



Rétrospective PoC architecture

Exploration MVP, ValueObject, CQRS...



Introduction

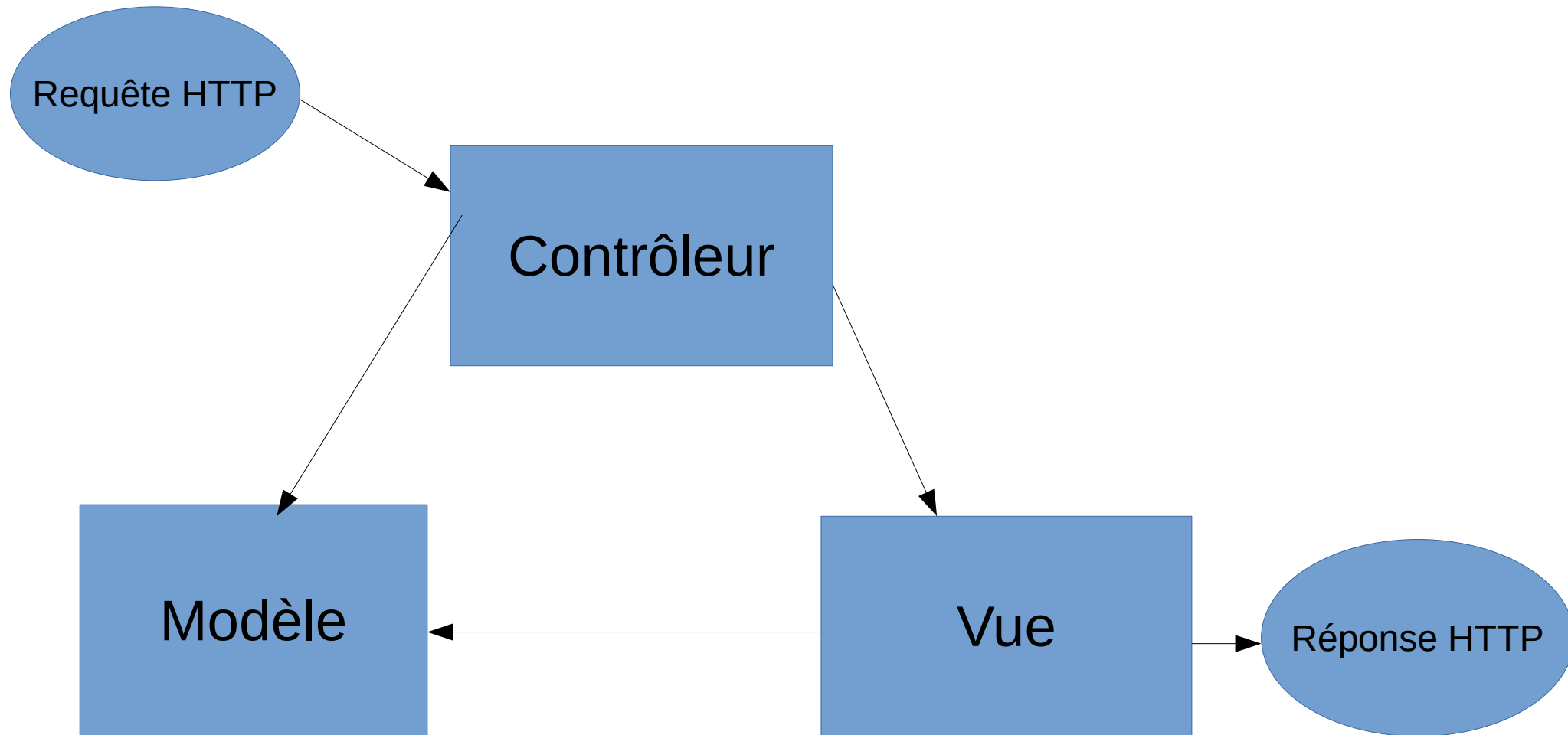
- Pas un cours magistral
- Simple aperçu de concepts d'architecture
- PoC application stateless
- Tests de technologies



