Microservices within a Monolith

Join at slido.com #devdays2019

@jlammerts

Joop Lammerts

Developer @procurios for +3 years

@jlammerts

Procurios Cluster

Procurios Cluster

for context


Our Monolith

Our monolith

~ backend:
3.000.000 lines of code distributed over
18.000 PHP files

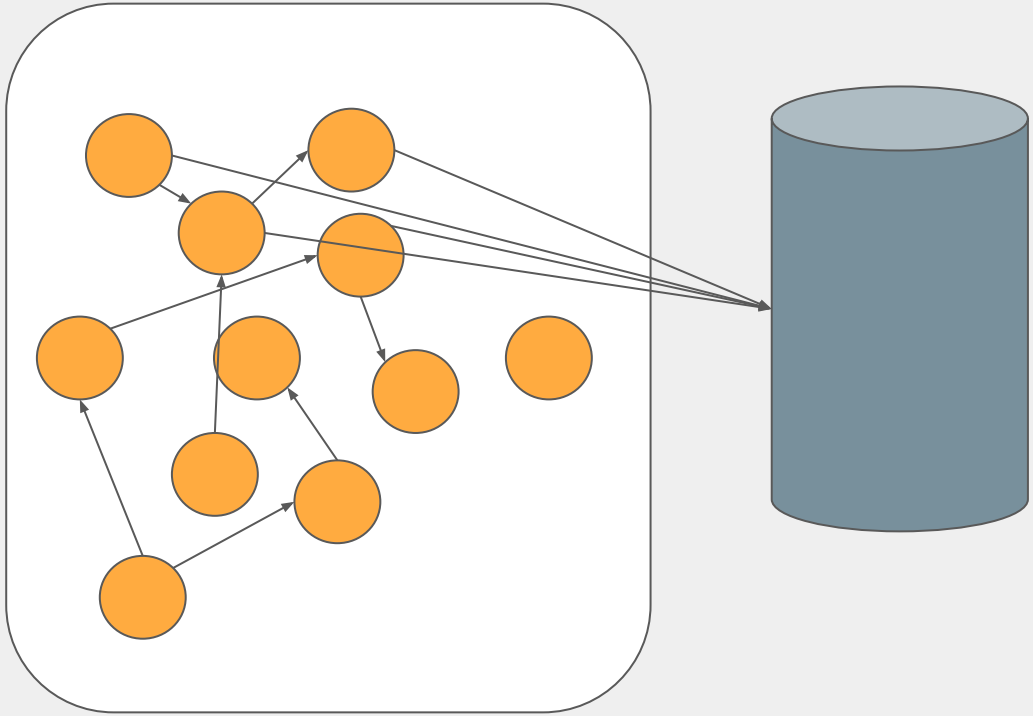
~ frontend:
350.000 lines in 1800 JavaScript files
800.000 lines of CSS code

