From Vision to Cloud

Martin Lorenz

martinlorenz30

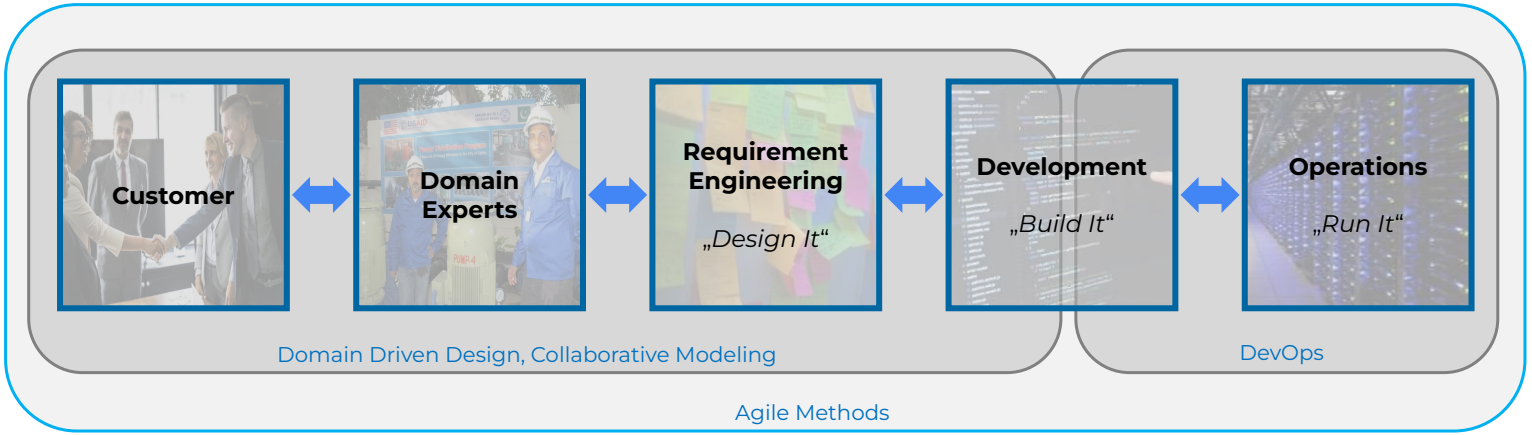
Goal

- Show you a new end-to-end approach to develop business applications

- Problems I see in typical current software development projects
- Present solutions for those problems

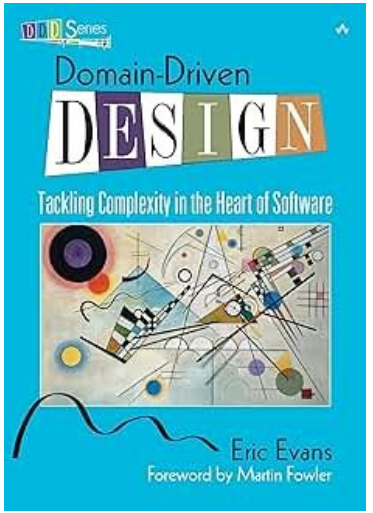
- Introduce a framework that packages up those solutions

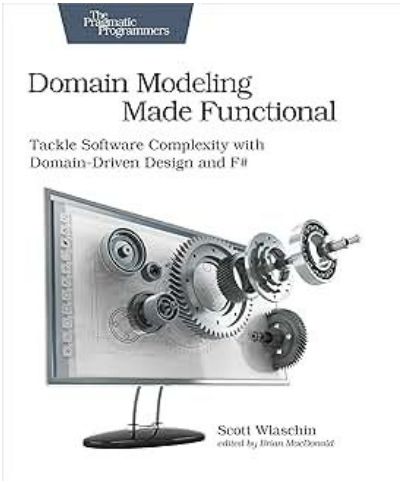
Problem: Communication between stakeholders



Solution: Domain-Driven Design

- Ubiquitous language
- Strategic Patterns
  - Problem Space
    - Core Domain, Generic Subdomain, Supporting Subdomain
  - Solution Space
    - Bounded Context, Context Mapping
- Tactical Patterns
  - Entity, Value Object, Aggregate, Domain Event
- Collaborative Modeling