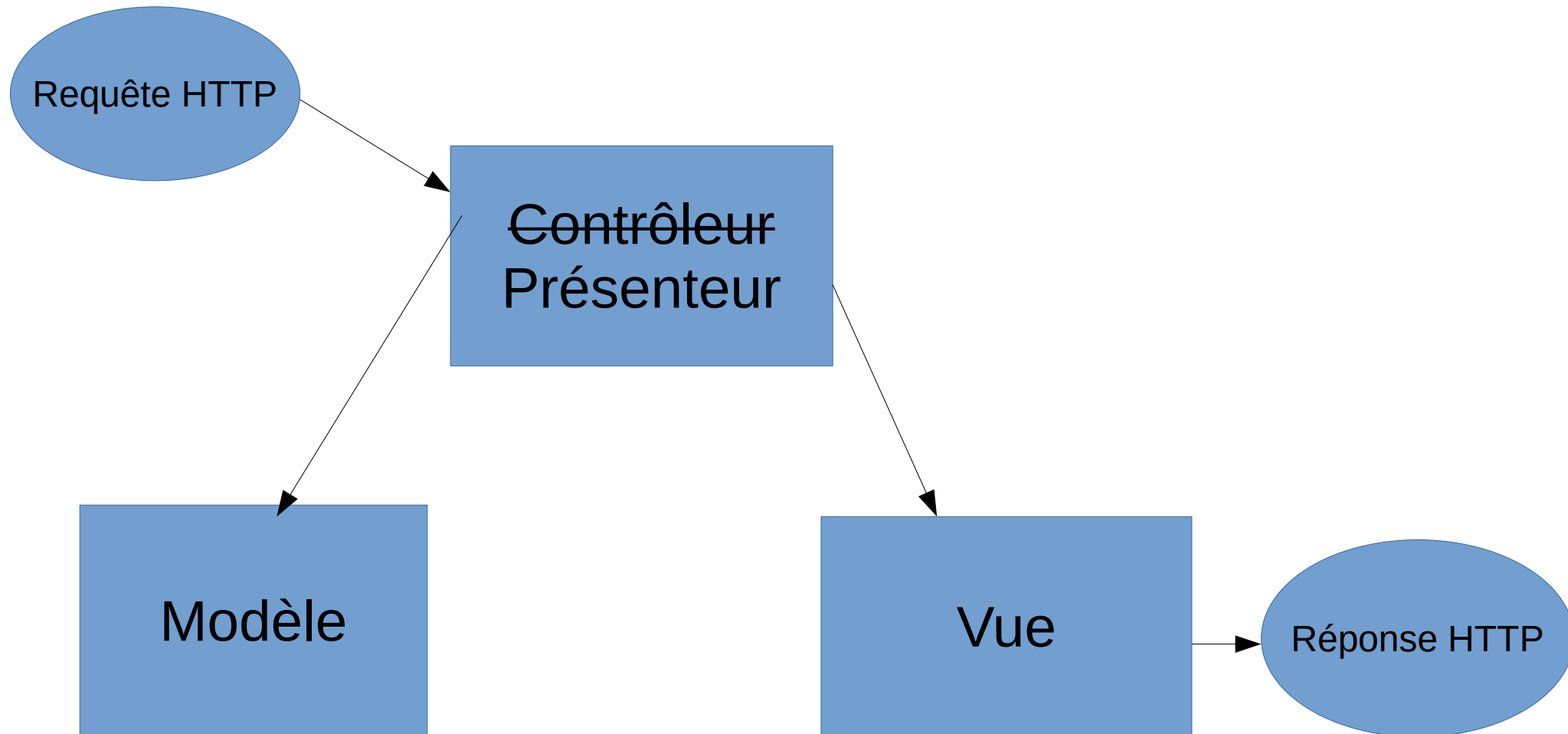
Sommaire

- Architecture MVP
- ValueObject
- Immutabilité et flow typesafe
- CQRS
- Architecture hexagonale
- Workerman et OpenSwoole