Usage

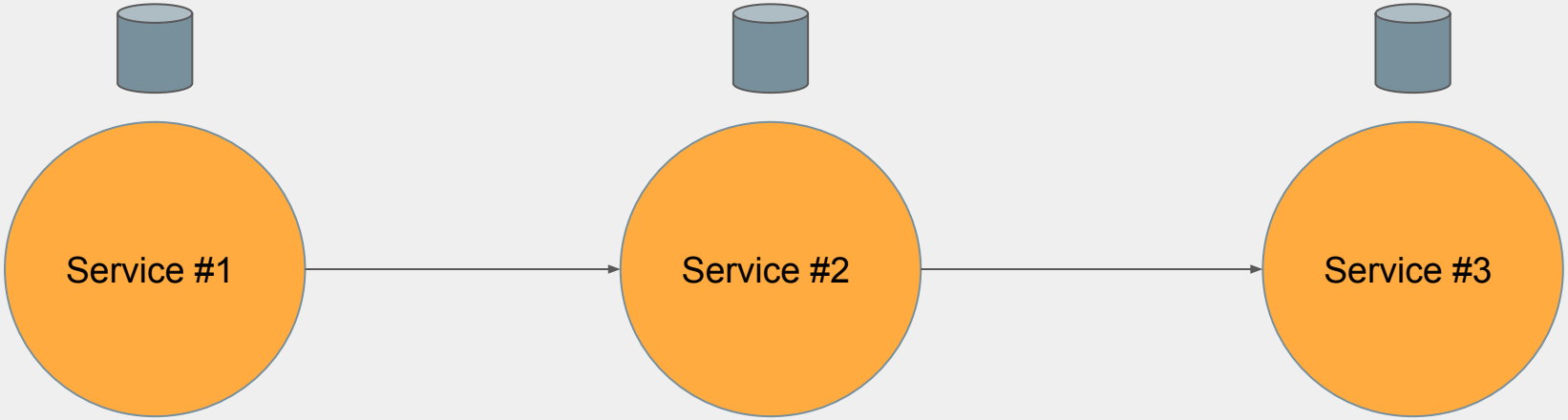
Usage

- 2000 clients
- 800.000 users
- 500.000 visitors an hour

Monolith



Microservices


Service #1
Service #2
Service #3

Microservices, or microservice architecture, is an approach to application development in which a large application is built as a suite of modular components or services.

