Event Sourcing

Let’s write some history

Quick introduction

Willem-Jan Zijderveld

- @willemjanz
- github.com/wjzijderveld

Qandidate.com

Rotterdam

http://labs.qandidate.com

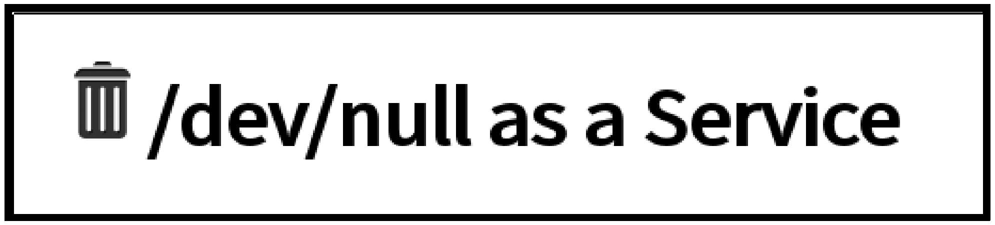
Broadway

Event Sourcing and CQRS library for PHP

github.com/qandidate-labs/broadway

Event Sourcing

You are throwing away data!

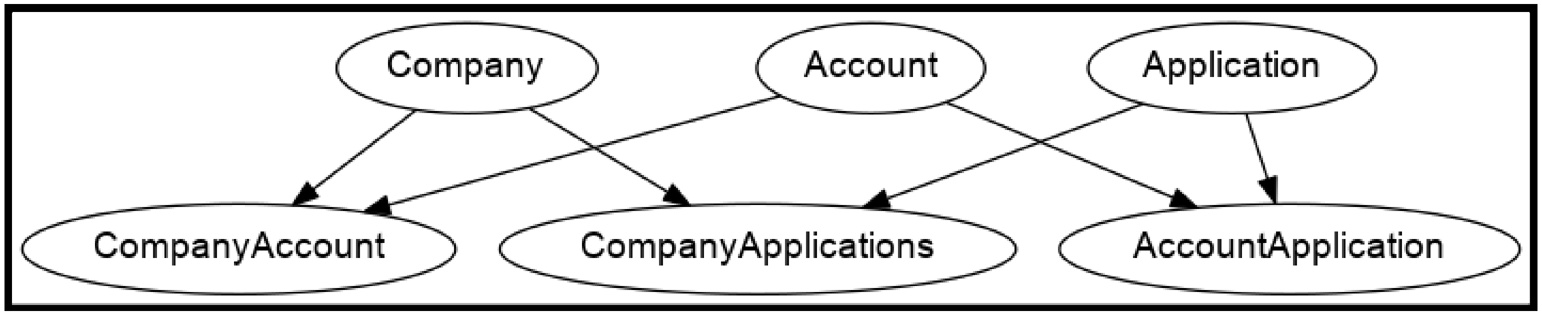

/dev/null as a Service

State gets stored

A **company** has one or more connected **accounts**

A **company** has one or more enabled **applications**

An **account** has access to one or more **applications**


Company
Account
Application
CompanyAccount
CompanyApplications
AccountApplication

Complicated solutions to store and retrieve the data

Boss: Which accounts **received access** to application X in December?

We’ll add a timestamp and we will know

From that point in time

With CRUD you lose information

You only store the latest information

Revoking access to an application for an account

You could lose **when** the access was revoked

You could lose **why** it got revoked

You could lose **for which application** it was revoked

You **wouldn't know** anything anymore

Let's assume you have a soft-delete in place

Boss: I want to know for which applications access got revoked more than once in the last year

Keep track of revocations

account_id	app_id	datetime
75623	8	2015-01-01T00:00:00+0000
75623	8	2015-01-27T18:54:18+0000

Past revocations are gone

How does Event Sourcing help me with that?

