CQRS/ES in Elixir

Nikola Petrusic

theScore

nikola.petrusic@thescore.com

Overview

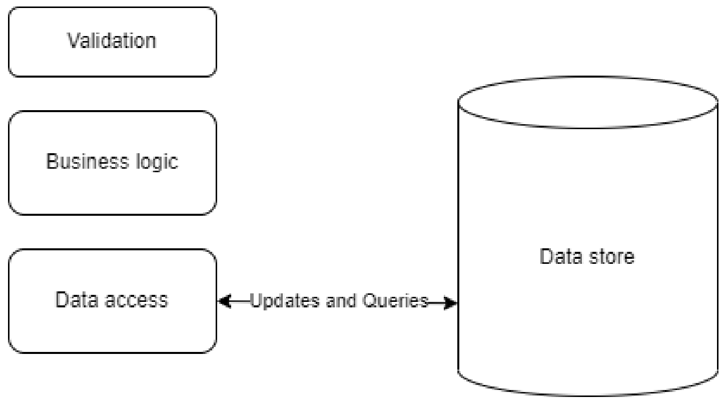
- CQRS (Command and Query Responsibility Segregation)
- ES (Event Sourcing)
- CQRS/ES in Elixir
- Commanded
- Microservices + Kafka
- Comparisons
- Conclusion

CQRS

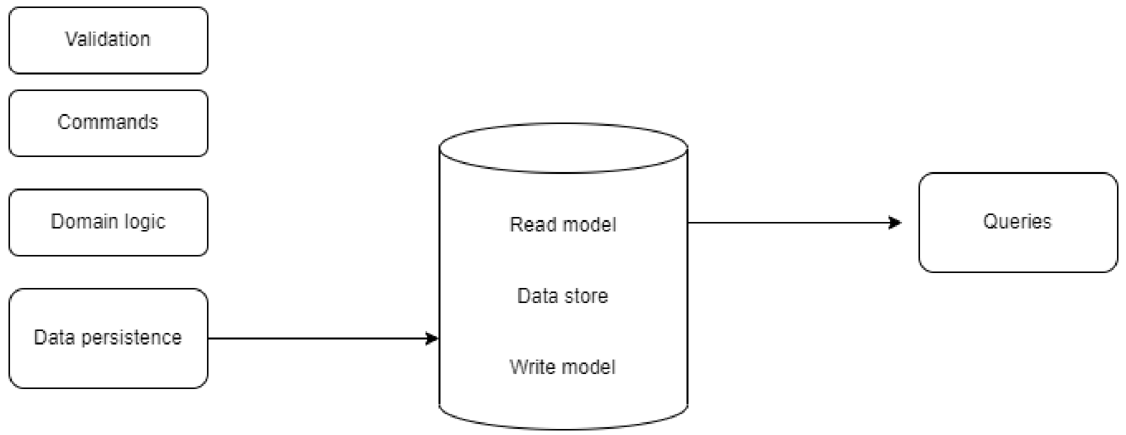
- Command and Query Responsibility Segregation
- Traditionally :More complex application -> Harder to maintain DB model
- Separate read and write models
- Pros: Performance, Scalability, Security
- Cons: More complicated to implement

