

My first year with event-sourcing

...

Froscon 25-08-2018



Tim Huijzers
@Dragem
Developer @ drukwerkdeal.nl
Founder of DeventerPHP
Deventer NL

First Try

- Limited knowledge
- No experience
- No ES framework
- Doomed from the start

Second Try

- Limited knowledge
- Limited experience
- New Framework
- New DI manager
- New ES Framework
- Doomed To Fail

Third Try

- Some knowledge
- Some experience
- Known framework
- known database
- known ES framework
- Still doomed



I KNOW WHAT I'M DOING!



BeerWarehouse

Why use Event Sourcing

CRUD

Beer
Brewer: Founders
Name: KBS
Bought: 2018-03-12
Location: Fridge
Style: Imperial Russian Stout

We will save a new entry in our system because we just bought it and will store it in the fridge for later.

Beer
Brewer: Founders
Name: KBS
Bought: 2018-03-12
Location: Shelf
Style: Imperial Russian Stout

If we change the location the system only knows about that location.

Beer
Brewer:
Name:
Bought:
Location:
Style:

We drank it so it's not in the system anymore

Beer
Brewer: Founders
Name: KBS
Bought: 2018-03-12
Location: All Gone
Style: Imperial Russian Stout

We want to keep a history of everything we drank.

Beer

Brewer: Founders

Name: KBS

Bought: 2018-03-12

Location: All Gone

Style: Imperial Russian Stout

ConsumptionDate: 2018-05-03

I want to know when I drank this in my history.

But that's only for new beers.

A man in a dark suit, white shirt, and patterned tie is shown from the chest up, looking back over his right shoulder with a serious, intense expression. He is in an office setting with large windows in the background showing a blurred cityscape. Another person's back is visible on the right side of the frame.

THAT'S NOT
GOOD ENOUGH

Events

BeerAddedToStorage
Brewer: Founders
Beer: KBS
Bought: 2018-03-12
Location: Fridge
Style: Imperial Russian Stout

Same Information as before + Explicit action about what happened

Make Small Events

BeerBought
Brewer: Founders
Beer: KBS
Bought: 2018-03-12
Style: Imperial Russian Stout

Removed Location and changed name because in the real world you might not know this yet.

BeerMoved
Identifier: BeerId
Location: Fridge


When returning home I put the beer in my fridge

BeerMoved
Identifier: BeerId
Location: Shelf

I need room in my fridge so I take it out. Using the same
Event

BeerConsumed
Identifier: BeerId

And at last a event about when I consumed it.

A medium shot of Simon Cowell, a judge on the TV show American Idol. He is seated at a panel, wearing a dark suit jacket over a white shirt. He is looking slightly to his right with a focused expression. His right hand is raised towards his chin. In front of him are two orange cups with the 'American Idol' logo and a microphone. The background is dark and out of focus, showing the silhouettes of other people.

BRILLIANT!



Crud

- I know what beer I have.
- I know when it was consumed.
- I know where it is.

Event-Sourcing

- I know what beer I have.
- I know when it was consumed.
- I know where it is.
- I know where it was before.
- I know when it was moved.
- I know where it was at any point in time
- I know how many times it was moved.
- I know when it was added to the system.
- I know what else was moved in that day.

“Every software program relates to some activity or interest of its user.”

Eric Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software

#SISTERSMOVIE

I LIKE THE **STORY** IT TELLS

When To Use Event Sourcing

- You need an audit log
- You like scalability
- You want to separate the read and write of an application
- You want to replay event on a dev machine to get an accurate situation of what the state was at a point in time.
- You want reporting but don't know what yet.
- You are done with mapping objects to tables

When NOT To Use Event Sourcing

- You only need a simple CRUD system.
- You are processing a lot of personal data.
- You just want to query a lot of things on the DB
- You are starting on a big project for production