Store your data in **a different way**

Record what has **changed**

The resulting state becomes a natural effect

EventStream

A serie of **facts**

CompanyRegistered

AppEnabled

AccountConnected

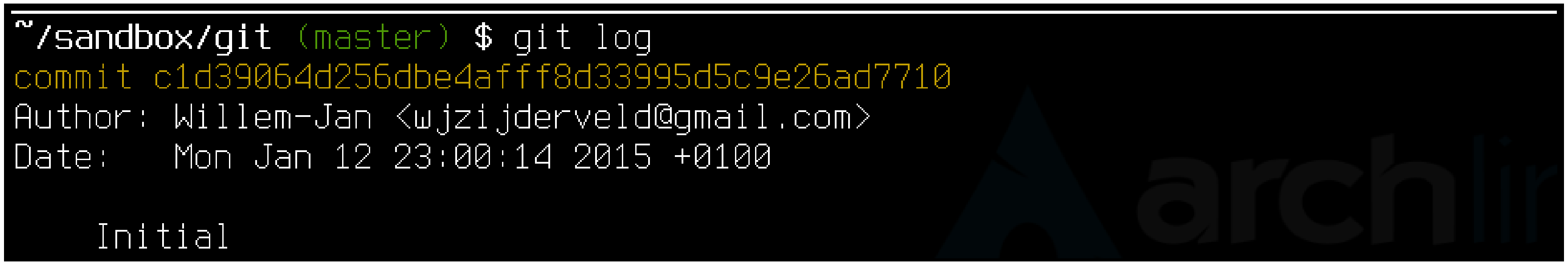
AccessGrantedToApp

Been there, **done** that

Single source of truth

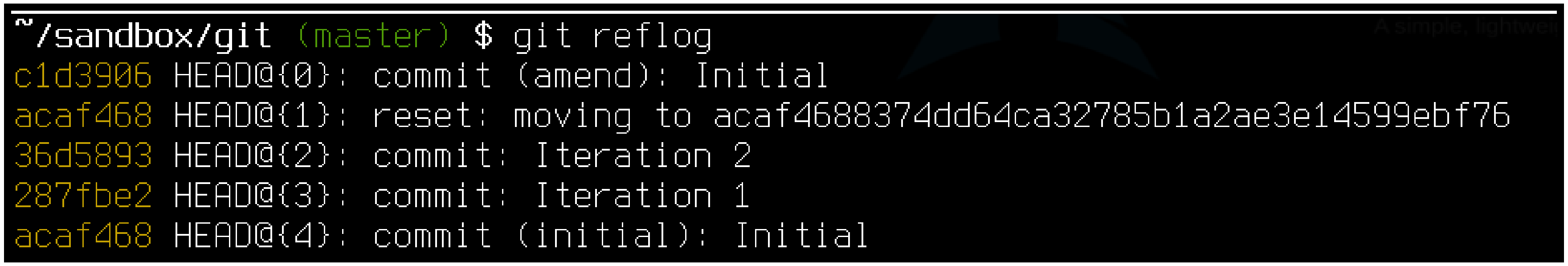
- One source to rule all state
- The events cannot lie, it happened, deal with it

Your version control is event sourced



```
~/sandbox/git (master) $ git log
commit c1d39064d256dbe4afff8d33995d5c9e26ad7710
Author: Willem-Jan <wjzijderveld@gmail.com>
Date:   Mon Jan 12 23:00:14 2015 +0100

    Initial
```



```
~/sandbox/git (master) $ git reflog
c1d3906 HEAD@{0}: commit (amend): Initial
acaf468 HEAD@{1}: reset: moving to acaf4688374dd64ca32785b1a2ae3e14599ebf76
36d5893 HEAD@{2}: commit: Iteration 2
287fbe2 HEAD@{3}: commit: Iteration 1
acaf468 HEAD@{4}: commit (initial): Initial
```

Sure thing, but how?

Well.. that is a complicated question

Event Sourcing + CQRS + DDD

That is a lot to cover in one evening

But each can be used on its own

Domain Driven Design

The business should understand our events

Aggregate root

Responsible for keeping a group of entities consistent

Something happened…
now what?