CQRS

Traditional



CQRS

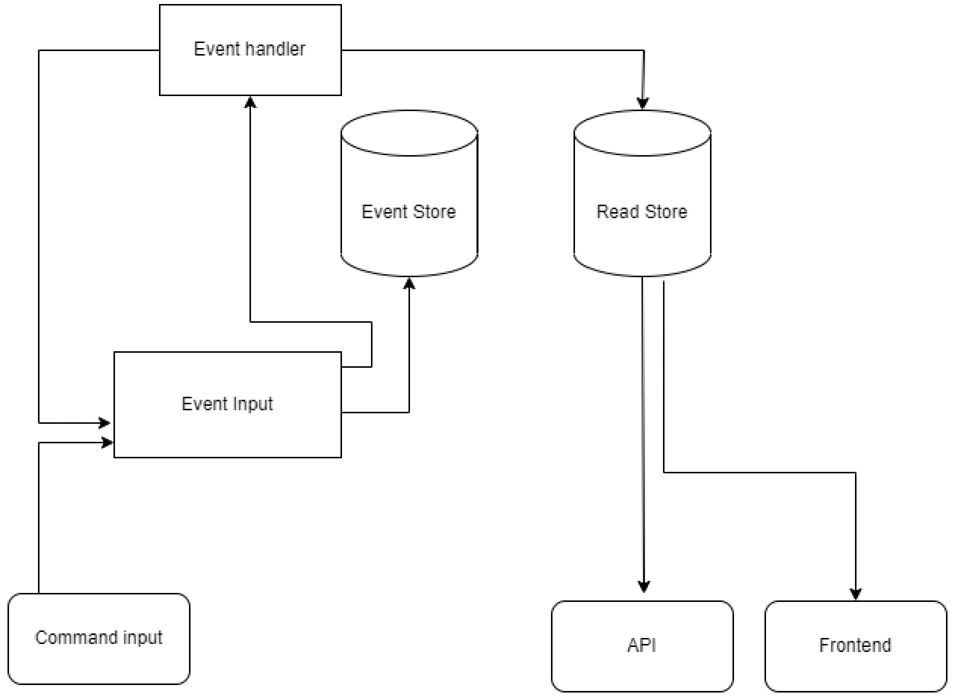


Event sourcing

- Current state is important
- How we got to the current state is also important!
- Changes to the state recorded as series of events
- Pros:
  - Entire changelog of the application accessible for debug, research etc.
  - Async operations (can improve performance)
  - Event store – consistent source of truth
  - Often very natural in the context of the domain
- Cons:
  - More complicated to implement
  - Eventual consistency

CQRS/ES in Elixir

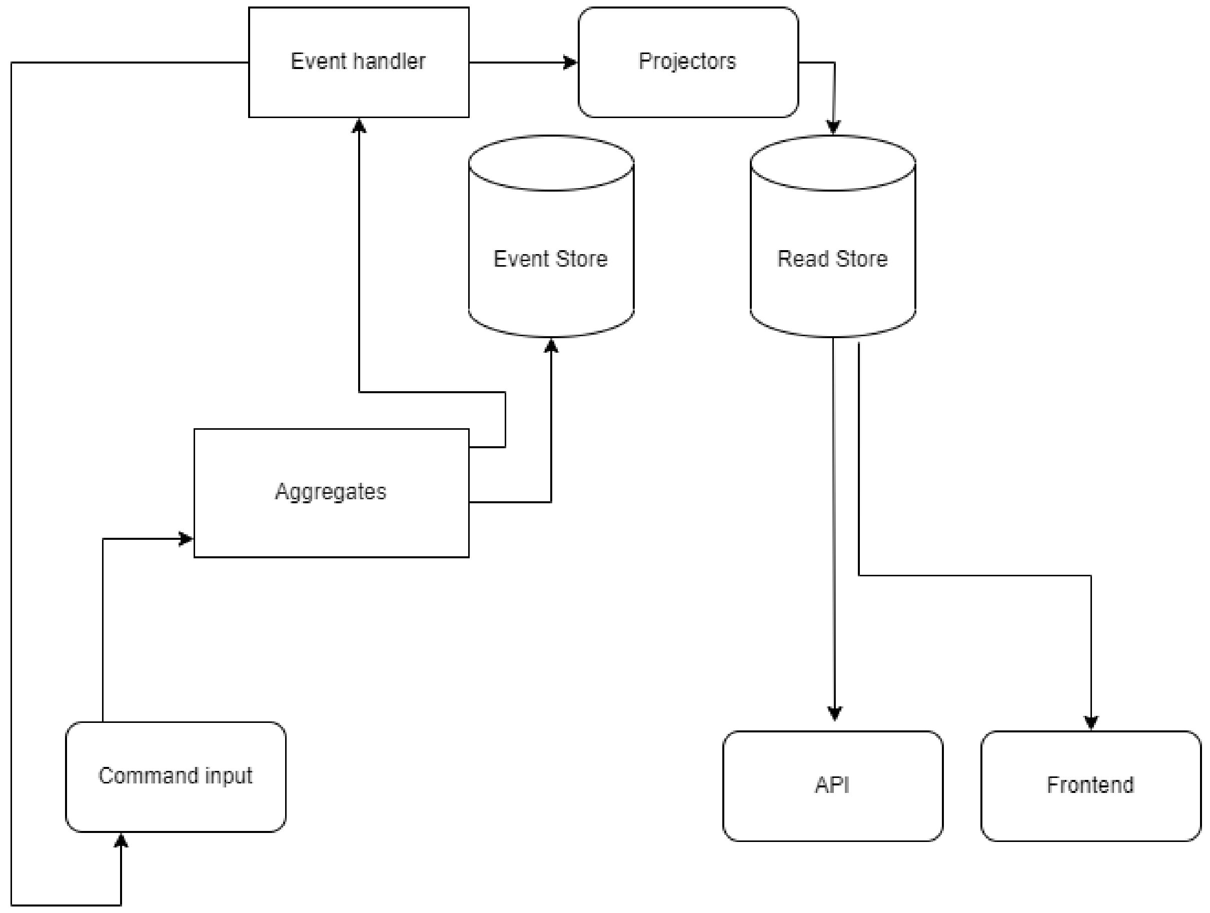
- CQRS and ES work very together
- Elixir -> Low latency, distributed -> Performance, scalability -> CQRS/ES



