Event Sourcing

Servisno orijentisane arhitekture

Univerzitet u Novom Sadu
Fakultet tehničkih nauka

Event sourcing

- **Event sourcing** je šablon za skladištenje podataka u sistemu kao niza događaja
- Događaji se čuvaju u hronološki uređenom append-only logu (**event store**)
- **Događaj** predstavlja bilo koju promenu stanja sistema i njegov sadržaj se ne može menjati
- Trenutno stanje entiteta (ili njegovo stanje u nekom trenutku u prošlosti) može se formirati agregacijom događaja relevantnih za taj entitet u redosledu njihove pojave

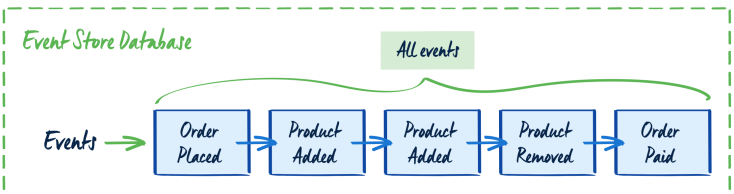
- Kada primenjujemo klasičan pristup perzistenciji podataka (čuvamo samo trenutno stanje), gubimo informacije koje nam mogu biti bitne za rad sistema (stanje sredstava na računu, sistemi koji analiziraju ponašanje kupaca prilikom online kupovine itd)
- Uz event sourcing, imamo vrlo detaljan uvid u sve akcije koje su se desile
- Možemo se jednostavno vratiti u bilo koji trenutak u prošlosti
- Možemo pronaći tačan uzrok neke greške
- Možemo praviti različite poglede na naše podatke po potrebi

Događaj

- Događaj predstavlja činjenicu o akciji koja se desila, a vezana je za domen našeg sistema – nepromenljiva (immutable) poslovna činjenica (*OrderPlaced, PostLiked, MessageSent ...*)
- Kako na osnovu događaja kreiramo trenutno stanje sistema, oni za nas predstavljaju source of truth, bez njih sistem je blokiran i ne može doneti nikakvu odluku
- Snapshot događaja u nekom trenutku (trenutno stanje) nam nije toliko važno, uvek ga možemo uništiti i napraviti novi snapshot na osnovu postojećih događaja

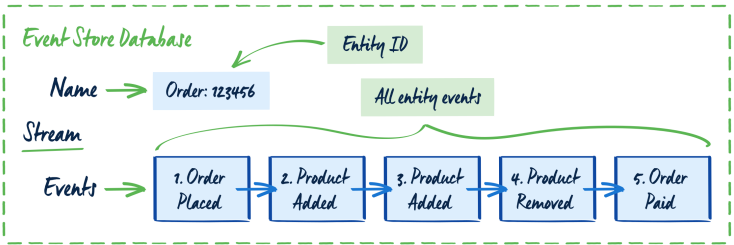
Event store

- Event store predstavlja bazu podataka u kojoj ćemo da čuvamo naše događaje
- Pored baza koje ste do sad radili, postoje i specijalizovane baze za skladištenje događaja
- One optimizuju svoje operacije koristeći append-only immutable prirodu loga događaja



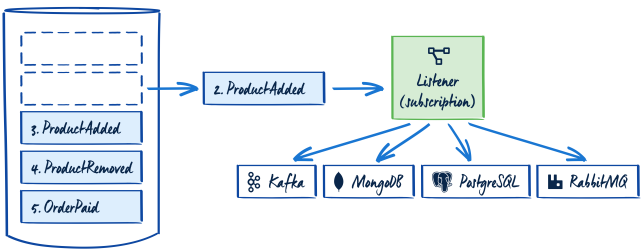
Streams

- Događaji se mogu grupisati u više tokova (streams) tako da događaji koji pripadaju jednom domenu ili objektu pripadaju jednom stream-u
- Događaji u stream-u imaju svoju jedinstvenu poziciju koja je najčešće predstavljena celobrojnom vrednošću koja se inkrementira za svaki novi događaj



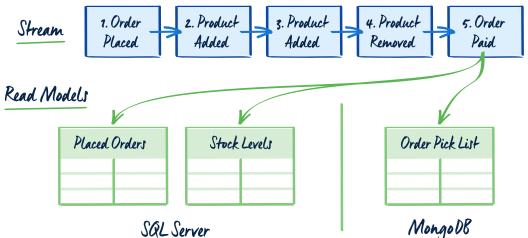
Objavljivanje događaja

- Svaki događaj koji se uskladišti, treba da se objavi na nekom kanalu, kako bi drugi servisi mogli ispravno da ga obrade
- Na primer notification servis sluša događaje PostCreated, PostLiked, FollowingCreated i za svaki od njih kreira notifikaciju
- Zbog asinhronog načina komunikacije, servisi su slabije povezani i izbegavano ulančane blokirajuće pozive među njima
- Čuvanje događaja i njegovo objavljivanje treba da bude atomično



Query model

- Kako bismo mogli da odgovaramo na upite korisnika, potrebno nam je trenutno stanje sistema, ne možemo se osloniti samo na događaje
- Takođe, neke komande moramo validirati pre nego što ih završimo, za šta nam je opet potreban model koji oslikava trenutno stanje sistema