Event Sourcing in code

Prooph


<http://getprooph.org/>

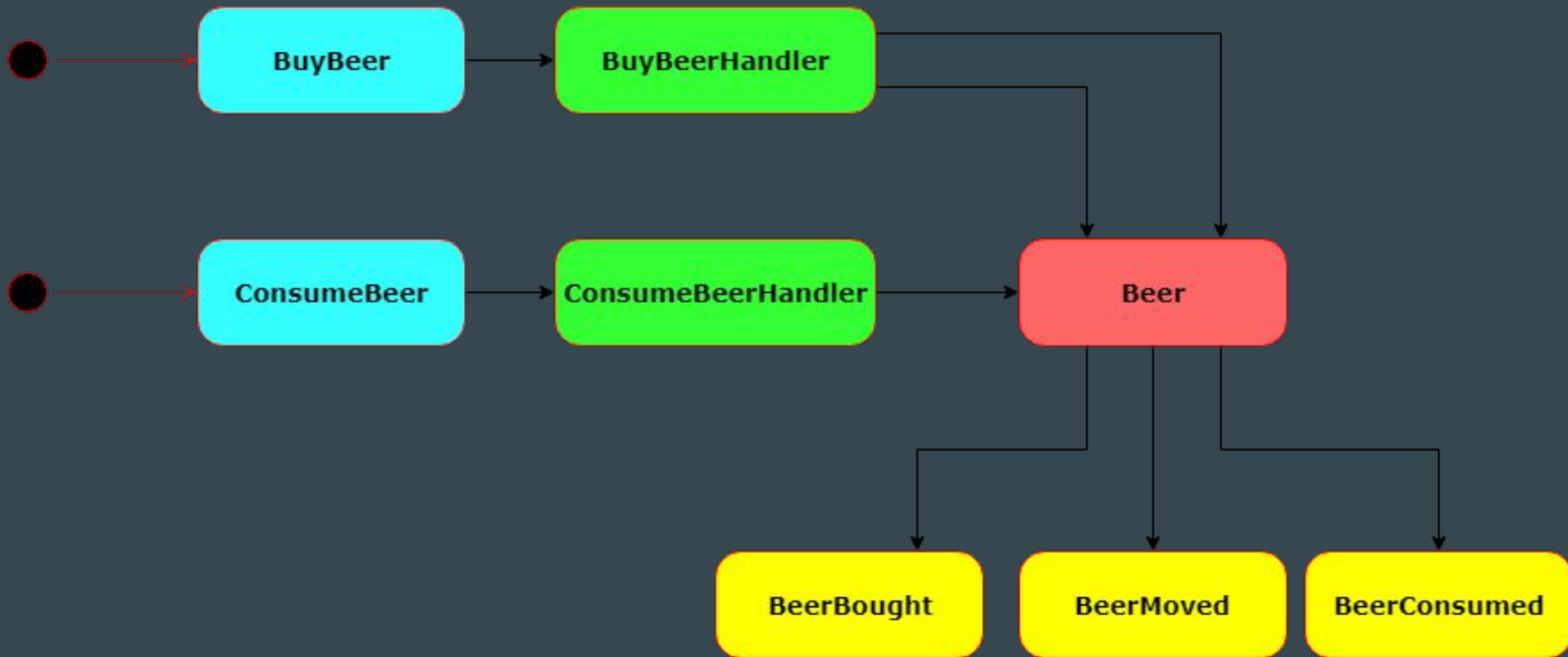


THE CQRS AND EVENT SOURCING COMPONENTS FOR PHP

Crafted for Your Enterprise App

INSTALL

TAKE A TOUR 



Command

```
final class BuyBeer extends Command implements PayloadConstructable
{
    use PayloadTrait;
    public static function forWarehouse(
        string $brewer,
        string $name,
        string $style,
        ?string $location = null
    ): BuyBeer {
        return new self([
            'brewer' => $brewer,
            'name' => $name,
            'style' => $style,
            'location' => $location
        ]);
    }
}
```



```
final class ConsumeBeer extends Command implements PayloadConstructable
{
    use PayloadTrait;
    public static function forWarehouse(
        string $beerId
    ): ConsumeBeer {
        return new self([
            'beerId' => $beerId
        ]);
    }

    public function beerId(): BeerId
    {
        return BeerId::fromString($this->payload['id']);
    }

    protected function setPayload(array $payload): void
    {
        $this->payload = $payload;
    }
}
```