Solution: Collaborative Modeling

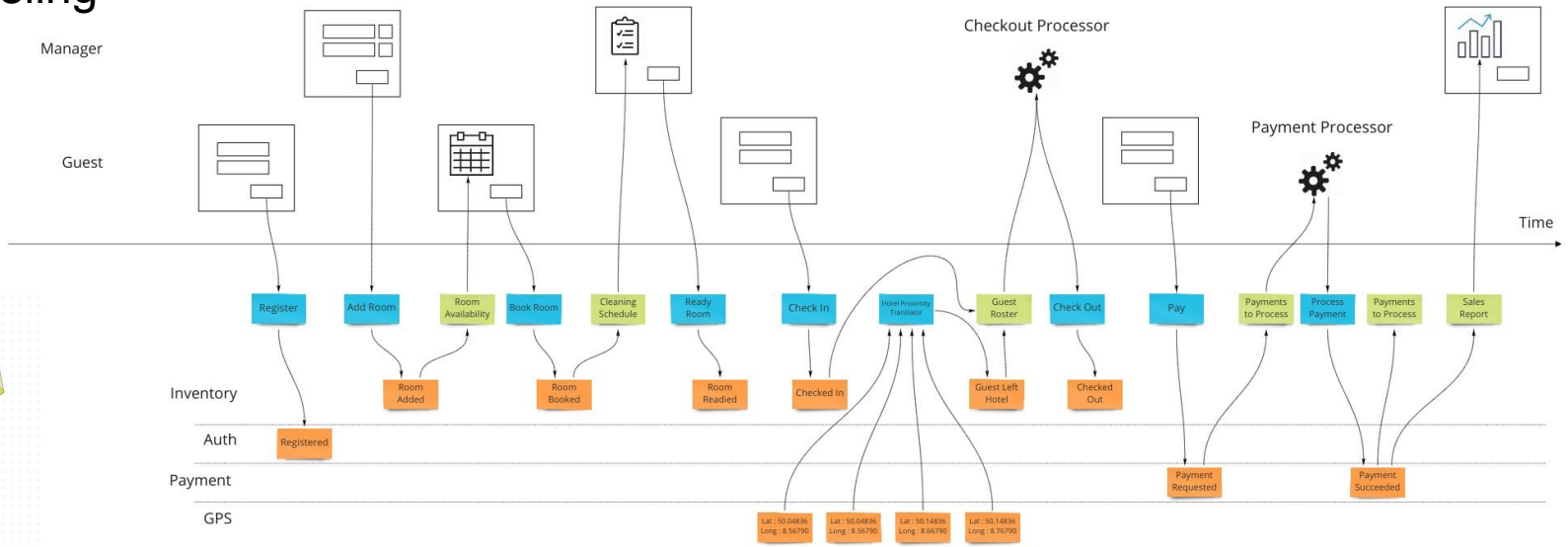
Event Storming

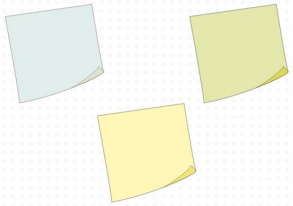
https://www.eventstorming.com

Solution: Collaborative Modeling

https://eventmodeling.org

Event Modeling





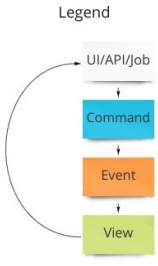
A hands-on Guide

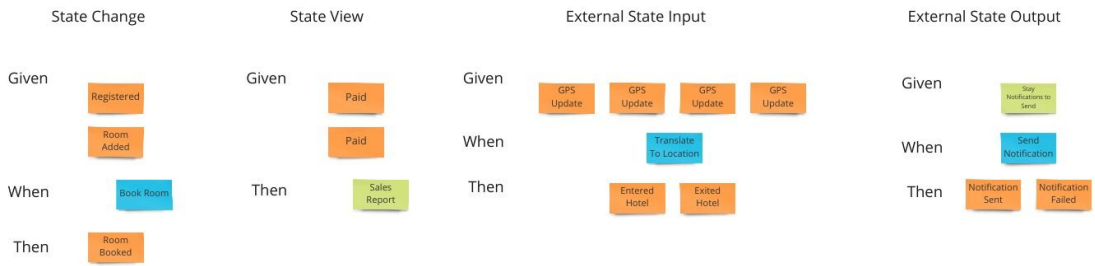
UNDERSTANDING EVENTSOURCING

PLANNING AND IMPLEMENTING SCALABLE SYSTEMS WITH EVENTMODELING AND EVENTSOURCING

Martin Dilger

