Alati za razvoj softvera

Traceing, OpenTelemetry, Jaeger

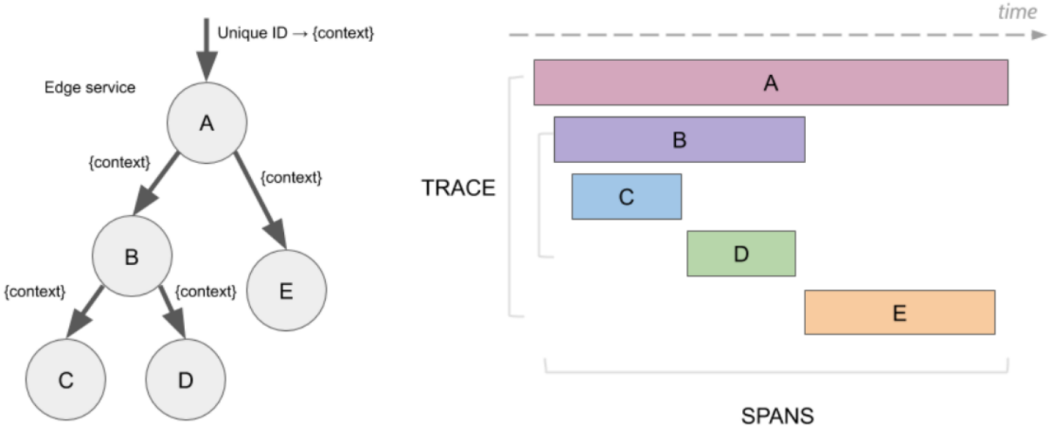
Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Traceing

- Da bi rešili problem kako da nadgledaju distribuiranu infrastrukturu, došlo se do metode koja se zove **tracing**
- Tracing obuhvata skup podataka koji se šalju svim učenicima u komunikaciji da bi se ustanovilo ko je pozvan pre ili posle koga
- Odnosno, da bi dobili kompletan **graf poziva** zajedno sa logovima i svim drugim potrebnim elementima i to na jednom mestu
- Ideja iza tracing-a, je vrlo jednostavna
- Obeležiti početak i kraj izvršavanja svakog parčeta kod-a u svakom servisu, funkciji, metodi itd.

Osnovni pojmovi

- Svako izvršavanje neke celine (npr. funkcije) naziva se **span**
- Unutar svakog span-a, možemo dodati logove, možemo dobiti vreme izvršavanja svakog elementa kod-a
- Ovo nam omogućava da vidimo ujedno i gde su uska grla našeg kod-a, i da to popravimo
- Svi spanovi jednog poziva čine **trace**
- Na taj način znamo ko se izvršavao pre koga, i možemo da dobijemo jedinstven graf izvršavanja
- Najjednostavniji način za prenos informacija kroz naše funkcije u Go-u je paket *Context* o kom je već bilo priče



(Graf poziva i trace)

Instrumentacija

- Instrumentaija je jako bitan aspekt svakog većeg sistema
- Tu je da nam pomogne, upotreba relativno jednostavna
- Postojo dosta alata, Jaeger i Open telemetry su neki od njih
- Upotreba relativno jednostavna – dodatno proǐrenje postojeće baze koda
- Rešava dosta problema

Jaeger and Open telemetry

- Potrebno je dodati alat u *docker-compose*

```
tracing:
        image: jaegertracing/all-in-one
        container_name: jaeger
        ports:
                - "6831:6831/udp"
                - "6832:6832/udp"
                - "16686:16686"
                - "14268:14268"
                - "14250:14250"
```

- Proširiti *applikaciju* dodatnim varijablama

```
app:
        build: .
                restart: always
        ports:
                - "8000:8000"
        depends_on:
        - consul
        environment:
                - DB=consul
                - DBPORT=8500
                - JAEGER_SERVICE_NAME=posts
                - JAEGER_AGENT_HOST=tracing
                - JAEGER_AGENT_PORT=6831
                - JAEGER_SAMPLER_MANAGER_HOST_PORT=jaeger:5778
                - JAEGER_SAMPLER_TYPE=const
                - JAEGER_SAMPLER_PARAM=1
```