Architecture MVC



Architecture MVP





Architecture MVP

- Le présenteur a le rôle du Contrôleur
- Le lien entre vue et modèle est supprimé
- Le présenteur formate la donnée du modèle pour la vue
- Données sont formatées avant d'être passé à la vue
- La vue n'a pas accès aux entités



Exemple MVC/MVP

- Affiche les messages d'un chat
- Une liste de messages est donc chargée
- La liste des auteurs aussi

Exemple MVC - Contrôleur

```
/**
 * @param ShowChatRequest $request
 */
/* Vincent QUATREVIEUX */
public function handle(object $request): object
{
    $messages = $this->repository->all();
    $usersIds = [];

    foreach ($messages as $message) {
        if ($message->userId !== null) {
            $usersIds[$message->userId->value] = $message->userId->value;
        }
    }

    $users = $this->userRepository->findAllById(array_values($usersIds));

    return $this->render('chat/show', [
        'messages' => $messages,
        'users' => $users,
        'me' => $request->user,
    ]);
}
```


Exemple MVC - Vue

```
<div id="chat">
  <h1>Chat</h1>

  <section
    class="messages"
    data-source="<?=$renderer->url(ShowChatRequest::ajax()) ?>"
    data-events="<?=$renderer->url(new SubscribeEventRequest()) ?>"
    data-pulling-delay="60000"
  >
    <?php foreach($this->messages as $message): ?>
      <div class="message <?=$this->me && $message->userId->id->value == $this->me->id->value ? 'from-me' : 'from-other' ?>">
        <span class="author">
          <?php $user = $message->userId ? $this->users[$message->userId->value] ?? null : null; ?>
          <?=$user->html() ?? 'Deleted' ?>
        </span>
        <span class="date"><?=$message->createdAt->format('d/m/Y H:i:s') ?></span>
        <span class="content"><?=$message->message->html() ?></span>
      </div>
    <?php endforeach; ?>
  </section>

  <?php if ($renderer->hasAccess($sendMessage = new SendMessageRequest())): ?>
    <section class="input">
      <form action="<?=$renderer->url($sendMessage) ?>" method="post">
        <input type="text" name="message" placeholder="Message" autofocus />
        <input type="submit" value="Envoyer" />
      </form>
    </section>
  <?php endif ?>
</div>
```

Exemple MVP - Présenteur

```

    Vincent QUATREVIEUX
    public function handle(object $request): ShowChatResponse
    {
        $messages = $this->repository->all();
        $usersIds = [];

        foreach ($messages as $message) {
            if ($message->userId !== null) {
                $usersIds[$message->userId->value] = $message->userId->value;
            }
        }

        $users = $this->userRepository->findAllById(array_values($usersIds));

        return new ShowChatResponse(
            messages: array_map(
                function ($message) use ($users, $request) {
                    if ($message->userId) {
                        $user = $users[$message->userId->value] ?? null;
                    } else {
                        $user = null;
                    }

                    return new ChatMessageWithUser(
                        id: $message->id,
                        message: $message->message,
                        createdAt: $message->createdAt,
                        isMine: $message->userId && $message->userId == $request->user->id,
                        pseudo: $user->pseudo,
                    );
                },
                $messages,
            ),
            ajax: $request->ajax,
        );
    }
}

```

Exemple MVP – Modèle de vue

```
5 usages  👤 Vincent Quatreveux
class ShowChatResponse
{
    1 usage  👤 Vincent Quatreveux
    public function __construct(
        /**
         * @var ChatMessageWithUser[]
         */
        public readonly array $messages,
        public readonly bool $ajax = false,
    ) {
    }
}
```

```
2 usages  👤 Vincent QUATREVIEUX
class ChatMessageWithUser
{
    1 usage  👤 Vincent QUATREVIEUX
    public function __construct(
        public readonly ChatMessageId $id,
        public readonly MessageContent $message,
        public readonly DateTimeImmutable $createdAt,
        public readonly bool $isMine,
        public readonly ?Pseudo $pseudo,
    ) {
    }
}
```