Forewords by Adam Dymitruk and Gabriel N. Schenker



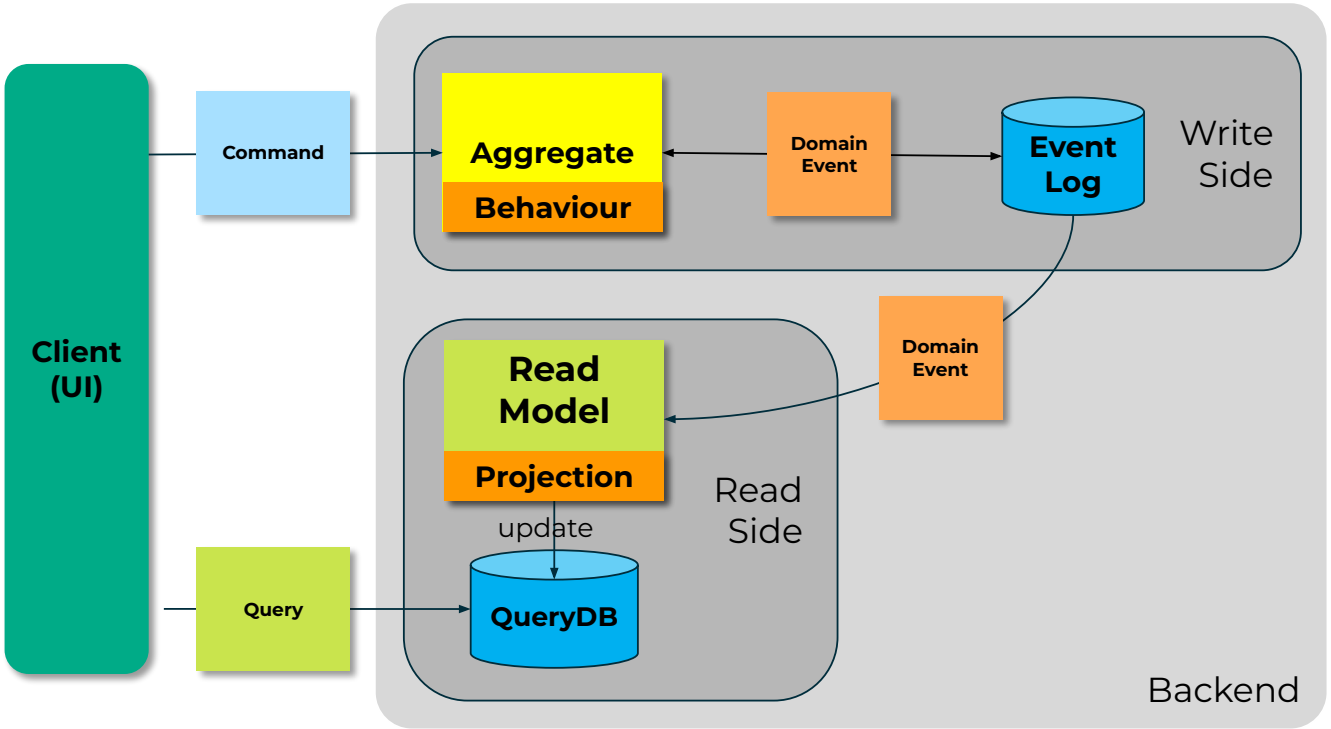


Problem: Coupling between Services

- Splitting up code base into Microservices (bounded contexts)
  - One team is responsible for design, implementation, test, deployment, maintenance
- Often high coupling between services
  - Database sharing
  - Code sharing
  - Synchronous calls between services
  - Monolith => Distributed Monolith
  - Services can not be tested independently => Mocking necessary