Commanded

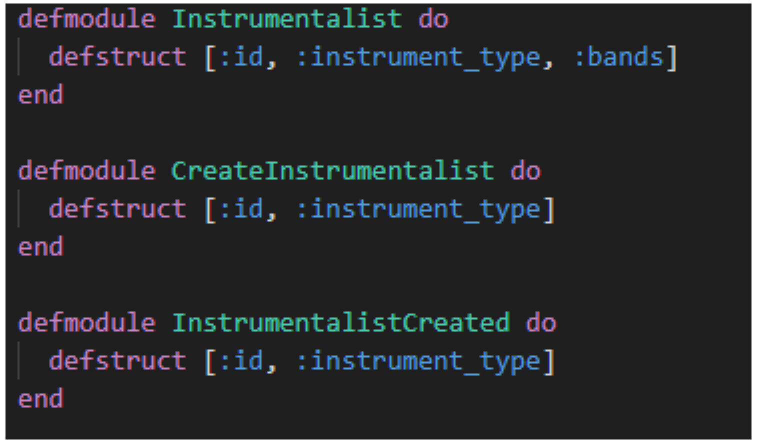
- Elixir framework for CQRS/ES applications
- Provides support for:
  - Command registration and dispatch
  - Hosting and delegation of aggregates (state holders)
  - Event handling
- Event store: Postgres or EventStoreDB

Commanded


Event handler
Projectors
Event Store
Read Store
Aggregates
Command input
API
Frontend

Commanded

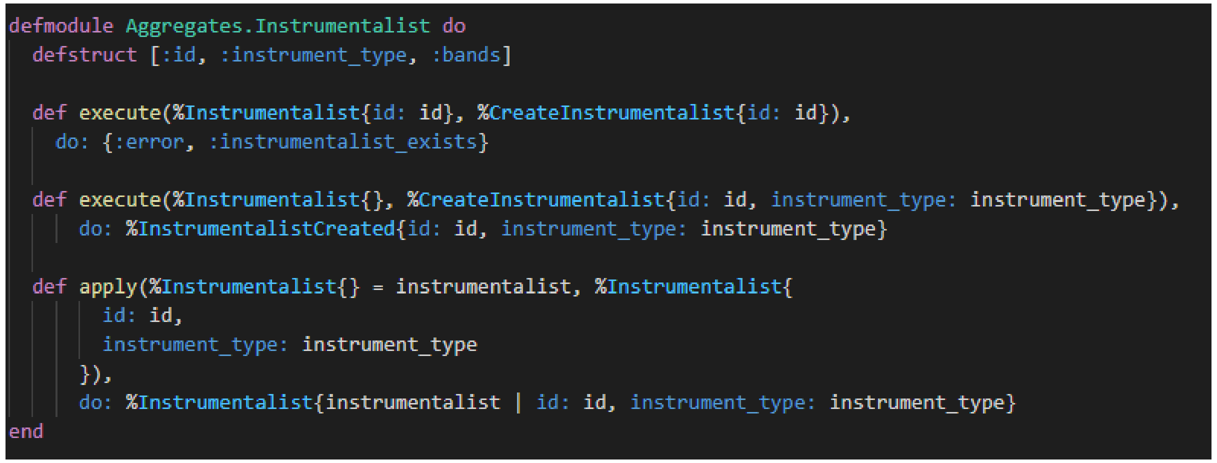
Band finder app



```elixir
defmodule Instrumentalist do
  defstruct [:id, :instrument_type, :bands]
end

defmodule CreateInstrumentalist do
  defstruct [:id, :instrument_type]
end

defmodule InstrumentalistCreated do
  defstruct [:id, :instrument_type]
end
```