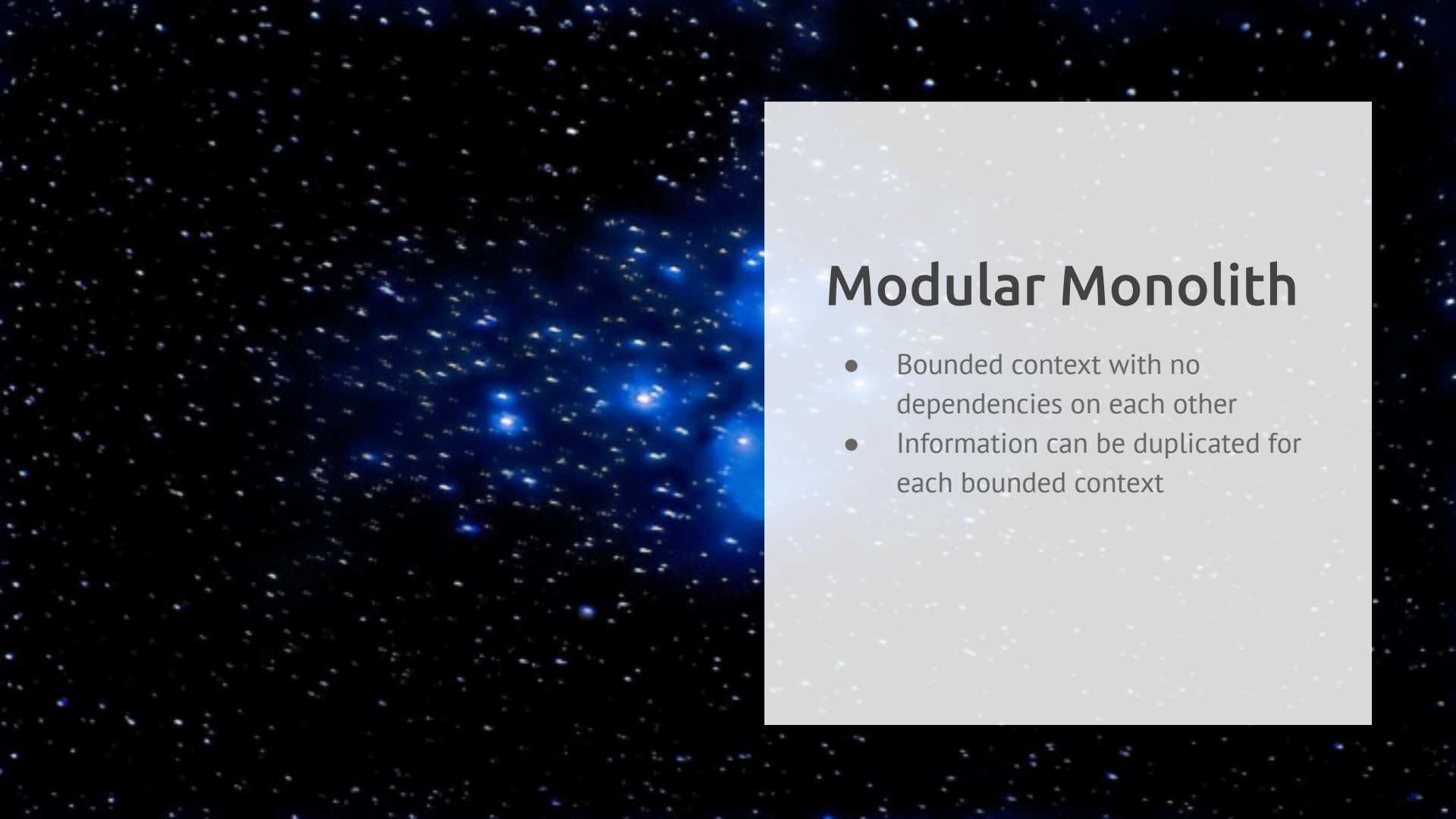
Assumption

Microservices, or microservice architecture, is an approach to application development in which a large application is built as a suite of modular components or services.

Assumption

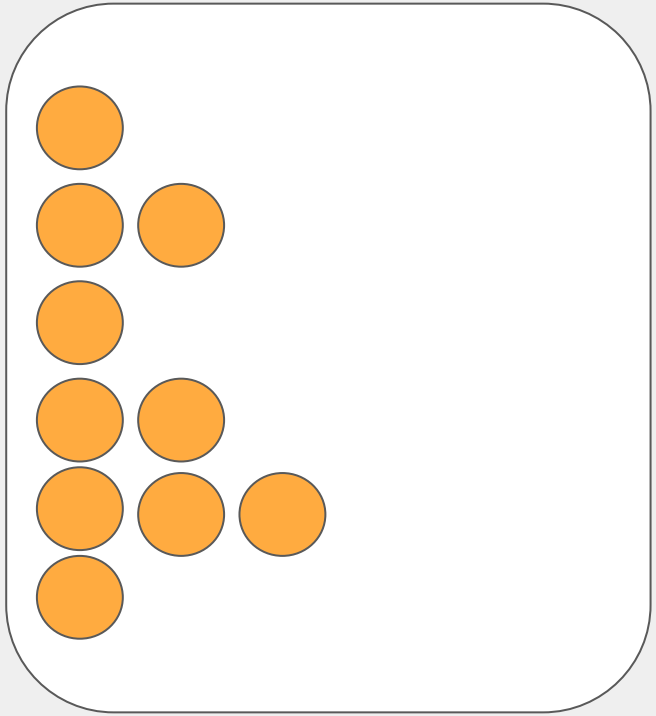
Modules

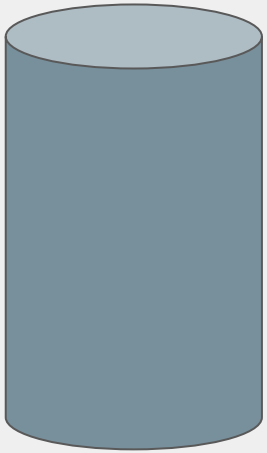
Modular Monolith



Modular Monolith

- Bounded context with no dependencies on each other
- Information can be duplicated for each bounded context