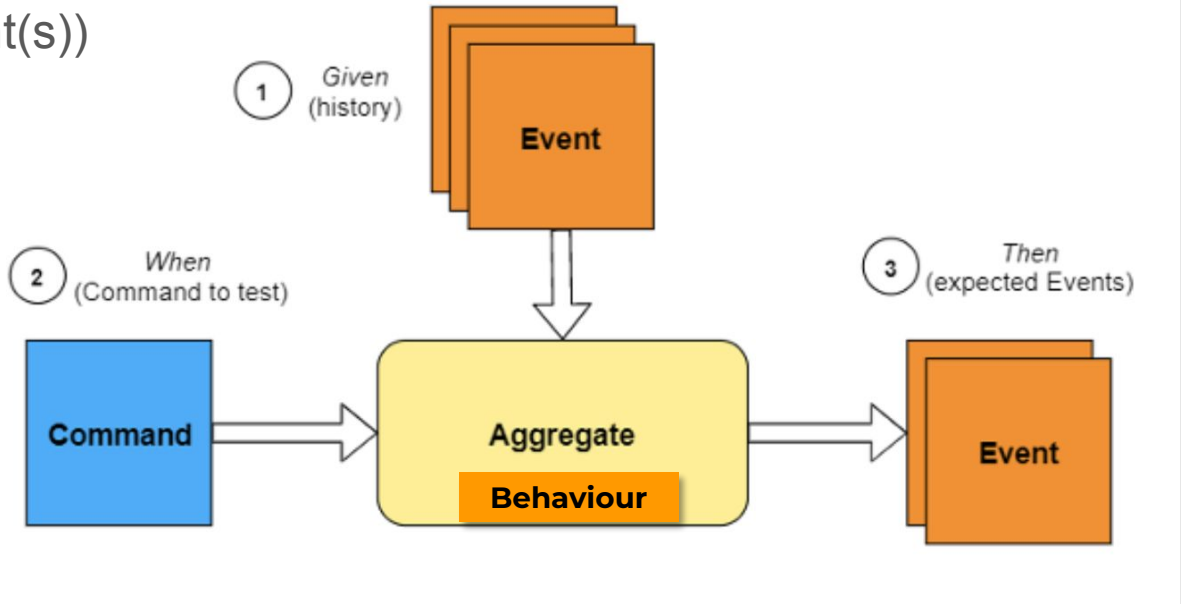
Solution: Event-Driven Architecture

Event Sourcing & CQRS (Command Query Responsibility Segregation)



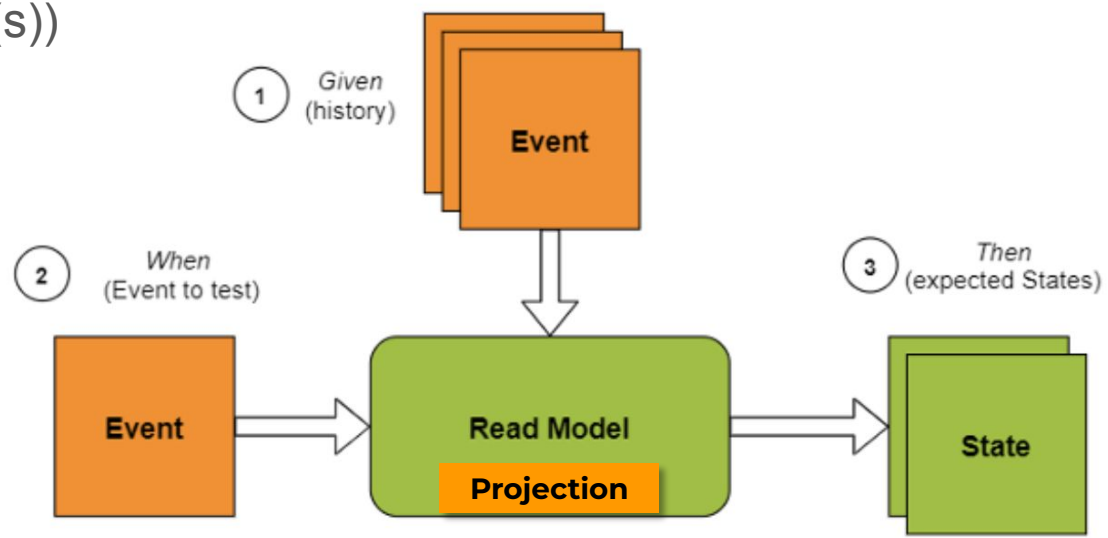
Behavior Test

1. **Given** (Event history)
2. **When** (Command to test)
3. **Then** (expected Event(s))