Record the change

DomainMessage

A message to tell your application what happened

DomainMessage

- Identifier
- Sequencenumber
- Event
- Timestamp
- Metadata

Identifier + sequencenumber

Event

It tells you WHAT happened

The name should be descriptive

Should contain everything

It should only depend on previous events

```
final class CompanyRegisteredEvent
{
  private $companyId;
  private $companyName;

  // constructor + getters
}
```

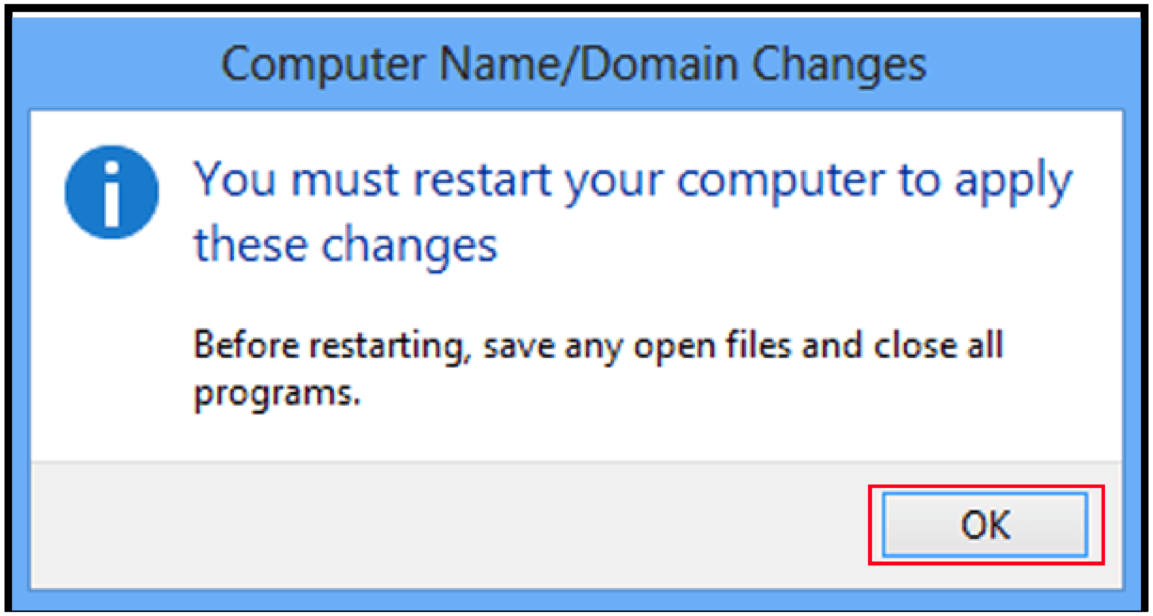
Timestamp

It tells you WHEN it happened

Metadata

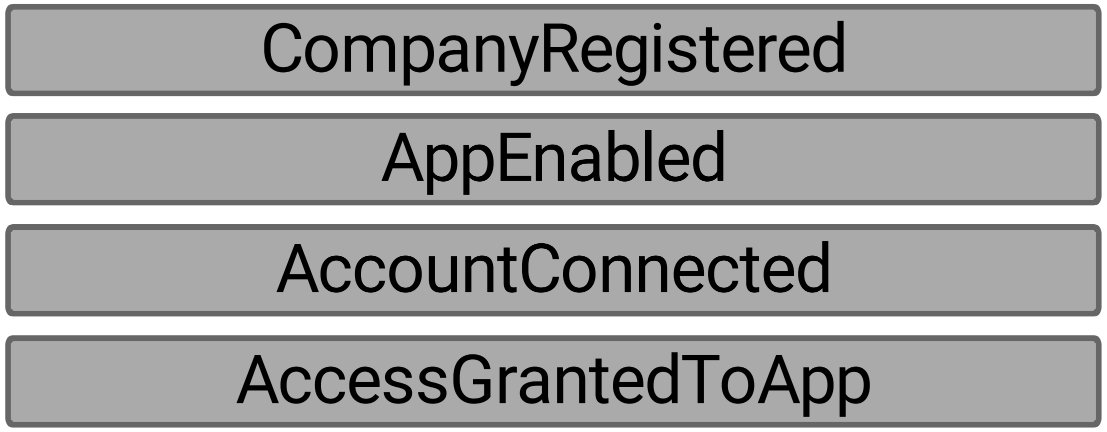
Descriptive, not structural

Apply the event


Computer Name/Domain Changes

You must restart your computer to apply these changes

Before restarting, save any open files and close all programs.