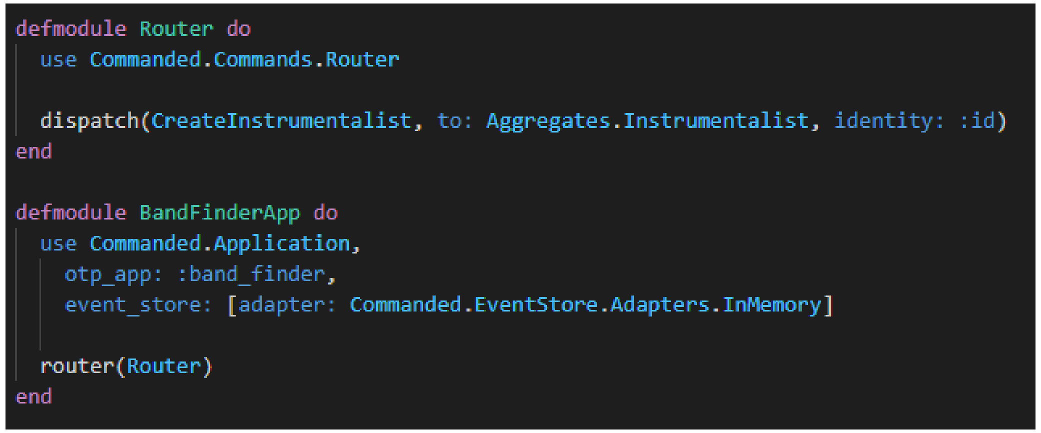


```elixir
defmodule Aggregates.Instrumentalist do
  defstruct [:id, :instrument_type, :bands]

  def execute(%Instrumentalist{id: id}, %CreateInstrumentalist{id: id}),
    do: {:error, :instrumentalist_exists}

  def execute(%Instrumentalist{}, %CreateInstrumentalist{id: id, instrument_type: instrument_type}),
      do: %InstrumentalistCreated{id: id, instrument_type: instrument_type}

  def apply(%Instrumentalist{} = instrumentalist, %Instrumentalist{
        id: id,
        instrument_type: instrument_type
      }),
      do: %Instrumentalist{instrumentalist | id: id, instrument_type: instrument_type}
end
```

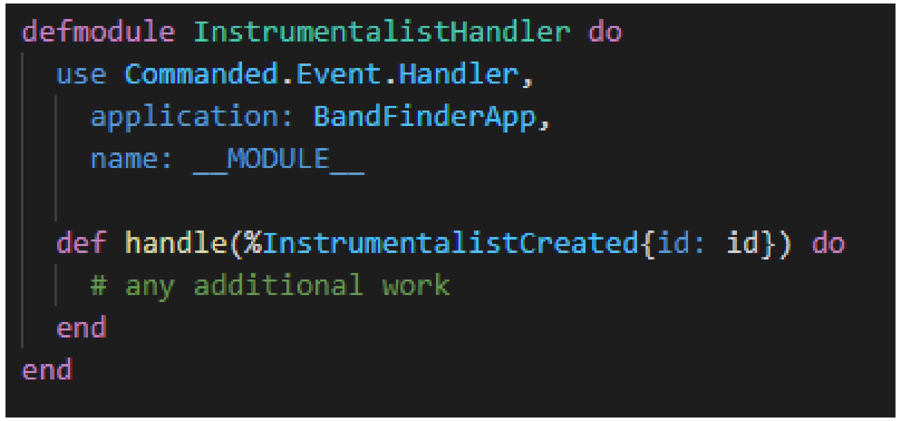


```elixir
defmodule Router do
  use Commanded.Commands.Router

  dispatch(CreateInstrumentalist, to: Aggregates.Instrumentalist, identity: :id)
end

defmodule BandFinderApp do
  use Commanded.Application,
    otp_app: :band_finder,
    event_store: [adapter: Commanded.EventStore.Adapters.InMemory]

  router(Router)
end
```



```elixir
defmodule InstrumentalistHandler do
  use Commanded.Event.Handler,
    application: BandFinderApp,
    name: __MODULE__

  def handle(%InstrumentalistCreated{id: id}) do
    # any additional work
  end
end
```