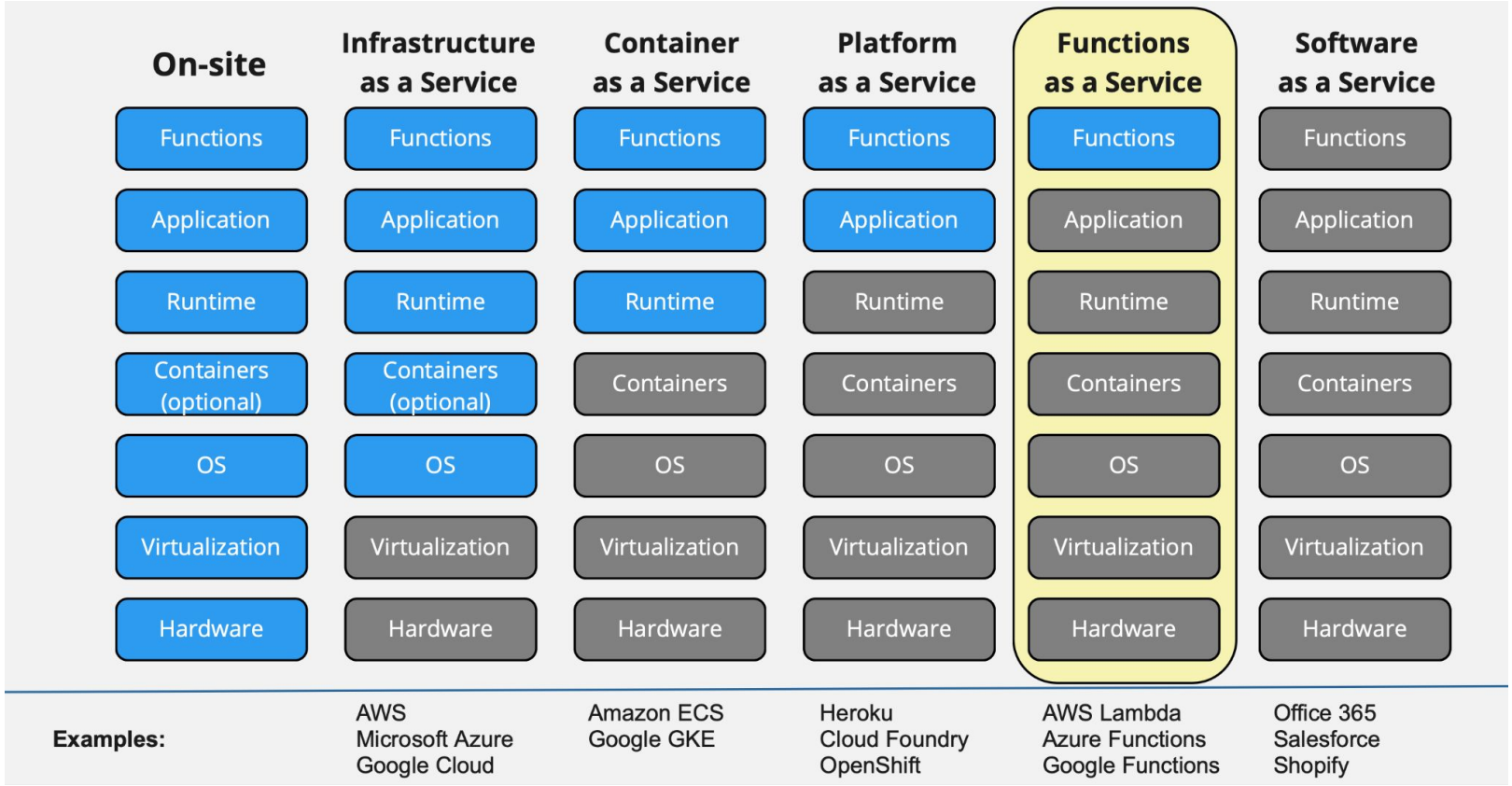


Projection Test

1. **Given** (Event history)
2. **When** (Event to test)
3. **Then** (expected State(s))



Cloud Computing Service Models



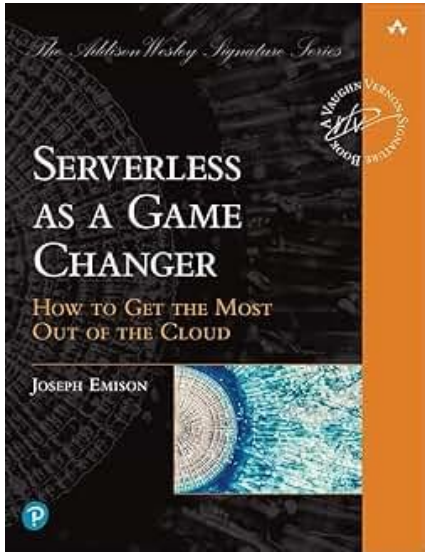
Problem: High efforts & costs for operation