Exemple MVP - Vue

```
<div id="chat">
  <h1>Chat</h1>

  <section
    class="messages"
    data-source="<?= $renderer->url(ShowChatRequest::ajax()) ?>"
    data-events="<?= $renderer->url(new SubscribeEventRequest()) ?>"
    data-pulling-delay="60000"
  >
    <?php foreach($this->messages as $message): ?>
      <div class="message <?= $message->isMine ? 'from-me' : 'from-other' ?>">
        <span class="author"><?= $message->pseudo->html() ?? 'Deleted' ?></span>
        <span class="date"><?= $message->createdAt->format('d/m/Y H:i:s') ?></span>
        <span class="content"><?= $message->message->html() ?></span>
      </div>
    <?php endforeach; ?>
  </section>

  <?php if ($renderer->hasAccess($sendMessage = new SendMessageRequest())): ?>
    <section class="input">
      <form action="<?= $renderer->url($sendMessage) ?>" method="post">
        <input type="text" name="message" placeholder="Message" autofocus />
        <input type="submit" value="Envoyer" />
      </form>
    </section>
  <?php endif ?>
</div>
```



Avantages MVP

- Vue plus simple : les données sont préformatées pour la vue
- Indépendance métier et vue
- Évolutivité : une modification du modèle n'implique pas de modification de la vue
- Cohérence de la donnée grâce au typage



Inconvénients MVP

- Plus de classes :au moins une classe de réponse par contrôleur
- Performance : donnée dupliquée
- Plus de code dans le contrôleur



ValueObject

- Un objet immutable
- Contient une valeur primitive
- Valeur validée au constructeur
- Toutes les propriétés des entités doivent être un value object
- Ajoute une sémantique à la valeur

ValueObject - Example

```
final class UserId implements ValueObjectInterface, JsonSerializerizable
```

```
{
```

```
    ⤴ Vincent Quatreveux *
```

```
    public function __construct(
```

```
        public readonly int $value,
```

```
    )
```

```
    {
```

```
        if ($value < 1) {
```

```
            throw new InvalidValueException(self::class, 'User id must be greater than 0');
```

```
        }
```

```
    }
```

```
no usages    ⤴ Vincent Quatreveux
```

```
    public function jsonSerialize(): int
```

```
    {
```

```
        return $this->value;
```

```
    }
```

```
    ⤴ Vincent Quatreveux
```

```
    public function value(): int
```

```
    {
```

```
        return $this->value;
```

```
    }
```

```
    ⤴ Vincent Quatreveux *
```

```
    public function __toString(): string
```

```
    {
```

```
        return (string) $this->value;
```

```
    }
```

```
    ⤴ Vincent Quatreveux *
```

```
    public static function from(mixed $value): static
```

```
    {
```

```
        if (!is_int($value)) {
```

```
            throw new InvalidPrimitiveTypeError(self::class, 'int', $value);
```

```
        }
```

```
        return new static($value);
```

```
    }
```

```
    ⤴ Vincent Quatreveux *
```

```
    public static function tryFrom(mixed $value): ?static
```

```
    {
```

```
        if (!is_int($value)) {
```

```
            return null;
```

```
        }
```

```
        try {
```

```
            return new static($value);
```

```
        } catch (ValueObjectException) {
```

```
            return null;
```

```
        }
```

```
    }
```

```
}
```


ValueObject - Entité

```
16 usages  👤 Vincent QUATREVIEUX
final class ChatMessage
{
    2 usages  👤 Vincent QUATREVIEUX
    public function __construct(
        public readonly ChatMessageId $id,
        public readonly MessageContent $message,
        public readonly ?UserId $userId,
        public readonly DateTimeImmutable $createdAt,
    ) {
    }
}
```



ValueObject - Utilité

- Cohérence des données garantie (ex : un email est forcément un email)
- Le type reflète la donnée (ex : un UserId vient de l'entité User)
- Impossibilité de jongler entre 2 données incompatibles
- Helper method directement sur le value object

ValueObject - Utilisation

- Ajoute contrainte fonctionnelle et pas métier
- La valeur doit être validée avant de créer le VO
- Pas de try/catch (VO ne remplace pas un form)
- Pas de VO sur les form
- Faut-il en utiliser niveau paramètre repository ?
(ex : findById prend UserId ou int ?)