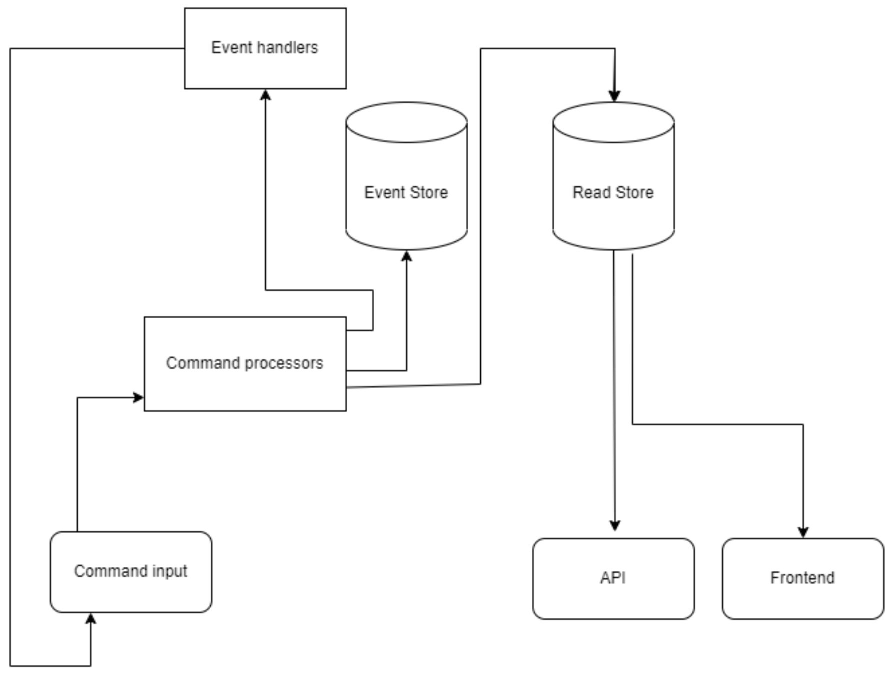
Commanded

- Pros:
  - Ready for production framework for CQRS/ES
  - Relatively easy to implement
  - Pluggable Data store

- Cons:
  - Framework (always comes with limitations and caveats)
  - Event store limited to 2 options
  - Scalability in question

Microservices + Kafka

- Designing architecture to implement CQRS/ES
- Microservices represent parts of CQRS/ES ecosystem we need
- Communication through Kafka



Microservices + Kafka

- Example from before
- Set up Kafka
- Separate service for API (command input and read), Command processor, event handler

- Decision on event storage and data storage (can be Kafka itself)
- Too much code for one slide :)

Microservices + Kafka

- Pros:
  - Scalability
  - Everything is pluggable

- Cons:
  - Might be an overkill
  - Hard to implement/maintain
  - Number of services can grow quite large

Comparisons

- I want Commanded when:
  - CQRS/ES makes sense for my application
  - "Infinite" scalability **is not** a concern
  - Rapid development

- I want Micro + Kafka when:
  - CQRS/ES makes sense for my application
  - "Infinite" scalability **is** a concern
  - Long term expectation of large usage

- I don't want CQRS/ES when:
  - Good ol' CRUD works for my app
  - Need of large instant updates in my application

Conclusion

- CQRS/ES can be very useful
- Provides support for better evolving of the system over time
- There is no one perfect implementation of CQRS/ES in Elixir
- CQRS/ES is the perfect solution for every problem

Thanks for listening

- Useful links:
  - CQRS https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs
  - ES https://docs.microsoft.com/en-us/azure/architecture/patterns/event-sourcing
  - Commanded https://github.com/commanded/commanded
  - Kafka https://kafka.apache.org/