- Maintain Infrastructure for On-site & IaaS
- Maintain containers & clusters with PaaS & CaaS

- High operational costs even with little load when starting & experimenting

- Difficult to scale when you are successful and load is increasing

Solution: Serverless Cloud



- Build and run applications without thinking about servers
  - Only use services that are managed by the cloud provider
  - Connect services by configuration and "glue" code (FaaS)
  - Event- driven architecture (EDA)
- Focus on business logic
- Ship faster
- Automatically scale up & down
- Pricing is based on the actual amount of resources consumed

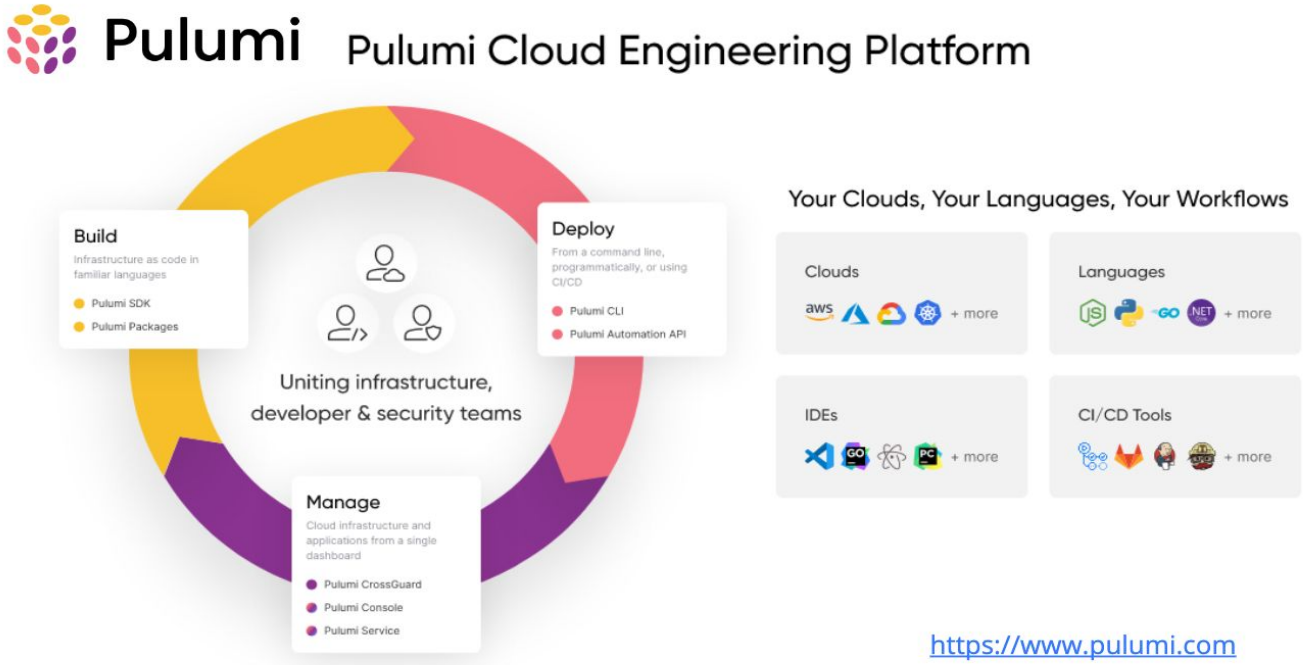
- Cons (mitigations on upcoming slides)
  - Vendor Lock- in
  - Difficult management of high number resources
  - Resource limits (not applicable for all computing workloads)

Problem: Hard to manage infrastructure resources

- First tests & experiments can be done in AWS Console, but not whole applications
  - 100s of resources
  - repetitive work
  - error prone
  - unclear, what has to be to created, deleted or changed

Solution: Infrastructure as Code (IaC)

- **Automate** the provisioning of cloud infrastructure
- imperative vs. **declarative** approach
- DSLs (YAML, JSON, HCL) vs. **General purpose languages**



https://www.pulumi.com

Problem: Bad Developer Experience (DX)