CompanyRegistered
AppEnabled
AccountConnected
AccessGrantedToApp

CompanyRegisteredEvent

```php
// Company
function applyCompanyRegisteredEvent(CompanyRegisteredEvent $event)
{
  $this->companyId = $event->getCompanyId();
}
```

AccountConnectedEvent

```php
// Company
function applyAccountConnectedEvent(AccountConnectedEvent $event)
{
  $this->accounts[$event->getAccountId()] = $event->getAccountId();
}
```

AppEnabledEvent

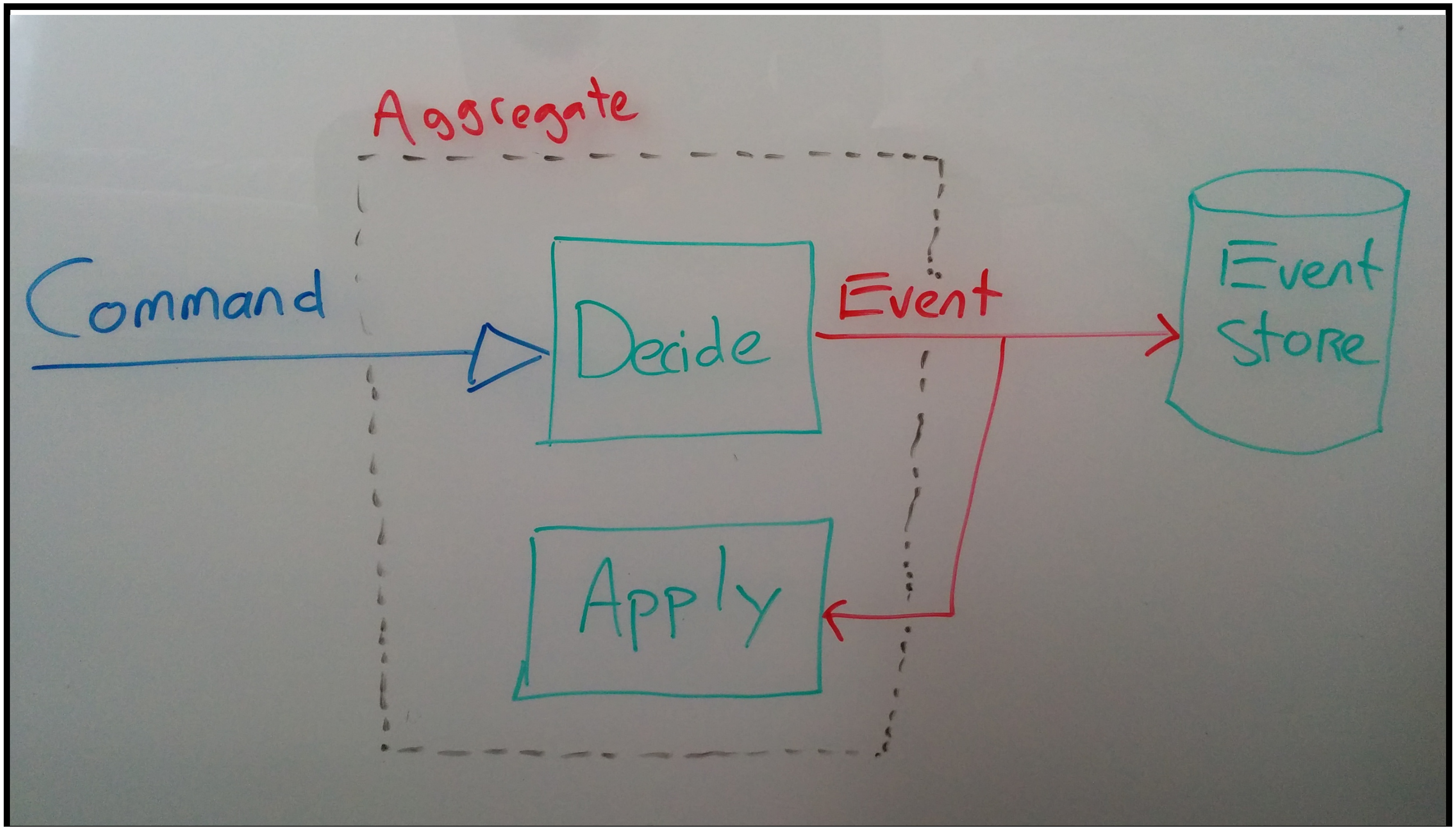
```php
// Company
function applyAppEnabledEvent(AppEnabledEvent $event)
{
  $subscription = new Subscription($this->companyId, $event->getAppId());
  $this->subscriptions[$event->getAppId()] = $subscription;
}
```