Command Handler

```
final class BuyBeerHandler
{
    private $beerCollection;

    public function __construct(BeerCollection $beerCollection)
    {
        $this->beerCollection = $beerCollection;
    }

    public function __invoke(BuyBeer $command): void
    {
        $beer = Beer::buyBeer(
            $command->brewer(),
            $command->name(),
            $command->style()
        );
        $beer->moveTo($command->location());
        $this->beerCollection->save($beer);
    }
}
```

```
final class ConsumeBeerHandler
```

```
{
```

```
    private $beerCollection;
```

```
    public function __construct(BeerCollection $beerCollection)
```

```
    {
```

```
        $this->beerCollection = $beerCollection;
```

```
    }
```

```
    public function __invoke(ConsumeBeer $command): void
```

```
    {
```

```
        $beer = $this->beerCollection->getBeer($command->beerId());
```

```
        $beer->consume();
```

```
        $this->beerCollection->save($beer);
```

```
    }
```

```
}
```

Aggregate

```
public static function buyBeer(  
    Brewer $brewer,  
    BeerName $name,  
    BeerStyle $style  
): Beer {  
    $self = new self();  
    $beerId = BeerId::fromString((string)Uuid::uuid4());  
    $self->recordThat(BeerBought::withData($beerId, $brewer, $name, $style));  
    return $self;  
}
```

```
public function moveTo(Location $location): void  
{  
    $this->recordThat(BeerMoved::withData($this->beerId, $location));  
}
```

```
public function consume(): void  
{  
    $this->recordThat(BeerConsumed::now($this->beerId));  
}
```

Event

```
final class BeerBought extends AggregateChanged
```

```
{
```

```
    public static function withData(
```

```
        BeerId $beerId,
```

```
        Brewer $brewer,
```

```
        BeerName $name,
```

```
        BeerStyle $style
```

```
): BeerBought {
```

```
    $event = self::occur(
```

```
        (string)$beerId,
```

```
        [
```

```
            'brewer' => (string)$brewer,
```

```
            'name' => (string)$name,
```

```
            'style' => (string)$style
```

```
        ]
```

```
    );
```

```
    $event->beerId = $beerId;
```

```
    $event->brewer = $brewer;
```

```
    $event->name = $name;
```

```
    $event->style = $style;
```

```
    $event->date = BoughtDate::fromDateTime($event->createdAt());
```

```
    return $event;
```

```
}
```