What to do with your legacy Monolith?

```
Project
|--- core
|--- modules
     |--- cms
     |--- meeting
     |--- relation
     |--- user
```

```
Project
|--- core
|--- modules
     |--- cms
     |--- meeting
          -> attendee
          -> meeting
          -> registration
          -> ticket
     |--- relation
     |--- user
```

```php
final class RegistrationController
{
  public function register()
  {
    $userId = $_SESSION['userId'];
    $pdo = new \PDO('localhost');
    $statement = $pdo->prepare('
      SELECT * FROM `user`
      WHERE `id` = ?
    ');
    $statement->execute([$userId]);
    $userData = $statement->fetch()[0];
    if (!$userData) {
      HttpResponse::redirect('/login');
    }

    $form = $this->getRegistrationForm();
    $data = $form->getData();
    if (!$data['meetingId'] || !$data['ticketId'] || !$data['remark']) {
      return $form;
    }

    $statement = $pdo->prepare('
      UPDATE `tickets`
      SET `sold` = 1
      WHERE WHERE `id` = ?
    ');
    $statement->execute([
      $data['ticketId']
    ]);

    if ($statement->rowCount() !== 1) {
      return 'There are no tickets available';
    }

    $statement = $pdo->prepare('
      INSERT INTO `attendee` SET
        `user_id` = ?,
        `first_name` = ?,
        `last_name` = ?,
        `meeting_id` = ?,
        `ticket_id` = ?,
        `remark` = ?,
    ');
    $statement->execute([
      $userData['id'],
      $userData['firstName'],
      $userData['name'],
      $data['meetingId'],
      $data['ticketId'],
      $data['remark'],
    ]);

    /*
     * send confirmation stuff
     */

    \HttpResponse::redirect('/');
  }
}
```

```php
namespace Meeting\Registration;

final class RegistrationController
{
    public function register()
    {
        $userId = $_SESSION['userId'];
        $pdo = new \PDO('localhost');
        $statement = $pdo->prepare("
            SELECT * FROM `user`
            WHERE `id` = ?
        ");
        $statement->execute([$userId]);
        $userData = $statement->fetch()[0];
        if (!$userData) {
            HttpResponse::redirect('/login');
        }

        //
}
```

```
namespace Meeting\Registration;

final class RegistrationController
{
    public function register()
    {
        $user = UserService::getCurrentUser();
        if (!$user->isAuthenticated()) {
            HttpResponse::redirect('/login');
        }

        //
}
```

```php
namespace User;

final class UserService
{
  public static function getCurrentUser(): User
  {
     $userId = $_SESSION['userId'] ?? null;
     if ($userId === null) {
        return User::guest();
     }

     $pdo = new \PDO('localhost');
     $statement = $pdo->prepare("
        SELECT * FROM `user`
        WHERE `id` = ?
     ");
     $statement->execute([$userId]);
     $userData = $statement->fetch()[0];
     return $userData
          ? User::populate($userData);
          : User::guest();
   }
```

User is now a bounded context

```
final class RegistrationController
{
  public function register()
  {
    $user = UserService::getCurrentUser();
    if (!$user->isAuthenticated()) {
      HttpResponse::redirect('/login');
    }

    $form = $this->getRegistrationForm();
    $data = $form->getData();
    if (!$data['meetingId'] || !$data['ticketId'] || !$data['remark']) {
      return $form;
    }

    $statement = $pdo->prepare("
      UPDATE `tickets`
      SET `sold` = 1
      WHERE WHERE `id` = ?
    ");
    $statement->execute([
      $data['ticketId']
    ]);

    if ($statement->rowCount() !== 1) {
      return 'There are no tickets available';
    }

    $statement = $pdo->prepare("
      INSERT INTO `attendee` SET
        `user_id` = ?,
        `first_name` = ?,
        `last_name` = ?,
        `meeting_id` = ?,
        `ticket_id` = ?,
        `remark` = ?,
    ");
    $statement->execute([
      $userData['id'],
      $userData['firstName'],
      $userData['name'],
      $data['meetingId'],
      $data['ticketId'],
      $data['remark'],
    ]);

    /*
     * send confirmation stuff
     */

    \HttpResponse::redirect('/');
  }
}
```

But wait, there is more!

```php
public function register()
 {
    //
    $form = $this->getRegistrationForm();
    $data = $form->getData();
    if (!$data['meetingId'] || !$data['ticketId'] || !$data['remark']) {
        return $form;
    }

   $statement = $pdo->prepare("
        /* */
   ");
   $statement->execute([
        $data['ticketId']
    ]);
    if ($statement->rowCount() !== 1) {
        return 'There are no tickets available';
    }
   //
}
```

```php
public function register()
 {
    //
    $form = $this->getRegistrationForm();
    $data = $form->getData();
    if (!$data['meetingId'] || !$data['ticketId'] || !$data['remark']) {
        return $form;
    }

    try {
        $ticket = TicketService::purchase($data['ticketId']);
    } catch (CouldNotPurchaseTicket $e) {
        return 'There are no tickets available';
    }
   //
}
```

```php
final class TicketService
{
    public static function purchase(int $ticketId): Ticket
    {
        $connection = DB::getConnection();
        $statement = $connection->prepare("
            /* */
        ");
        $statement->execute([
            $data['ticketId']
        ]);

        if ($statement->rowCount() !== 1) {
            throw CouldNotPurchaseTicket::becauseNoTicketsLef();
        }

        return new Ticket($ticketId);
    }
}
```

```php
public function register()
 {
    //
    $statement = $pdo->prepare("
      /* */
    ");
    $statement->execute([
      $user->getId(),
      $user->getFirstName(),
      $user->getName(),
      $data['meetingId'],
      $data['ticketId'],
      $data['remark'],
    ]);
   //
}
```

```php
public function register()
 {
    //
    $attendee = new Attendee($user->getId(), $user->getFirstName(),  $user->getName());
    $registration = new Registration($data['meetingId'], $attendee, $ticket, $data['remark']);
    RegistrationService::register($registration);

   /*
    * send confirmation stuff
    */

  HttpResponse::redirect('/');
}
```