- lots of boilerplate code
- lots of errors only visible at runtime => lots of trivial tests necessary
- Mutability by default
  - Unpredictable state changes
  - Concurrency hazards
  - Debugging complexity
- Side effects
  - Coupling & hidden state
  - Testing complexity
  - Mocking necessary
- often no real full stack experience

Solution: Functional Programming with strong typing (Full Stack)

https://rescript-lang.org

https://graphql.org

- functional
- sound type system with exceptional inference
- lightning fast compilation
- compiles to JavaScript
- reuse existing npm libraries (with some "glue code")
- based on mature language OCaml (same family as F#, Haskell)
- well integrated with React (for frontend)

- fully typed
- introspection
- no over/underfetching
- well integrated in ReScript ecosystem

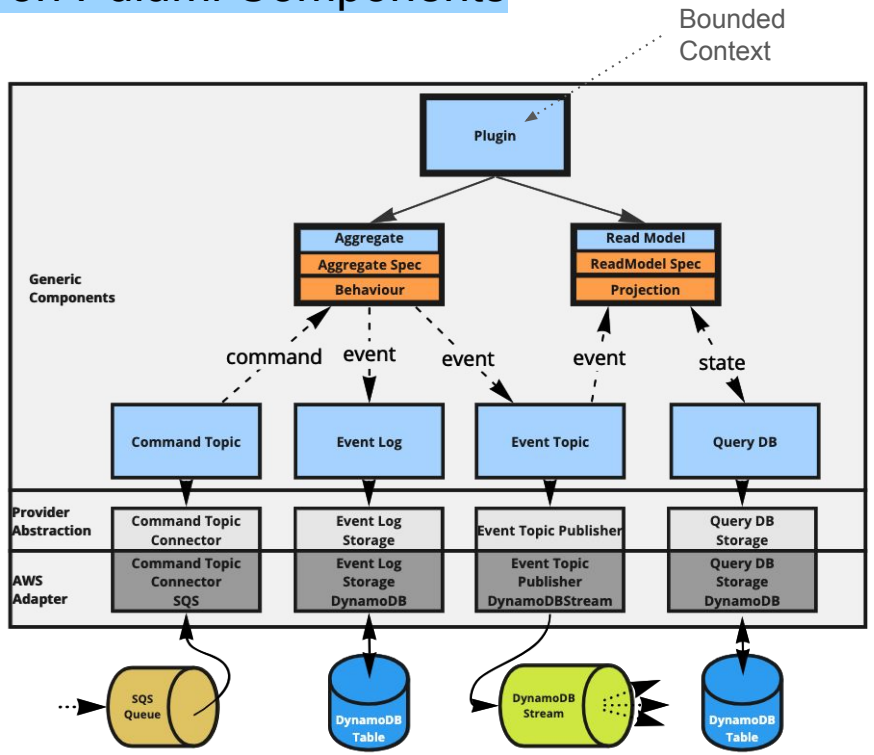
Full Stack - fully typed

- Backend
- API (GraphQL)
- Frontend (React)
- Infrastructure (Pulumi)

Reventless Framework

- **Re-usable Building Blocks** based on Pulumi Components
  - Cloud Infrastructure Resources
  - Run-time functionality
- **Abstractions & Adapters for Cloud Services / Providers**
  - Support different Cloud Services (depending on requirements)
  - Avoid Vendor Lock-In
- **Project/Domain specific code**

- In production for 6+ years
- Soon to be open sourced



Aggregate Spec Example (Customer)

```
@schema
type command =
  | Register({name: string, address: string})
  | ChangeAddress({address: string})
  | ChangeName({name: string})
  | Delete
```

command definitions

```
@schema
type event =
  | Registered({name: string, address: string})
  | AddressChanged({address: string})
  | NameChanged({name: string})
  | Unchanged
  | Deleted
```

event definitions

```
@schema
type error =
  | AlreadyExisting
  | NotExisting
```

error definitions

Aggregate Behaviour Example (Customer_Behaviour)

```
@schema
type state = {
  address: string,
  name: string,
  deleted: bool,
}
```