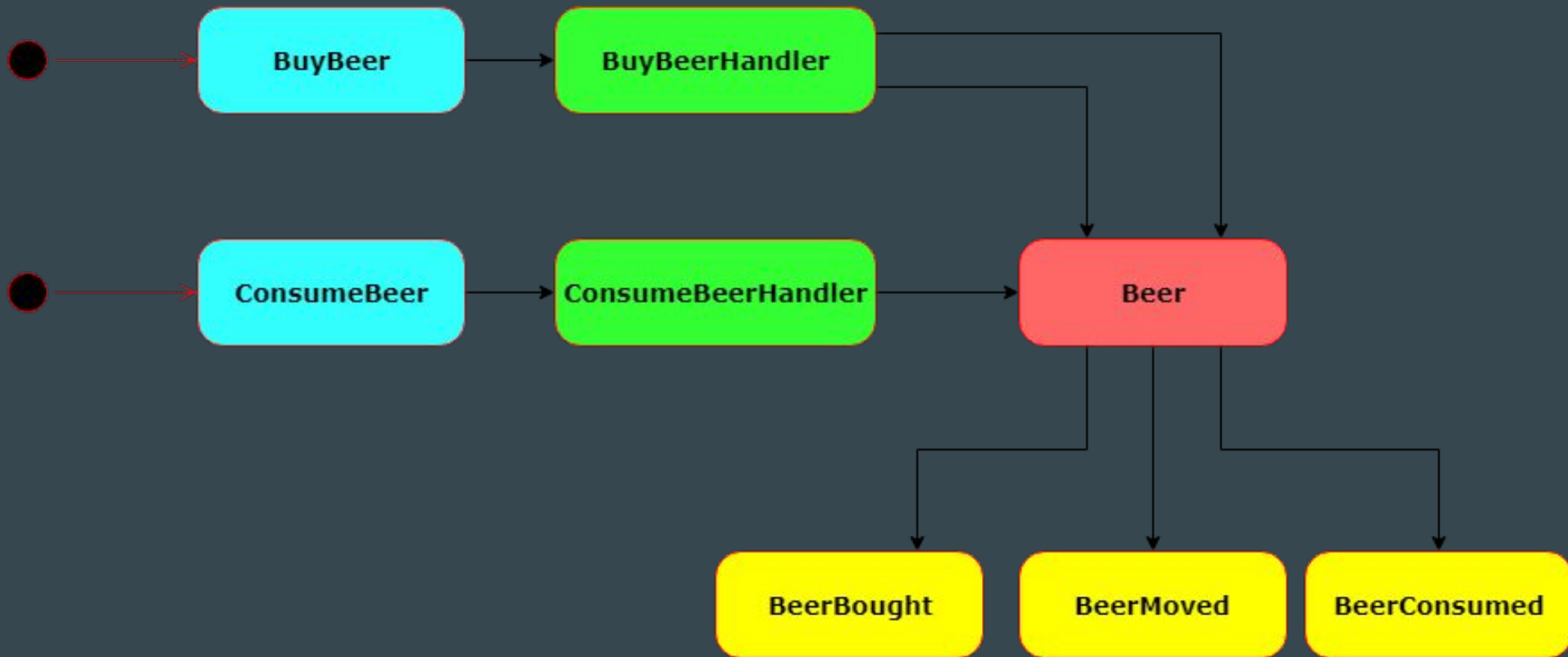

Back to the Aggregate

```
protected function whenBeerWasBought(BeerBought $event): void
{
    $this->brewer = $event->brewer();
    $this->name = $event->name();
    $this->style = $event->style();
    $this->bought = $event->date();
}

protected function whenBeerWasMoved(BeerMoved $event): void
{
    $this->location = $event->location();
}

protected function whenBeerWasConsumed(BeerConsumed $event): void
{
    $this->consumed = $event->date();
}

protected function apply(AggregateChanged $event): void
{
    switch(true) {
        case $event instanceof BeerBought:
            $this->whenBeerWasBought($event);
            break;
        case $event instanceof BeerMoved:
            $this->whenBeerWasMoved($event);
            break;
        case $event instanceof BeerConsumed:
            $this->whenBeerWasConsumed($event);
            break;
    }
}
```

A man with a mustache and dark hair, wearing a light blue suit jacket, a light blue shirt, and a light-colored tie, is shown from the chest up. He is looking off to the left with a slight smile. The background is an office setting with a window on the left, a bookshelf on the right, and a small boat hanging on the wall behind him.

GREAT SUCCESS!

Think About Side Effects

```
interface BeerCollection
{
    public function save(Beer $beer);

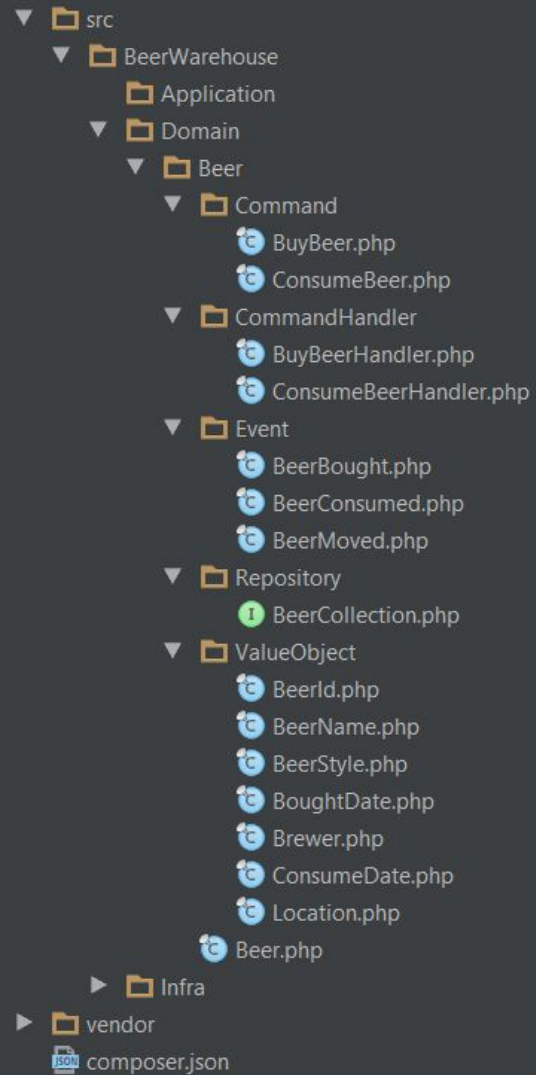
    public function getBeer(BeerId $beerId): Beer;
}
```

A medium shot of Donald Trump speaking at a podium. He is wearing a dark suit, white shirt, and red tie. His mouth is open as if he is speaking, and his right hand is raised with the index finger pointing up. The background consists of several American flags. A microphone is positioned in front of him.