AccessGrantedToAppEvent

```
// Company
public function getChildEntities()
{
  return $this->subscriptions;
}
```

```
// Subscription
function applyAccessGrantedToAppEvent(AccessGrantedToAppEvent $event)
{
  if ($this->appId !== $event->getAppId()) {
    return;
  }

  $this->grantedAccounts[$event->getAccountId()] = $event->getAccountId();
}
```


Aggregate
Command
Decide
Event
Event Store
Apply

From Controller to CommandHandler

```php
class CompanyController
{
  function createAction(Request $request)
  {
    $this->commandBus->dispatch(new RegisterCompanyCommand(
      new CompanyId(Uuid::uuid4()),
      new CompanyName($request->request->get('companyName'))
    ));
  }
}
```

From CommandHandler to Aggregate

```php
class CompanyCommandHandler
{
  function handleRegisterCompanyCommand(RegisterCompanyCommand $command)
  {
    $company = Company::register(
      $command->getCompanyId(),
      $command->getCompanyName()
    );

    $this->aggregateRepostitory->save($company);
  }
}
```

From Aggregate to Event

```php
class Company extends EventSourcedAggregateRoot
{
  public static function register(CompanyId $companyId, CompanyName $name)
  {
    $company = new Company();
    $company->apply(new CompanyRegisteredEvent($companyId, $name));

    return $company;
  }

  public function applyCompanyRegisteredEvent(/**/ $event)
  {
    $this->companyId = $event->getCompanyId();
  }
}
```

Scenario based testing

```
$this->scenario
  ->given([
    new CompanyRegisteredEvent(new CompanyId(123))
  ])
  ->when(function ($company) {
    $company->enableApp(new AppId(42));
  })
  ->then([
    new AppEnabledEvent(new AppId(42), new CompanyId(123))
  ]);
```

How to create a **list of companies**

Creating read models

Listen to the events

CompanyRegistration

```php
class CompanyRegistration implements ReadModel
{
  private $companyId;
  private $companyName;
  private $registeredOn;

  public function __construct(
    CompanyId $companyId,
    CompanyName $companyName,
    DateTime $dateTime
  ) {
    // ..
  }
}
```

CompanyRegistrationProjector

```php
class CompanyRegistrationProjector
{
  public function applyCompanyRegisteredEvent(
    CompanyRegisteredEvent $event,
    DomainMessage $domainMessage
  ) {
    $company = new CompanyRegistration(
      $event->getCompanyId(),
      $event->getCompanyName(),
      $domainMessage->getRecordedOn()
    );

    $this->repository->save($company);
  }
}
```

Combine **different** read model repositories

Use the right tool for the right job

Another scenario test

```php
class CompanyRegistrationProjectorTest
{
  public function it_creates_a_company_registration()
  {
    $this->scenario
      ->given([])
      ->when(new CompanyRegistered('123', 'Acme Inc.'), $dateTime)
      ->then([new CompanyRegistration('123', 'Acme Inc.', $dateTime)]);
  }
}
```

Possibilities are endless

The ability to create multiple read models

- List of company registrations
- Graph of all connections between companies and accounts
- Creating reports about the amount of revocations

Time travel **is** possible!

Use events you recorded to create a new report multiple years after the fact

You **made a mistake** in a projection?

So what? Correct your projector and recreate your read model from your event stream

Questions?

joind.in/13466

@willemjanz

wjzijderveld@gmail.com

Freenode: #qandidate

More information

- http://codebetter.com/gregyoung/2010/02/20/why-use-event-sourcing/
- http://codebetter.com/gregyoung/2010/02/13/cqrs-and-event-sourcing/
- http://martinfowler.com/eaaDev/EventSourcing.html
- http://martinfowler.com/bliki/CQRS.html
- http://www.axonframework.org/docs/2.3/domain-modeling.html
- http://labs.qandidate.com/