ValueObject - Inconvénients

- Prise en compte sur l'ORM
- Impact sur les performances
- Fait planter l'application si db corrompue
- Plus de code pour déclarer les VO et convertir les primitives en VO

Entités immutables

- Toutes les propriétés sont readonly
- Pas d'objet « partiels » : toutes les propriétés non optionnelles ont une valeur
- Toutes modifications retournent une nouvelle instance
- Les données « incomplètes » ou décrivant un état doivent avoir leur propre classe
- Méthodes pour passer d'un « état » à un autre

Immutabilité – Création User

- Classe UserCreation avec les données sans id
- Méthode created qui prend id en paramètre
- Méthode created appelée par repository après insert
- User représente un utilisateur valide en base

Immutabilité - Authentication

- Classe `AuthenticatedUser` qui représente le user authentifié
- Méthode `User::authenticate()`
- `AuthenticatedUser` garde que les données pertinentes

Immutability - Modification

- Classe ModifiedUser qui représente un User modifié par encore sauvegardé
- Méthodes withXXX qui créent une nouvelle instance avec nouvelle valeur
- Ces méthodes gardent un trace des propriétés modifiées
- Repository lit cette trace pour faire l'update + appelle save() qui retourne un User valide

Immutabilité – Avantages

- Limite les effets de bords
- Typage empêche entité invalide dans métier
- Flow clair grâce aux nom de classes
- Entités plus simple (pas besoin de getter ou setter grâce à readonly)




Immutabilité - Inconvénients

- Léger impact sur les performances
- Compatibilité avec ORM et form
- Maintenir les différentes classes de flow : leurs propriétés doivent rester cohérentes entre elles



CQRS - Introduction

- Séparation entre lecture (query) et écriture (command)
- Nécessite pas un bus
- Simple séparation des interface pour lire et écrire suffit
- Permet d'avoir un backend différent pour l'écriture et lecture (ex : elasticsearch en lecture)



CQRS - Query

- Pas de nécessité à faire des classes de query
- Simple interface avec liste des méthodes de lecture
- Implémentation du dépôt de lecture peut être séparé de celui d'écriture (non obligatoire)
- Utiliser UNIQUEMENT l'interface niveau métier et contrôleurs



CQRS - Command

- Toutes les écritures doivent passer par une commande
- Une interface pour écriture sur dépôt doit être créée
- Une classe par commande
- Seul les handlers ont accès au dépôt d'écriture
- Données doivent être validées avant de créer la commande
- Handler doit pouvoir retourner un résultat (cas contrainte unique)



CQRS - Avantages

- Évolution vers SQL+NoSQL simple
- Actions métiers visibles et claire avec commands
- Gestion de la donnée cachée au métier
- Simplification du contrôleur / limite de besoin de classes de « service »
- Facile à migrer vers ce modèle



CQRS - Inconvénients

- Nouvelle façon de faire
- Commandes peuvent rendre la navigation dans le code plus complexe
- Gestion des erreurs en asynchrone



Architecture hexagonale

- Indépendance aux services externes
- Indépendance au framework
- Interface pour définir API et DI pour faire la liaison (« ports »)



Architecture hexagonale

- Utiliser interfaces dépôts CQRS
- Classe requête + réponse
- Implémentations dans dossier Infrastructure