LIVE
RNC
• 2016 •

***We are going to build a
great boarder wall!***

Structuring your application



Understanding the DB

event_streams

no

event_id

event_name

payload

metadata

created_at

aggregate_version

aggregate_id

aggregate_type

How many beers do I have?

How many different styles do I have?

How many beers have I drank last 30 days?

Projection

A Projection allows you to loop through all event (past and present) and build your own views.

- Read Model
 - Define the data you would like to use.
- Projection
 - Loops through the events and applies that data to your view
- Finder
 - Helps you find data from that view.


```
BeerMoved::class => function($state, BeerMoved $event) {  
    /** @var BeerReadModel $readModel */  
    $readModel = $this->readModel();  
    $readModel->stack( type: 'update', [  
        'id' => (string)$event->id(),  
        'location' => (string)$event->location()  
    ]);  
},  
BeerConsumed::class => function($state, BeerConsumed $event) {  
    /** @var BeerReadModel $readModel */  
    $readModel = $this->readModel();  
    $readModel->stack( type: 'delete', [  
        'id' => (string)$event->id(),  
    ]);  
}
```

```
]);
```

```
return $projector;
```

```
}
```

Pitfalls

**Refactoring is harder, think about your
architecture**



OPEN

Open
Mon
Tue-Thur
Fri-Sat
Sunday

Versioning

- Change an Event but support the old version
- Make a new Event
- Make the Event right from the start

Something wrong with the event

Event are immutable, So don't change them

- Try solving it another way first.
- Correct errors with new events
- Try a upcaster
- Make a new stream and fill it with mutated events (and test)
- Change the events in the database

**But what if I have like 100 trillion gazillion
events?**

LEBRON BE LIKE..

A photograph of LeBron James in a red Miami Heat jersey with the number 6. He is standing on a basketball court with his arms outstretched, palms facing forward, in a gesture of calm or protest. The background is a blurred crowd of spectators in a stadium.

CALM THAT SH*T DOWN!

Snapshots

You Do Not Need Snapshots From The start

Trigger on Event Count

Pure Event Sourcing Is Not A Holy Grail

Do Not Save Personal Data In Events

Make Projections For All Your Lists

Try It In A Hackathon First

Most Of The Time Your DB Is Not Holy

What Now?

<http://getprooph.org/>



THE CQRS AND EVENT SOURCING COMPONENTS FOR PHP

Crafted for Your Enterprise App

INSTALL

TAKE A TOUR 



This repository

Search

Pull requests

Issues

Marketplace

Explore



prooph / proophessor-do-symfony

Watch

19

Star

80

Fork

33

<> Code

Issues 4

Pull requests 0

Projects 0

Wiki

Insights

Symfony version of proophessor-do CQRS + Event Sourcing example app <http://getprooph.org/>

75 commits

1 branch

0 releases

11 contributors

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download



prolic use mysql 5.7 only

Latest commit cf0989f 6 days ago

.docker	Rewrite to use flex and prooph v7	7 months ago
bin	Add EventStoreMgmtUi	2 months ago
config	Merge pull request #25 from UFOMelkor/hotfix/projections	7 days ago
docs	Add EventStoreMgmtUi	2 months ago
public	Add EventStoreMgmtUi	2 months ago
src	Add EventStoreMgmtUi	2 months ago

Source

<https://github.com/prooph/proophessor-do-symfony>

<http://getprooph.org/>

Other Tools

- Broadway
 - No Upcaster,
 - No Snapshots,
 - No Replaying
- Axon
 - Upcasting by MessageFactory,
 - Snapshots by Trigger on event count,
 - Replaying by Example code for replay
- Akka
 - Upcasting by Event Adapter,
 - Snapshots decided by actor,
 - Replaying



Thanks, Any Questions?

Example code from talk on:

<https://github.com/webbaard/BeerWarehouse>