write side state

```
// command handler for not existing Aggregate: command ⇒ array<event>
let create = (command, context, error) ⇒
  switch command {
  | Register({name, address}) ⇒ [Registered({name, address})]
  | _ ⇒ error(NotExisting, command, context)
  }

// command handler for existing Aggregate: (state, command) ⇒ array<event>
let execute = (state, command, context, error) ⇒
  switch (command, state) {
  | (Register({name, address}), {deleted: true}) ⇒ [Registered({name, address})]
  | (Register(_), {deleted: false}) ⇒ error(AlreadyExisting, command, context)

  | (ChangeAddress(_), {deleted: true}) ⇒ error(NotExisting, command, context)
  | (ChangeAddress({address}), {address: oldAddress}) if address ≠ oldAddress ⇒ [
      AddressChanged({address: address}),
    ]
  | (ChangeAddress(_), _) ⇒ [Unchanged]
  | (ChangeName(_), {deleted: true}) ⇒ error(NotExisting, command, context)
  | (ChangeName({name}), {name: oldName}) if name ≠ oldName ⇒ [NameChanged({name: name})]
  | (ChangeName(_), _) ⇒ [Unchanged]
  | (Delete, {deleted: true}) ⇒ [Unchanged]
  | (Delete, {deleted: false}) ⇒ [Deleted]
  }
```

command handling

```
// projection for not existing Aggregate: event ⇒ state
let init = event ⇒
  switch event {
  | Registered({address, name}) ⇒ {
      address,
      name,
      deleted: false,
    }
  | _ ⇒ invalidEvent(event)
  }

// projection existing Aggregate: (state, event) ⇒ state
let apply = (state, event) ⇒
  switch event {
  | AddressChanged({address}) ⇒ {...state, address}
  | NameChanged({name}) ⇒ {...state, name}
  | Unchanged ⇒ state
  | Deleted ⇒ {...state, deleted: true}
  | Registered(_) ⇒ {...state, deleted: false}
  }
```

event projection

ReadModel Projection Example (Customer_Projection)

```
@schema
type state = {
  name: string,
  address: string,
}
```

read side state

```
// {event, id} ⇒ action<state>
let map = ({event, id}) ⇒ {
  switch event {
  | Customer.Registered({name, address}) ⇒ Create(id, {name, address})
  | AddressChanged({address}) ⇒ Update(id, state ⇒ {...state, address})
  | NameChanged({name}) ⇒ Update(id, state ⇒ {...state, name})
  | Deleted ⇒ Delete(id)
  | Unchanged ⇒ Ignore
  }
}
```

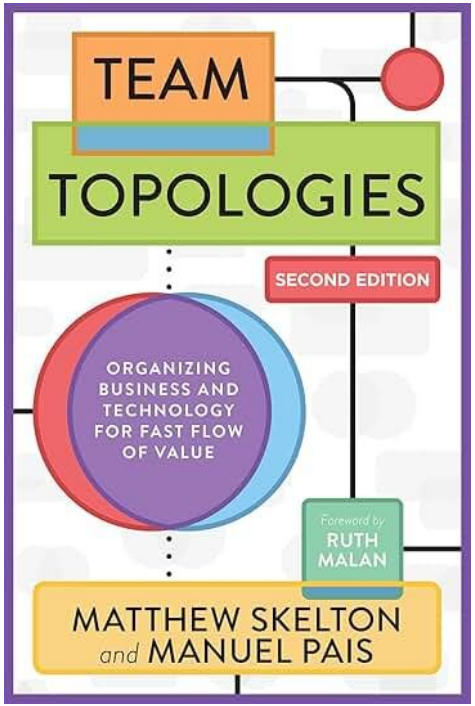
event projection

Clear separation of teams

- **Stream-aligned Teams** (Application development teams)
  - Full focus on the business (and not technology)
  - Create and maintain
    - Project/Domain specific code (as shown on the last slides)
    - A bit of configuration (which Cloud Services should be used)

- **Platform Team**
  - Runs an internal developer platform
  - Based on Reventless framework

- **Reventless Open Source Community**
  - add Adapters for new Services (or new Cloud Providers)
  - improve & optimize existing Adapters
  - introduce new Abstractions
  - All existing & future applications using the Reventless framework benefit from those extensions & improvements



Summary

- Most technologies are quite mature (20+ years)
- Combination is unique - Mindset Shift
- Application development teams can fully focus on the business
- Deploy to the cloud very fast
- Scale automatically up & down
- Keep operational cost down
- Fits well for AI-based development & applications using AI
- It’s a lot of fun to work with !

- Framework Reventless will be Open Sourced soon
  - There are a lot of ideas for improvements & extensions
  - If you are interested: Be part of it !

Martin Lorenz

martinlorenz30