Serveur HTTP en PHP

- Similaire à express JS
- Runtime long : pas besoin de redémarrer l'application à chaque requête
- Traitement asynchrone sans queue
- Ne passe pas par apache/nginx



Workerman

- Écrit en PHP (pas besoin d'extensions)
- Supporte http, websocket, TCP, UDP
- Multiprocessus + Timers
- Excellentes performances avec le JIT
- Doc en Chinois
- Pas de gestion de l'asynchrone

Workerman - Benchmark

```
Benchmarking FPM + Nginx
wrk -c 64 -t 8 -d 15s 'http://127.0.0.1:8401/fpm/search?autocomplete=0&query=sql'
Running 15s test @ http://127.0.0.1:8401/fpm/search?autocomplete=0&query=sql
 8 threads and 64 connections
  Thread Stats   Avg    Stdev    Max   +/-  Stdev
   Latency    55.86ms   18.70ms  115.65ms   82.38%
   Req/Sec   143.15    44.38   232.00   55.42%
17137 requests in 15.05s, 74.24MB read
Requests/sec: 1138.94
Transfer/sec:  4.93MB
```

```
Benchmarking Workerman + Nginx
wrk -c 64 -t 8 -d 15s 'http://127.0.0.1:8401/workerman/search?autocomplete=0&query=sql'
Running 15s test @ http://127.0.0.1:8401/workerman/search?autocomplete=0&query=sql
 8 threads and 64 connections
  Thread Stats   Avg    Stdev    Max   +/-  Stdev
   Latency    6.06ms    3.51ms   60.61ms   87.39%
   Req/Sec    1.41k    81.33    1.99k    73.33%
168122 requests in 15.03s, 725.99MB read
Requests/sec: 11183.28
Transfer/sec:  48.29MB
```

```
Benchmarking Workerman
wrk -c 64 -t 8 -d 15s 'http://127.0.0.1:8402/workerman/search?autocomplete=0&query=sql'
Running 15s test @ http://127.0.0.1:8402/workerman/search?autocomplete=0&query=sql
 8 threads and 64 connections
  Thread Stats   Avg    Stdev    Max   +/-  Stdev
   Latency    3.51ms    1.77ms   32.45ms   76.32%
   Req/Sec    2.33k    634.06    7.53k    62.03%
278597 requests in 15.10s, 1.16GB read
Requests/sec: 18451.71
Transfer/sec:  78.97MB
```



OpenSwoole

- Extension PHP
- Supporte http, websocket, MQTT, TCP/UDP
- Gestion de l'asynchrone + clients asynchrone
- Multiprocessus, coroutines, shared memory...
- Doc en anglais et populaire (bundle sf existant)
- Moins performant que Workerman

OpenSwoole - Benchmark

```
Benchmarking FPM + Nginx
wrk -c 64 -t 8 -d 15s 'http://127.0.0.1:8401/fpm/search?autocomplete=0&query=sql'
Running 15s test @ http://127.0.0.1:8401/fpm/search?autocomplete=0&query=sql
 8 threads and 64 connections
  Thread Stats   Avg      Stdev     Max   +/-  Stdev
    Latency    55.58ms   19.03ms  117.61ms   82.70%
    Req/Sec    143.96    43.61   222.00    55.00%
 17234 requests in 15.05s, 74.66MB read
Requests/sec: 1144.92
Transfer/sec: 4.96MB

Benchmarking OpenSwoole + Nginx
wrk -c 64 -t 8 -d 15s 'http://127.0.0.1:8401/openswoole/search?autocomplete=0&query=sql'
Running 15s test @ http://127.0.0.1:8401/openswoole/search?autocomplete=0&query=sql
 8 threads and 64 connections
  Thread Stats   Avg      Stdev     Max   +/-  Stdev
    Latency     5.92ms    1.05ms   40.81ms   95.40%
    Req/Sec     1.36k    69.21    2.38k    88.92%
 162767 requests in 15.05s, 704.11MB read
Requests/sec: 10815.54
Transfer/sec: 46.79MB
```



Questions ?