Modules sent messages

```php
final class RegistrationService
{
    public static function register(Regitstation $registration): void
    {
        $repository = self::getRepository();
        $repository->save($registration);

        $registrationCreated = new RegistrationCreated($registration);

        foreach (self::getListenerProvider()->getListenersForEvent($registrationCreated) as $listener) {
            $listener->handle($registrationCreated);
        }
    }
}
```

```
final class NotifyAttendee
{
  public function handle(RegistrationCreated $registrationCreated): void
  {
     $registration = $registrationCreated->getRegistration();
     mail(
        $registration->getEmailAddressAsString(),
       'See you soon',
       sprintf(
'Hi %s<br />
See you soon at %s
',
       $registration->getFirstName(),
       $registration->getMeetingName()
     );
  }
}
```

```
final class RegistrationController
{
  public function register()
  {
    $user = UserService::getCurrentUser();
    if (!$user->isAuthenticated()) {
      HttpResponse::redirect('/login');
    }

    $form = $this->getRegistrationForm();
    $data = $form->getData();
    if (!$data['meetingId'] || !$data['ticketId'] || !$data['remark']) {
      return $form;
    }

    try {
      $ticket = TicketService::purchase($data['ticketId']);
    } catch (CouldNotPurchaseTicket $e) {
      return 'There are no tickets available';
    }

    $attendee = new Attendee($user->getId(), $user->getFirstName(),  $user->getName());
    $registration = new Registration($data['meetingId'], $attendee, $ticket, $data['remark']);
    RegistrationService::register($registration);

    HttpResponse::redirect('/');
  }
}
```

But wait, there is more!

```
final class NotifyAttendee
{
  public function handle(RegistrationCreated $registrationCreated): void
  {
     $registration = $registrationCreated->getRegistration();
     mail(
        $registration->getEmailAddressAsString(),
       'See you soon',
       sprintf(
'Hi %s<br />
See you soon at %s
',
       $registration->getFirstName(),
       $registration->getMeetingName()
     );
  }
}
```

```php
final class NotifyAttendeeByMandrill
{
    public function handle(RegistrationCreated $registrationCreated): void
    {
        $registration = $registrationCreated->getRegistration();
        $mandrill new Mandrill('/* */');

        $mandrillMessage = [
            /* */
        ];

     try {
        $mandrill->messages->send($mandrillMessage);
    } catch(Mandrill_Error $e) {
         //
    }
}
```

Oh no! The service is down

```php
final class NotifyAttendeeJob
{
    public function __construct(AsyncMessageBus $bus)
    {
        $this->bus = $bus;
    }

    public function handle(RegistrationCreated $registrationCreated): void
    {
        $registration = $registrationCreated->getRegistration();
        $message new SendConfimationMessage($registration);

        $this->bus->send($message);
    }
```

```php
final class SendConfimationMessageWorker
{
   public function handle(SendConfimationMessage $message): void
   {
      $mandrill new Mandrill('/* */');

      $mandrillMessage = [
          /* */
      ];

    try {
      $mandrill->messages->send($mandrillMessage);
      $mandrillMessage->markSend();
    } catch(Mandrill_Error $e) {
        //
    }
}
```

Takeaways

Locate and isolate Bounded contexts and turn them into modules

Let the world know what changed in an Event Driven plugin architecture

Procurios

Joop Lammerts

Website: www.procurios.com
Twitter: @jlammerts