Proširenje koda servera

- Potrebno je proširiti server da koristi ovaj mehanizam

```
const (
        name = "post_service"
)

type postServer struct {
        store  *ps.PostStore
        tracer opentracing.Tracer
        closer io.Closer
}

func NewPostServer() (*postServer, error) {
        store, err := ps.New()
        if err != nil {
                return nil, err
        }

        tracer, closer := tracer.Init(name)
        opentracing.SetGlobalTracer(tracer)
        return &postServer{
                store:  store,
                tracer: tracer,
                closer: closer,
        }, nil
}
```

Proširenje koda

- Potrebno je proširiti naše handler-e i funkcije da koriste ovaj mehanizam
- Za demonstraciju uzećemo kompletan poziva *getPostHandler*

```
func (ts *postServer) getPostHandler(w http.ResponseWriter, req *http.Request) {
        span := tracer.StartSpanFromRequest("getPostHandler", ts.tracer, req)
        defer span.Finish()

        span.LogFields(
                tracer.LogString("handler", fmt.Sprintf("handling get all posts at %s\n", req.URL.Path)),
        )

        ctx := tracer.ContextWithSpan(context.Background(), span)
        id := mux.Vars(req)["id"]
        task, err := ts.store.Get(ctx, id)
        if err != nil {
                http.Error(w, err.Error(), http.StatusBadRequest)
                return
        }
        renderJSON(ctx, w, task)
}
```

Proširenje koda

- Potrebno je proširiti funkcije koje rade sa bazom da koriste ovaj mehanizam
- Za demonstraciju uzećemo kompletan poziv *Get* funkcije

```
func (ps *PostStore) Get(ctx context.Context,id string) (*RequestPost, error) {
        span := tracer.StartSpanFromContext(ctx, "Get")
        defer span.Finish()

        kv := ps.cli.KV()

        pair, _, err := kv.Get(constructKey(id), nil)
        if err != nil {
                tracer.LogError(span, err)
                return nil, err
        }

        post := &RequestPost{}
        err = json.Unmarshal(pair.Value, post)
        if err != nil {
                tracer.LogError(span, err)
                return nil, err
        }
        return post, nil
}
```

Proširenje koda

- Potrebno je proširiti pomoćne funkcije da koriste ovaj mehanizam
- Za demonstraciju uzećemo kompletan poziv *decodeBody* funkcije

```
func decodeBody(ctx context.Context,r io.Reader) (*ps.RequestPost, error) {
        span := tracer.StartSpanFromContext(ctx, "decodeBody")
        defer span.Finish()

        dec := json.NewDecoder(r)
        dec.DisallowUnknownFields()

        var rt ps.RequestPost
        if err := dec.Decode(&rt); err != nil {
                tracer.LogError(span, err)
                return nil, err
        }
        return &rt, nil
}
```

Proširenje koda

- Potrebno je proširiti pomoćne funkcije da koriste ovaj mehanizam
- Za demonstraciju uzećemo kompletan poziv *renderJson* funkcije

```
func renderJSON(ctx context.Context,w http.ResponseWriter, v interface{}) {
        span := tracer.StartSpanFromContext(ctx, "decodeBody")
        defer span.Finish()

        js, err := json.Marshal(v)
        if err != nil {
        tracer.LogError(span, err)
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
        }

        w.Header().Set("Content-Type", "application/json")
        w.Write(js)
}
```

Važna napomena

- Važno je napomenuti da funkcije kao na primer: *StartSpanFromContext*, *ContextWithSpan*, *LogError*, *LogFields* su pomoćne funkcije
- Ove funkcije se nalaze unutar *tracer* paketa i *tracer.go* fajla
- Omogućavaju nam lakši rad sa primitivama alata
- Potrebno je kopirati ceo paket u vaš projekat
- Moguće je da vi napišete vaše helper funckije

Dodatni materijali

- Jaeger docs
- opentracing,jaeger, golang
- Beginner's Guide to Jaeger + OpenTracing Instrumentation for Go Albert Teoh
- Dapper, a Large-Scale Distributed Systems Tracing Infrastructure

Kraj predavanja

Pitanja? :)