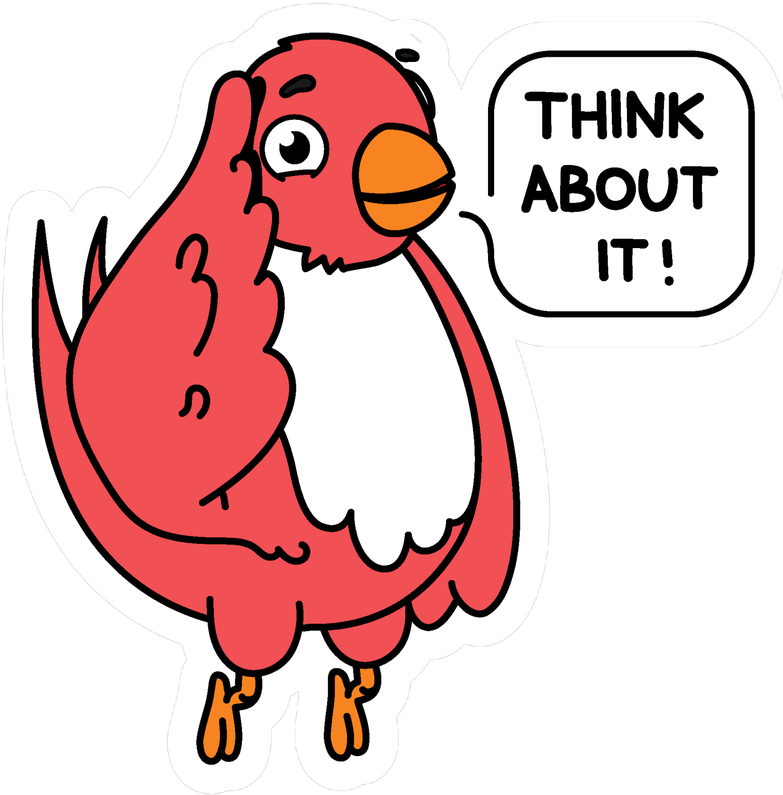
- Собирать с приложения как можно больше информации, позволяющей понимать, что с ним происходит

Логи

- Позволяют детализировать происходящее, должны содержать контекст того, что происходит
- Дополняют метрики деталями в интересные нам моменты

Метрики

- Полезны для алертов, информации о нагрузке, таймингах
- Желательно покрывать ими количество ошибок в сервисе



Pprof

- Профайлинг дает ответы на вопрос «почему тормозит»
- Можно снимать профайл по месту
- … или иметь continuous profiling

Трейсинг

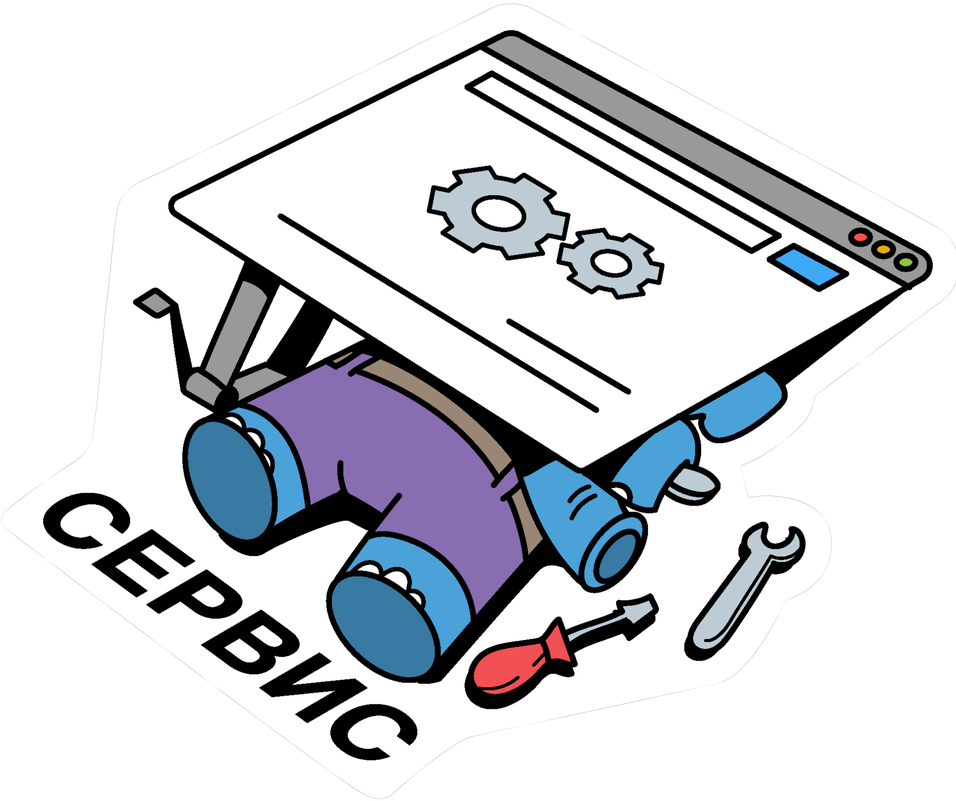
- Позволяет смотреть «сквозь» большое количество сервисов и слоев интеграции
- Дает информацию о том, в каком месте есть проблема для более глубокого анализа

Отладчик

- Для совсем тяжелых случаев
- Стоит избегать на проде
- Но иногда может быть средством «последнего шанса»

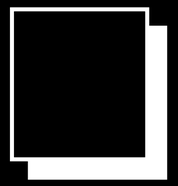
Все вместе

- Все эти инструменты дополняют друг друга
- У них есть своя цена: сложнее код, больше нагрузка на систему
- Но без них отладить проблему будет почти невозможно, поэтому стоит инвестировать в их использование в сервисе



Павел Агалецкий

Авито, ведущий инженер в платформе



@psagaletskiy

Теперь точно все,

спасибо за внимание!