Formiranje query modela

- **Na zahtev** - Generišemo iz događaja kada korisnik pošalje upit ili treba da validiramo komandu, pristup koji nije efikasan kada imamo veliki broj događaja
- **Snapshot** - Periodično pravimo sliku sistema i na nju dodamo samo događaje koji su kreirani nakon kreiranja snapshot-a, poboljšana efikasnost
- **CQRS** - U realnom vremenu, za svaki događaj koji se sačuva objaviće se poruka u nekom kanalu, query modul treba da sluša te poruke i ažurira query model, tako da on bude spreman kada nam zatreba

Event sourcing + CQRS

Napomene

- Query model je eventually consistent, novi događaji su možda sačuvani u event store, a on još uvek nije ažuriran
- Konkurentno dodavanje događaja u event store može dovesti sistem u nekonzistentno stanje (možemo iskoristiti poziciju događaja u stream-u da rešimo ovaj problem)
- Klijenti koji konzumiraju događaje (na primer query modul) moraju biti idempotentni

EventStoreDB

- EventStoreDB je event store koji ćemo koristiti
- Go driver
- Dokumentacija

Kreiranje klijenta

```
        connString := fmt.Sprintf("esdb://%s:%s@%s:%s?tls=false", cfg.ESDBUser, cfg.ESDBPass,
↪   cfg.ESDBHost, cfg.ESDBPort)
        settings, err := esdb.ParseConnectionString(connString)
        if err != nil {
                log.Fatal(err)
        }
        esdbClient, err := esdb.NewClient(settings)
        if err != nil {
                log.Fatal(err)
        }
```

Dodavanje događaja u stream

- Sadržaj događaja čuvamo kao serijalizovani json string

```
func (e EventStore) Store(stream string, eventType string, event []byte) error {
	id, err := uuid.NewV4()
	if err != nil {
		return err
	}
	eventData := esdb.EventData{
		EventID:     id,
		EventType:   eventType,
		Data:        event,
		ContentType: esdb.JsonContentType,
	}
	opts := esdb.AppendToStreamOptions{}
	_, err = e.client.AppendToStream(context.Background(), stream, opts, eventData)
	return err
}
```

Slušanje događaja

- Kako bismo preuzeli događaje koji se dodaju u event store, potrebno je da kreiramo supskripciju i da se zatim konektujemo na nju
- Možemo slušati događaje iz određenog stream-a ili iz svih stream-ova
- Od klijenata koji su prijavljeni na istu supskripciju i pripadaju istoj grupi, samo jednom će biti prosleđen događaj

```
    opts := esdb.PersistentAllSubscriptionOptions{
            From: esdb.Start{},
    }
    err := client.CreatePersistentSubscriptionAll(context.Background(), group, opts)
    ...
    opts := esdb.ConnectToPersistentSubscriptionOptions{}
    sub, err := s.client.ConnectToPersistentSubscriptionToAll(context.Background(),
↪ s.group, opts)
    if err != nil {
            return err
    }
```

Obrada događaja

```
for {
  e := s.sub.Recv()
  if e.EventAppeared != nil {
        streamEvent := e.EventAppeared.Event
        var event events.Event
        switch streamEvent.EventType {
                ...
        }
        if event == nil {
          log.Println("unknown event type")
          continue
        }
        ...
        if err != nil {
          s.sub.Nack(err.Error(),
↪   esdb.Nack_Retry, e.EventAppeared)
        } else {
          s.sub.Ack(e.EventAppeared)
        }
  }
  ...
```

```
...
  if e.SubscriptionDropped != nil {
    log.Println(e.SubscriptionDropped.Error)
    // retry subscription
    for err := s.subscribe(); err != nil; {}
  }
}
```

command servis

- Aplikacija podržava kreiranje novog bankovnog računa i prenos sredstava između tih računa
- Svi novokreirani računi i sva plaćanja čuvanju se kao događaji u EventStoreDB-u
- *command* servis upravlja svim komandama
- Kada izvrši neku od komandi, servis čuva događaje koji imaju sledeći model:

```
// account created
type Event struct {
        AccountNumber string
        HolderName    string
        ...
}
```

```
// payment made
type Event struct {
        PayerAccountNumber string
        PayeeAccountNumber string
        Amount             uint32
        ...
}
```

query servis

- Prijavljen je da sluša događaje iz svih stream-ova
- Sadržaj događaja deserijalizuje u zavisnosti od njegovog tipa
- Event handler ažurira query model računa na osnovu pristiglog događaja

```
type Account struct {
  AccountNumber                  string
  HolderName                     string
  Balance                        uint32
  LastAppliedPayerPaymentEventNumber int64
}
```

- *LastAppliedPayerPaymentEventNumber* atribut pamti broj poslednjeg događaja koji je povukao sredstva sa tog računa
- Koristimo ga u command servisu kada kreiramo novo plaćanje kako bismo validno proverili trenutna sredstva na računu platioca
- Zahtevamo da u event store-u ne postoje događaji plaćanja sa tog računa nakon onog događaja koji je zabeležio query model

Zadaci

- Proširiti primer komandom za izmenu HolderName atributa računa
- Proširiti command servis odgovarajućim command handler-om i nakon izvršene komande sačuvati događaj u event store
- Proširiti query servis event handler-om koji će da izmeni trenutni query model na osnovu događaja za izmenu HolderName atributa